The Forge-and-Lose Technique and Other Contributions to Secure Two-Party Computation with Commitments

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical & Computer Engineering

Luís T. A. N. Brandão

Licenciado, Engenharia Física Tecnológica,
Instituto Superior Técnico, Universidade Técnica de Lisboa (Portugal)

Carnegie Mellon University
Pittsburgh PA, USA

June, 2017

The thesis was also submitted to Faculdade de Ciências, Universidade de Lisboa (Portugal), in partial fulfillment of the requirements for the degree of Doutor em Informática. The work was supported through the CMU|Portugal dual-degree doctoral program between Universidade de Lisboa and Carnegie Mellon University.

Page intentionally blank

The Forge-and-Lose Technique and Other Contributions to Secure Two-Party Computation with Commitments

Luís T. A. N. Brandão

Thesis defense on October 07, 2016

(internal text updated on December 23, 2016)

Thesis defense committee:

- Professor Alysson Bessani (co-advisor) — FCUL, Departamento de Informática
- Professor Manuel Blum — CMU, School of Computer Science
- Professor Nicolas Christin (co-advisor and chair) — CMU, Electrical & Computer Engineering
- Professor Limin Jia — CMU, Electrical & Computer Engineering
- Doctor Vladimir Kolesnikov — Nokia Bell-Labs
- Doctor René Peralta — NIST, Computer Security Division
- Professor André Souto — FCUL, Departamento de Informática

CMU = Carnegie Mellon University; FCUL = Universidade de Lisboa, Faculdade de Ciências;
NIST = National Institute of Standards and Technology

Submitted* to Faculdade de Ciências, Universidade de Lisboa,
in partial fulfillment of the requirements for the degree of
Doutor em Informática
(Lisboa, Portugal)

* (with a different set of title-pages)

*The thesis was also submitted in **June 2017** to the College of Engineering, Carnegie Mellon University (USA), in partial fulfillment of the requirements for the degree of Electrical & Computer Engineering. The work was supported through the CMU/Portugal dual-degree doctoral program between Universidade de Lisboa and Carnegie Mellon University.*

Page intentionally blank

Copyright notice

This dissertation contains material from:

- "Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique" (@ ASIACRYPT 2013) [Bra13]: extended abstract is subject to copyright © IACR 2013; full version is at the IACR Cryptology ePrint Archive, Report 2013/577; previous high-level related oral presentations made at the Eurocrypt 2013 and Crypto 2013 rump sessions.
- "Very-efficient simulatable flipping of many coins into a well (and a new universally-composable commitment scheme)" (@ PKC 2016) [Bra16]: extended abstract is subject to copyright © IACR 2016; full version is at the IACR Cryptology ePrint Archive, Report 2015/640; previous high-level related oral presentation made at the TCC 2014 rump session.
- "Toward mending two nation-scale brokered identification systems" (@ PETS 2015) [BCDA15] (co-authored with Nicolas Christin, George Danezis and an anonymous co-author): available under a Creative Commons Attribution-NonCommercial-NoDerivatives 3.0 License (CC BY-NC-ND 3.0) and subject to copyright "© Luís T. A. N. Brandão et al. 2015."

All material is included in this dissertation with permission, including in conformance with the Copyright and Publication Policy (2013) of IACR. It has been adjusted and restructured across different chapters, containing major revisions and further original contributions.

Contact: luis.papers+thesis@gmail.com

Page intentionally blank

Dedicated to my mother and my father

Page intentionally blank

Acknowledgments

Being a Ph.D. student in the CMU|Portugal program was an amazing opportunity — a long and enjoyable path, for its positive experiences and by appreciating the bright side of difficulties.

Affiliation and financial support. As a Ph.D. student at Faculty of Sciences, University of Lisbon (FCUL), and at Carnegie Mellon University (CMU), I was financially supported by a scholarship from the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program, under grant SFRH/BD/3370/2009. The support of living stipend and university tuitions was fundamental to empower me as a student non-employee, and for me that made the difference. As a Ph.D. student: at FCUL, department of Informatics, I was a member of LaSIGE/Navigators until September 2013; while at CMU I was at CyLab; I visited the Cryptography group at the National Institute of Standards and Technology as a foreign guest researcher in Summer 2012; my attendance in several conferences and workshops was supported in the way of registration-fee waivers and/or accommodation and/or traveling support, by funding made available to the respective events. I also did a paid internship at Bell Labs in Summer 2015.

I feel privileged for having been part of a variety of research settings, interacting with colleagues, across the years of Ph.D. This dissertation, besides being a result of technical research in secure computation and related topics, has inherently benefited from a multitude of enriching academic experiences — writing and reviewing papers, taking courses and teaching, attending and giving research presentations, organizing a few and participating in many events, having technical and philosophical discussions, among other research activities. A highlight was networking in cryptography and privacy related conferences, which helped me glimpse better the shape of the research community, notably including the IACR. Knowing that the academic structures and events are supported by long-standing collective efforts, I am grateful for the opportunity to be part of it and thankful to those who make it happen. A special note of appreciation goes to the CMU|Portugal team, for promoting the Ph.D. program.

Academic advising and research collaboration. Recalling the prelude, I thank Prof. José Tribolet for having encouraged me at a crucial time to apply for a Ph.D. program, highlighting academia as an option and conveying that the main product of such journey is not the final thesis but rather the resulting person.

After I first applied to the CMU|Portugal Ph.D. program, Prof. Paulo Veríssimo welcomed me at FCUL, within the Navigators group at LaSIGE. I am grateful for Paulo's leadership in promoting the participation of FCUL within the CMU|Portugal program, including my initial

research experience at CMU. I thank Paulo for having stimulated me early on to reflect on a vision for my Ph.D. path, and introducing me to the research topic of intrusion tolerance.

Prof. Priya Narasimhan welcomed me at CMU. I am grateful for the opportunity to interact with Priya's research group, namely collaborating with Soila Pertet, doing research in anomaly detection in distributed systems. I thank Priya for the advising provided in the initial part of the Ph.D. and for being supportive of my research explorations.

Prof. Alysson Bessani was my Ph.D. advisor from FCUL since 2010. I am grateful for Alysson's pragmatic research perspective, which helped me specially in my early time as a Ph.D. student, and for our research collaboration in the area of intrusion tolerance. I thank Alysson for his valuable recurring advice about paper design (including persuading me to use LaTeX for scientific writing), which was very helpful for evolving my writing style, including specific feedback that contributed to improve the dissertation.

Around mid 2011 I reasserted my vocation for research in cryptography and Prof. Virgil Gligor became my new Ph.D. advisor at CMU. I thank Virgil for his advising for my qualifying exam preparation at CMU, and for his useful emphasis on the importance of security proofs in cryptography. I am grateful for having been teaching assistant in Virgil's courses of introduction to computer security and applied cryptography, which motivated valuable conversations on security and cryptography topics.

In the summer of 2012 I visited the cryptography group at NIST's computer security division. My visit was hosted by Dr. René Peralta, who introduced me to the topic of privacy-enhancing cryptography and suggested interesting research questions. I am grateful for René's mentoring, namely the many conversations and brainstorming we had on cryptography and privacy, which expanded my perspective about research in the area. I specially thank René for his enduring academic support thereafter.

In late 2013 I searched for a complementary research context at CMU — I thank Prof. José Moura for his helpful mediation toward establishing a new advising connection at CMU.

In 2014 Prof. Nicolas Christin became my new Ph.D. advisor at CMU. I am grateful for Nicolas' openness for finding a synergy between different research areas, which paved the way for my final Ph.D. stage, and for our research collaboration in the area of brokered identification. I thank Nicolas for his advice during my Ph.D. prospectus preparation, and subsequently till completing my dissertation. I also appreciated much the weekly Cybercrime research group meetings, as a space promoting rehearsals and discussion of ongoing research.

I am grateful for the research collaboration with my co-authors, which besides Alysson Bessani and Nicolas Christin included George Danezis and an anonymous co-author.

I thank the presence and valuable feedback received from the committee members of my three main formal Ph.D. examinations: Prof. Anupam Datta at the prospectus exam at CMU; Prof. Francisco Couto and Prof. Antónia Lopes at the prova de qualificação at FCUL; Prof. Limin Jia and Prof. André Souto at the thesis defense exam; Prof. Manuel Blum and Dr. Vladimir Kolesnikov at the prospectus and thesis defense exam; Prof. Alysson Bessani, Prof. Nicolas Christin and Dr. René Peralta at the three exams. I also thank the faculty members that evaluated my doctoral-seminar presentations at FCUL and my qualifying exam at CMU.

Other academic notes. My Ph.D. path was not only technical — I also had the urge to philosophize about my student condition. The chosen challenges resulting therefrom have required spending more energy and time than desired, but that pursuit enhanced my educational experience. I am grateful to the Ombudsman at CMU for helping the university to acknowledge and validate my identity assertion as student non-employee.

I have been part of research collaborations and have also performed solitary research explorations, respectively leading to co-authored and single-authored publications. The later was not easy as a Ph.D. student — I came to realize that the ethics of authorship is a taboo in academia, and an open reflection about it has jeopardized the completion of my Ph.D. I am grateful for my prior experience as a scout, which helped me persevere and succeed in asserting my academic freedom throughout a time of tough dilemmas. I also thank Alysson for continuing as my Ph.D. advisor across this journey.

Colleagues in academia. Throughout my Ph.D. process I made friends, met exceptional people and lived memorable moments. I had the privilege to interact with colleagues in academia, sharing dilemmas and aspirations, and having fun occasions, during alternated stays in Lisbon and Pittsburgh and while traveling. While many academic connections have not directly intersected with Ph.D. research, they have influenced my path at FCUL and CMU, and so I take the chance to recall here some names, mostly of fellow students.

From my initial time at FCUL-LaSIGE, before the Ph.D. start, I recall the many lunch and coffee times with João, Giuliana, Mônica, Wagner, Marcírio, Simão and André — a fun context for informally interpreting the surrounding academia. My initial Ph.D. course-work at FCUL-DI, delving into security concepts, was done together with the CMU|Portugal MSITIS students, including Francisco, Paulo, Carlos, Sérgio, Alberto. Upon leaving to CMU for more than two years, I returned to FCUL. Creating the DI-Smalltalks series was then helpful to get to know better the research topics of my new fellow colleagues. I interacted with Bruno, Diego, Jeferson, Miguel, Pedro, Tiago C. and Vinícius, among others. I am thankful to all that contributed to the DI-Smalltalks, which were later also co-organized with Pedro and then also with Miguel.

During my first stay at CMU, CyLab, before the Ph.D. start, I met and exchanged ideas with many other students, including: at CyLab, L. Costa, Tiago C., Ricardo C. and Ricardo O. (MSITIS students), and Soila, Patrick, Aaron and Wesley; from the CMU|Portugal program, André M., Gustavo, Carla, J. Viegas, J. Brito, André D., João, Kátia, Mary, Nuno, Sofia. Upon returning to FCUL and starting the Ph.D., I came back to CMU in 2010, interacting with more colleagues across time, including: at CyLab — L. Pinto, Rolando, Shah, Zongwei, Jun, Thanassis, Divya, Saurabh; from the CMU|Portugal Ph.D. program — J. Mota, Jerónimo, Ivonne, S. Pequito, David, Mate, Leid, L. Marujo, J. Semedo., Hugo P.; from other Master programs — Alexandre, Hugo M., Nuno J., N. Brito, Nina. Later, after two years at FCUL, I was again at CMU since late Summer 2014, with new colleagues, including: at CyLab — Janos, Kyle, Mahmood, Zack, Aya; others at CMU — Gonçalo and Vasco and the other InRes 2014 participants, Damião, Tiago P., Evgeny, Adwait, André.

Also as a Ph.D. student, I visited NIST in Maryland for 2+ months in the Summer of 2012 and for some short periods sometimes after. I had a great time there, mostly due to the great company of Meltem and René, and the welcoming environment from those in the Crypto group. In 2015 I also did an internship at Bell Labs, made possible by Vlad, enjoying two months of brainstorming in cryptography with Vlad and Luke. These periods contributed to widen my research experience, namely in topics complementary to the dissertation topic.

I am also grateful for the kind and friendly assistance, helpful across time, from people from: the ICTI / CMU-Portugal program offices, namely Sara and Alexandra; the CMU-ECE Graduate Program Office, namely Elaine, Samantha and Nathan; and LaSIGE, namely Pedro. I also thank all administrative assistance obtained throughout time at FCUL and CMU, namely at CyLab where friendliness was always abundant.

Final notes. Not all that counts is counted here, such as connections outside academia. I made a multitude of other international connections that heightened my spirit throughout these years. I am grateful for the housemates, dance buddies, conference and travel companions, couch-surfing hosts and visitors, and other friends. I cherish the memory of these personal connections — some brief, others long-lasting — during which the Ph.D. gained shape. The human interaction was an essential contrast to the periods of solitary research endeavor. I thank everyone, including those that, even though not mentioned here, have somehow been part of my Ph.D. quest ... some in very important ways.

Most significantly, the continuous support of my family kept me going throughout this long Ph.D. I dedicate this dissertation to my parents, for whom I am foremost grateful. They have encouraged and supported me all the way, in many ways. Their ever-present support, specially as I consciously pursued risky academic paths, was an inspiration to remain enthusiastic!

Abstract

This doctoral dissertation presents contributions advancing the state-of-the-art of secure two-party computation (S2PC) — a cryptographic primitive that allows two mutually distrustful parties, with respective private inputs, to evaluate a function of their combined input, while ensuring privacy of inputs and outputs and integrity of the computation, externally indistinguishable from an interaction mediated by a trusted party. The dissertation shows that *S2PC can be made more practical by means of innovative cryptographic techniques, namely by engineered use of commitment schemes with special properties, enabling more efficient protocols, with provable security and applicable to make systems more dependable.* This is one further step toward establishing S2PC as a practical tool for privacy-preserving applications.

The main technical contribution is a new protocol for S2PC of Boolean circuits, based on an innovative technique called *forge-and-lose*.[1] Building on top of a traditional cut-and-choose of garbled circuits (cryptographic versions of Boolean circuits), the protocol improves efficiency by reducing by a factor of approximately 3 the needed number of garbled circuits. This significantly reduces a major communication component of S2PC with malicious parties, for circuits of practical size. The protocol achieves simulatable S2PC-with-commitments, producing random commitments of the circuit input and output bits of both parties. The commitments also enable direct linkage of several S2PCs in a malicious adversarial setting.

As second result, the dissertation describes an improvement to the efficiency of one of the needed sub-protocols: simulatable two-party coin-flipping.[1] The sub-protocol is based on a new universally composable commitment scheme that for bit-strings of increasing size can achieve an asymptotic communication-complexity rate arbitrarily close to 1.

The dissertation then discusses how S2PC-with-commitments can enable in brokered identification systems a difficult-to-achieve privacy property — a kind of *unlinkability*.[1] This mitigates a vector of potential mass surveillance by an online central entity (a hub), which is otherwise empowered in systems being developed at nation scale for authentication of citizens. When the hub mediates between identity providers and service providers the authentication of users, an adequate S2PC (e.g., of a block-cipher) can prevent the hub from learning user pseudonyms that would allow linking transactions of the same user across different services providers.

Keywords: cryptography, secure two-party computation (S2PC), forge-and-lose technique, extractable and equivocable commitment schemes, brokered identification and authentication.

[1] Parts of these contributions were previously presented at ASIACRYPT 2013, PETS 2015 and PKC 2016.

Page intentionally blank

Resumo (Summary in Portuguese)

Esta dissertação apresenta contribuições[2] para o avanço do estado da arte da "computação segura entre dois agentes" (*secure two-party computation*, S2PC) — uma primitiva criptográfica que permite a dois agentes mutuamente desconfiados, cada um com o seu próprio input privado, avaliar corretamente uma função da combinação dos seus inputs, garantindo privacidade do input e output de cada agente. Um protocolo de S2PC num *mundo real* não depende de uma entidade terceira, mas *emula* um protocolo que teria lugar num mundo ideal em que uma entidade terceira atuaria como mediadora confiável, recebendo os inputs privados dos dois agentes, computando localmente a função desejada e retornando o(s) output(s) ao(s) agentes. Mais genericamente, os protocolos de S2PC são *simuláveis*, i.e., a sua segurança é definida e provada matematicamente no paradigma ideal/real de simulação, garantindo que o que é permissível no mundo real também seria possível no mundo ideal, mesmo que os agentes se comportem arbitrariamente e com malícia.

Embora os protocolos de S2PC sejam conhecidos e tenham melhorado ao longo das últimas três décadas, o seu elevado custo computacional e de comunicação ainda dificulta que a S2PC seja considerada como um bloco construtivo em sistemas reais. Esta dissertação mostra que *a "computação segura entre dois agentes" pode tornar-se mais prática por meio de técnicas criptográficas inovadoras, nomeadamente através da utilização engenhosa de "esquemas de comprometimento"* (commitment schemes) *com propriedades especiais, permitindo protocolos mais eficientes, com segurança demonstrável, e aplicáveis a tornar os sistemas mais fidedignos.* As contribuições apresentadas são mais um passo no sentido de tornar a S2PC uma ferramenta prática bem estabelecida para aplicações preservadoras de privacidade.

S2PC-com-compromissos e a técnica forjar-e-perder. A principal contribuição técnica descrita nesta dissertação é um novo protocolo para S2PC de circuitos Booleanos, baseado numa utilização inovadora de "compromissos de bits" (*bit commitments*) e numa nova técnica denominada "forjar-e-perder" (*forge-and-lose*). O protocolo induz uma melhoria sobre a técnica tradicional de "cortar-e-escolher" (*cut-and-choose*) associada a "circuitos

[2]Parcialmente apresentadas nas conferências ASIACRYPT 2013, PETS 2015 e PKC 2016.

baralhados" (*garbled circuits*), para contextos em que os agentes possam estar corrompidos maliciosamente. Concretamente, o novo protocolo reduz por um fator de aproximadamente 3 o número necessário de circuitos baralhados, que constituem uma componente principal da complexidade de comunicação associada a S2PC de circuitos de tamanho prático.

Na abordagem cortar-e-escolher, o conjunto dos circuitos baralhados gerados pelo agente construtor é dividido pelo agente avaliador em dois sub-conjuntos: um para avaliação e outro para verificação. Nas técnicas tradicionais, o critério de sucesso requer, no caso dos circuitos verificados serem todos corretos, que do conjunto de circuitos avaliados uma maioria seja correta. Essas técnicas requerem assim pelo menos 123 circuitos, dos quais 74 para verificação e 49 para avaliação, para garantir uma segurança estatística de 40 bits (uma probabilidade de erro de cerca que 1 em cada milhão de milhão de tentativas — um padrão de referência comum nesta área). Em contraste, na técnica forjar-e-perder basta que um dos circuitos avaliados seja correto. O diferente critério de sucesso induz uma melhoria estatística, sendo então suficiente usar 40 circuitos para obter os 40 bits de segurança estatística.

Na prática, a comunicação de circuitos de verificação pode ser trocada por elementos muito pequenos, tornando a complexidade de comunicação dependente sobretudo do número de circuitos de avaliação, que pode reduzir-se em troca de um aumento (maior) do número de circuitos de verificação. Por exemplo, para um total de 123 circuitos, na técnica forjar-e-perder o número de circuitos avaliados pode ser limitado a um máximo de 8, para o mesmo objetivo de 40 bits de segurança estatística, i.e., uma redução de aproximadamente 6 vezes em comparação aos 49 necessários em técnicas tradicionais.

O novo protocolo também realiza S2PC-com-compromissos, produzindo, para ambos os agentes, compromissos aleatórios dos bits de input e de output dos circuitos. Num esquema de comprometimento, uma fase inicial de *comprometimento* permite a *vinculação* de um agente transmissor, perante um agente recetor, a um valor (e.g., uma sequência de bits) escolhido pelo transmissor, ao mesmo tempo garantindo a *ocultação* desse valor perante o recetor. Numa fase de *abertura* o transmissor consegue revelar o valor escolhido de forma convincente, i.e., sem que possa mudar o valor comprometido. Os compromissos obtidos no protocolo de S2PC-com-compromissos permitem diretamente que várias S2PC sejam encadeadas de forma segura num contexto malicioso.

Esquemas de comprometimento simuláveis. Para que o protocolo de S2PC-com-compromissos seja *simulável*, i.e., seguro no modelo ideal/real de simulação, também tem de ser simulável um subprotocolo de "moedas ao ar entre dois agentes" (*two-party coin-flipping*) utilizado para randomizar os compromissos. Neste contexto, a dissertação descreve, como segunda contribuição principal, uma nova técnica que melhora a eficiência de coin-flipping *simulável*. A melhoria é obtida através de um novo esquema de comprometimento "componível universalmente" (*universally composable*) que permite, separadamente para o comprometimento e a abertura de sequências-de-bits de comprimento progressivamente maior (comprimento alvo), uma taxa de complexidade de comunicação assintoticamente próxima de um fator que pode ser escolhido arbitrariamente próximo de 1. Ou seja, o número de bits comunicados em cada fase não é maior do que o produto do comprimento alvo por esse fator.

Para que o esquema de comprometimento seja simulável, tem de ser extraível (*extractable*) e equivocável (*equivocable*). Em aparente oposição à *ocultação*, ser extraível significa que, na fase de comprometimento, um simulador no papel de recetor é capaz de extrair o valor comprometido pelo transmissor. Em aparente oposição à *vinculação*, ser equivocável significa que, na fase de abertura, um simulador no papel de transmissor é capaz de revelar convincentemente qualquer valor à escolha. As propriedades de simulação são compatíveis com as propriedades definitórias (ocultação e vinculação) porque, comparativamente a transmissor ou recetor reais, o simulador tem poder adicional, e.g., derivado de informação secreta obtida numa configuração inicial que apenas o simulador pode adulterar. É fácil obter isoladamente cada uma das propriedades, mas difícil obtê-las conjuntamente com eficiência.

Nesta dissertação, a nova abordadem — denominada "expandir-mascarar-sintetizar" (*expand-mask-hash*) — utiliza um "gerador de pseudo aleatoriedade" (*pseudo-randomness generator*) e uma função de "síntese resistente a colisões" (*collision-resistant hash*) para combinar de forma eficiente compromissos extraíveis e compromissos equivocáveis, associados a sequências curtas de bits, obtendo uma propriedade unificada do tipo *extraível*-e-*equivocável* amplificada até um comprimento alvo arbitrariamente maior, amortizando o custo dos compromissos base. A abordagem pode ser instanciada com um protocolo não interativo em cada fase do esquema de comprometimento, i.e., tanto o comprometimento como a abertura podem ser executados pelo envio de uma única mensagem.

Identificação mediada. Como aplicação prática, a dissertação discute como a S2PC-com-compromissos pode permitir uma propriedade de privacidade difícil de obter em sistemas de "identificação mediada" (*brokered identification*). Em jogo está a autenticação de "utilizadores" (cidadãos de um país) em muitos "fornecedores de serviço" (*service providers*) *online*, com base na identidade de utilizador atestada por "fornecedores de identidade" (*identity providers*, IDP) *online*. Alguns sistemas em desenvolvimento à escala nacional em diversos países requerem que uma entidade central *online* (um "*hub*") medeie a autenticação de utilizadores, prevenindo que os IDP saibam quais fornecedores de serviço estão a ser acedidos por cada utilizador em cada transação de identificação/autenticação. Contudo, numa implementação direta desses protocolos, o *hub* (controlado pelo governo do respetivo país) ganha uma ampla capacidade de rastreio da atividade dos utilizadores, i.e., um potencial vetor de vigilância em massa, por ser capaz de ver os pseudónimos-de-utilizador definidos pelo IDP. Esta dissertação descreve como uma S2PC adequada (e.g., de uma cifra de bloco) pode ser usada para esconder estes pseudónimos, prevenindo assim o *hub* de correlacionar transações do mesmo utilizador em vários fornecedores de serviço.

A propriedade pretendida — denominada "não-correlação fraca (*weak unlinkability*), pelo hub, de pseudónimos-de-utilizador entre fornecedores de serviço" — deve ser conseguida sem impedir o hub de transformar o pseudónimo-de-utilizador em trânsito entre o IDP e o fornecedor de serviço. A transformação tem de ser reproduzível, para que cada utilizador consiga retornar à sua conta em cada fornecedor de serviço, e por questões de privacidade o seu resultado não pode ser calculável pelo IDP nem correlacionável entre diferentes fornecedores de serviço. Para conciliar os requisitos, a transformação de pseudónimos pode ser efetuada através de uma S2PC, em que o pseudónimo-de-utilizador definido pelo IDP é usado como input de uma cifra de bloco, e a chave da cifra é um identificador privado escolhido pelo hub para representar o fornecedor de serviço. Assim, o hub não aprende o pseudónimo-de-utilizador definido pelo IDP, e o IDP não aprende o pseudónimo-de-utilizador transformado. A utilização de S2PC-com-compromissos permite ainda que o hub e o IDP possam enviar um ao outro uma assinatura eletrónica (criptográfica, não repudiável), que permite a cada um comprovar mais tarde, numa eventual ação de auditoria, quais valores foram utilizados ou obtidos em interação real com a outra parte, sem no entanto terem de conhecer todos os pseudónimos envolvidos.

Palavras-chave: criptografia, computação segura entre dois agentes (S2PC), técnica forjar-e-perder, esquemas de comprometimento extraíveis e equivocáveis, identificação mediada.

Table of contents

List of Illustrations

List of Tables

List of Figures

Notation

Technical abbreviations and acronyms

- 1/2 OT: 1-out-of-2 oblivious transfer
- 2/1 OT: 2-out-of-1 oblivious transfer
- AES: advanced encryption standard
- BI: Blum integer
- BitCom: bit commitment
- C&C: cut-and-choose
- Com: commitment
- commun.: communication
- CR: collision resistant
- CRS: common reference string
- DDH: decisional Diffie-Hellman
- DL: discrete logarithm (abbr. discrete log)
- DLC: discrete-logarithm cryptography
- DQR: decisional quadratic-residuosity
- ECC: elliptic-curve cryptography
- Equiv: equivocable
- ElG: ElGamal (BitCom or BitCom scheme)
- eval: evaluation (type of challenge)
- Ext: extractable
- FFC: finite-field cryptography
- F&L: forge-and-lose
- GBI: good Blum integer
- GC: garbled circuit
- GCRS: global CRS
- GEB: good ElGamal BitCom
- GM: Goldwasser-Micali (BitCom scheme)
- GPKI: global PKI
- hom.: homomorphic or homomorphism
- HV: honest-verifier (used in HVZK)
- IDA: information dispersal algorithm
- IDP: identity provider
- IFC: integer-factorization cryptography
- kB: kilo Bytes (thousands of bit octets)
- LHT: linear homomorphic transformation
- max, min: maximum, minimum
- mod: modulo (remainder upon integer division)
- msg: message (committed value)
- NI: non-interactive (used in NIZK and NIZKPoK)
- NM: non-malleable or non-malleability
- NPRO: non-programmable random oracle
- OT: oblivious transfer
- Ped: Pedersen (BitCom scheme)
- PKI: public-key infrastructure
- PRG: pseudo-randomness generator
- RandLHT: Randomized LHT
- Rep: (Pedersen) representation

- RP: relying party (as a service provider)
- RSC: random seed checking
- S2PC: secure two-party computation
- SHA: secure hash algorithm
- TTP: trusted third party
- Type-C: type commit
- Type-RC: type reveal-for-check
- Type-RE: type reveal-for-evaluation
- Ver: Verify (predicate verification)
- XOR: (bit-wise) Boolean eXclusive OR
- ZK: zero knowledge
- ZKP: ZK *proof* (or ZK *argument* (ZKA))
- ZKPoK: ZKP of knowledge (or ZKA of knowledge)

Technical symbols

One or two interpuncts ($\cdot$ and $\cdot\cdot$) are used to denote arbitrary symbols; an ellipsis ($\cdots$) is used to denote an implicitly defined sequence of elements.

Sets

- $\cdot \in \cdot\cdot$ (element $\cdot$ belongs to set $\cdot\cdot$, or ranges over all elements of set $\cdot\cdot$)
- $[s]$ (set $\{1, ..., s\}$)
- $\varnothing$ (empty set)
- $\#(\cdot)$, $\# \cdot$ (size of set $\cdot$, number of $\cdot$)
- $\cdot \cup \cdot\cdot$ (union of sets $\cdot$ and $\cdot\cdot$)
- $\cdot \cap \cdot\cdot$ (intersection of sets $\cdot$ and $\cdot\cdot$)
- $\cdot \setminus \cdot\cdot$ (set $\cdot$ except the elements contained in $\cdot\cdot$)
- $\mathbb{F}$ (family of indexed functions)
- $\mathbb{N}$ (set of positive integers)
- $\mathbb{S}$ (family of indexed sets)

Arithmetic, logic, relations

- **Binary**
  - $+$, $-$, $\times$, $/$ (sum, subtract, multiply, divide)
  - $=$, $\approx$ (equal, approximately equal)
  - $\stackrel{?}{=}$ (equality verification — `true` or `false`)
  - $\equiv$ (equivalent, or equal by definition)
  - $<$, $>$ (less than, greater than)
  - $\leq$, $\geq$ (less or equal, greater or equal)
  - $\gtrsim$ (greater or approximately equal)
  - $\oplus$ (bitwise XOR operation)
  - $\vee$, $\wedge$ (logical OR, logical AND)
- $\sum_{\cdot \in \cdot\cdot}$, $\prod_{\cdot \in \cdot\cdot}$ (sum and product when variable $\cdot$ ranges over set $\cdot\cdot$)
- $\lfloor \cdot \rfloor$ (floor, i.e., highest integer less than $\cdot$)
- $\lceil \cdot \rceil$ (ceiling, i.e., lowest integer greater than $\cdot$)
- $|\cdot|$ (length of $\cdot$, i.e., number of bits of $\cdot$)
- $\neg \cdot$ (logical negation of the Boolean value $\cdot$)
- $2^{\cdot}$ (two (integer) to the power of $\cdot$ (integer))
- % (percent, i.e., 1/100)
- ∞ (infinite or infinity)

Entities in protocols

- $\mathcal{A}$ (the real-world *adversary*, playing in the real world or in a simulation created by $\mathcal{S}$)
- $\mathcal{F}$ (an ideal functionality — a TTP in the ideal world, or in a hybrid model)
- P_1, P_2 (1^{st}, 2^{nd} party to learn the coin-flip)
- P_A (GC constructor, a.k.a. Alice)
- P_B (GC evaluator, a.k.a. Bob)
- P_R, P_S (receiver, sender of a commitment)
- P, V (prover, verifier in a ZKP or ZKPoK)

- p, $\bar{p}$ (p is the index of a party, e.g., $p \in \{\mathrm{A}, \mathrm{B}\}$, or $p \in \{\mathrm{R}, \mathrm{S}\}$, or $p \in \{1, 2\}$; $\bar{p}$ is the index of the complementary party, e.g., B if p is A)
- $\mathcal{S}$ (*simulator* — the ideal-world adversary)
- $\mathcal{Z}$ (the *environment* — entity that tries to distinguish whether executions happen in the real world or the ideal world)
- $\cdot^*$ (corrupted/malicious version of party $\cdot$)
- $\widehat{\cdot}$ (counterpart, in ideal world, of real party $\cdot$)

Protocol notation

- $\downarrow\cdot$ (output $\cdot$, i.e., return $\cdot$ to the upper level that requested execution of the current function)
- $\leftarrow^{\$}\cdot$ (uniform random sampling from domain $\cdot$ or using probabilistic procedure $\cdot$)
- $\perp$ (undefined, e.g.: $\cdot = \perp$ declares symbol $\cdot$ without assigning it a value; $\cdot \neq \perp$ tests whether $\cdot$ has been assigned a concrete value; $\downarrow \perp$ terminates a function or thread without output)
- $\mathrm{P}_p : \cdot$ (P_p knows value $\cdot$ or makes computation $\cdot$)
- $\mathrm{P}_p \rightarrow \mathrm{P}_{\bar{p}} : \cdot$ (P_p sends message $\cdot$ to $\mathrm{P}_{\bar{p}}$)
- $\mathrm{P}_p \leftrightarrow \mathrm{P}_{\bar{p}} : \cdot$ (P_p and $\mathrm{P}_{\bar{p}}$ interact to obtain $\cdot$)
- 1^{κ} (cryptographic security parameter, also as κ (number of bits))
- 1^{σ} (statistical security parameter, also as σ (number of bits) — not the group element σ)
- $\cdot \in \mathit{poly}(\cdot\cdot)$ ($\cdot$ is asymptotically lower than a particular polynomial function of $\cdot\cdot$)

C&C challenges

- s (total number of C&C challenges)
- e, v (number of *evaluation*, *check* challenges)
- b (number of indices with bad instances)
- j (index of challenge, e.g., of each instance with a GC and connectors in a S2PC, or of each instance with a seed and hash commitments in Ext-and-Equiv-Com protocol)
- J_E (subset of indices selected for *evaluation*)
- J_V (subset of indices selected for *check*)
- J_{Ignore} (subset of indices ignored once detected as incorrect in the *evaluation* stage)

RSC technique

- λ_j (PRG seed to generate an instance (with index j) in the cut-and-choose structure)
- Λ (global hash)
- $\overline{\Lambda}$ (RSC Equiv-Com of Λ)
- $\underline{\Lambda}$ (randomness used to commit Λ)

Boolean circuits and garbled Circuits

- b (bit value)
- c (bit index; e.g., underlying bit of key $k^{[c]}$, bit encoded by group element $\mu^{(c)}$, position of commitment $\bar{k}^{\langle c \rangle}$ in a pair, bit associated with a multiplier $\beta^{(0)}_{j,i,c}$ class 0 of a wire of P_B)
- $c_{j,i}$ (permuted input bit of P_A, if $j \in I_\mathrm{A}$; tentative output bit in wire i of GC j, possibly permuted (if $j \in \mathrm{O}_\mathrm{A}$), correct if P_A was honest)
- C or F_C (Boolean circuit)
- C' (adjusted Boolean circuit — has extra input wires of P_A ($i \in I_{\mathrm{A}'}$, for bit-masks for the private output of P_A), and adjusts the set (O_AB) of common output wires into two disjoint sets ($\mathrm{O}_{\mathrm{A}'}$, $\mathrm{O}_{\mathrm{B}'}$), one per party)
- $\epsilon(\cdot)$ (function that reveals the bit underlying a circuit output wire key)
- $\mathrm{GC}_{\mathrm{Eval}}$ (algorithm to evaluate a GC)
- i (index of wire)
- I_A, I_B (set of indices of original circuit input

wires of $\mathrm{P_A}$ and $\mathrm{P_B}$, respectively)

- $I_{A'}$ (set of indices of extra input wires of $\mathrm{P_A}$, for bit-masks for private output bits of $\mathrm{P_A}$)
- InKeys, $\overline{\text{InKeys}}$ (sequence of garbled input keys, and respective commitments, possibly indexed (in subscript) per party and per garbled circuit, and (in superscript) containing one or two keys per wire)
- k (wire key of input or output of a GC, indexed per garbled circuit (j), per wire (i) and per bit value (c))
- $k^{\langle c \rangle}$ (key in position c (0 or 1) in a pair of keys)
- $k^{[c]}$ (key with underlying bit c)
- ℓ_A, ℓ_B, ℓ'_A, ℓ'_B (number of bits of input of $\mathrm{P_A}$, input of $\mathrm{P_B}$, output of $\mathrm{P_A}$, output of $\mathrm{P_B}$)
- $\mathrm{O_{AB}}$ (set of original indices of output wires common to $\mathrm{P_A}$ and $\mathrm{P_B}$)
- O_p (set of original indices of output wires of P_p, e.g., $\mathrm{O_A}$, $\mathrm{O_B}$ — with private (O_{pp}) and common wires (O_p), possibly intersecting $\mathrm{O}_{\bar{p}}$)
- O_{pp} (set of original indices of only the private output wires of P_p)
- $\mathrm{O}_{p'}$ (set of adjusted indices of output wires of P_p, e.g., $\mathrm{O_{A'}}$ or $\mathrm{O_{B'}}$, disjoint from all wire indices of $\mathrm{P}_{\bar{p}}$)
- OutKeys (same as InKeys, but referring only to output keys)
- π (permutation bit or bit-string, used to permute input bits of $\mathrm{P_A}$)
- ξ (wire key (in permuted position) of input bit of $\mathrm{P_A}$; or wire key of input bit of $\mathrm{P_B}$; or tentative wire key of output)
- x_p (bit-string input of P_p, e.g., x_A, x_B)
- y_p (bit-string output of P_p, e.g., y_A, y_B)

Groups

- $\mathbb{G}$ (group set)
- $*$, $*'$ (group operations, in multiplicative notation — when useful to distinguish, $*'$ may be used in the space of commitments, and $*$ in the space of "randomness" of commitments)
- $\overset{+}{*}$ (group addition or multiplication, in the space of "randomness" of a commitment, respectively with additive or multiplicative notation, e.g., respectively for ElGamal or GM BitComs)
- $\overset{+}{*}_{(\pi)}$, $*'_{(\pi)}$ (pseudo XOR-homomorphic operations explicitly dependent on bit or bit-string π, respectively adjusted from $\overset{+}{*}$ and $*'$)
- $(\cdot)^2$ (modular square of group element $\cdot$)
- g (generator of some multiplicative group)
- $\langle g \rangle$ (group generated by (powers of) g)
- h (group homomorphism onto XOR)
- $h^{-1}(b)$ (subgroup of elements whose homomorphic image is b)
- inv (inverse of group operation in the space of randomness of a scheme, e.g., multiplicative inverse for Blum and GM BitComs, additive inverse for ElGamal and Pedersen BitComs)
- $J_N(c)$ (subset of integers, modulo N, with Jacobi Symbol c)
- $\mathrm{JS}_N(\cdot)$ (Jacobi Symbol modulo N of integer $\cdot$; value in $\{-1, 1\}$ if $\cdot$ is in $\mathbb{Z}^*_N$, 0 otherwise)
- N (a concrete Blum integer)
- $\mathrm{NTSqrt1}_N$ (non-trivial square-root of 1, modulo a Blum integer N, i.e., $\mathrm{Sqrt}^{(1)}_N(1)$)
- QR_N (set of quadratic residues modulo N)
- $\mathrm{Sqrt}^{(b)}_N(\cdot)$ (square root class b of $\cdot$, modulo N)
- $\mathrm{PseudoSqrt}[N,t](w;c)$ (square-root class c, modulo N, of either w or its additive inverse,

computed with the help of trapdoor t)
- t (trapdoor of Com scheme identified in subscript, e.g., t_{A}, t_{B}, t_{CRS}, t_{FL}, t_N, t_{OT}, t_p)
- $\mathbb{Z}_q$ (set of residues of the additive group of integers modulo q, i.e., all non-negative integers smaller than q)
- $\mathbb{Z}_N^*$ (set of residues of the multiplicative group of integers modulo N, i.e., all positive integers co-prime with N and smaller than N)

Commitments

- $\mathscr{C}$, $\mathcal{O}$ (*commit* and *open* operations, possibly interactive and possibly probabilistic; $\mathscr{C}$ is also used generically to denote a Com scheme)
- $\mathscr{B}_{\mathrm{A}}$, $\mathscr{B}_{\mathrm{B}}$ (BitCom schemes for outer BitComs of bits of $\mathrm{P_A}$ and $\mathrm{P_B}$)
- $\mathscr{B}_{\mathrm{ConA}}$ (BitCom scheme for intermediate BitComs of input bits of $\mathrm{P_A}$)
- $\mathscr{B}_{\mathrm{FLA}}$, $\mathscr{B}_{\mathrm{FLB}}$ (dual BitComs schemes used to support the forge-and-lose technique, resp. for input bits of $\mathrm{P_A}$ and output bits of $\mathrm{P_B}$)
- $\mathscr{B}_{\mathrm{OT}}$ (BitCom scheme for intermediate BitComs of input bits of $\mathrm{P_B}$, supportive of OTs)
- $\mathscr{C}_{\mathrm{RSC}}^{\mathrm{Equiv}}$ (Equiv-commitment scheme used to commit a RSC hash)
- $\mathscr{C}_{\mathrm{InKey}}$ (commitment scheme used to commit wire-keys of input of $\mathrm{P_A}$)
- $\mathscr{E}$ (hom. encryption related to 1-out-of-2 OT)
- $\overline{k}$ (commitment of a wire key)
- $\underline{k}$ (randomness used to open a wire key Com)
- α', β', ϕ', γ', μ', ν', ρ', σ', ς' (hom. commitments produced using the respective "randomnesses" α, β, ϕ, γ, μ, ν, ρ, σ, ς)
- **In protocol for Ext&Equiv Com:** *Auth* (authenticator function, indexed by nonce z), a_j (authenticator (output) of m'_j)), α_j (tentative authenticator output), ℓ_a (length of authenticator output), ℓ_z (length of nonce), m (committed message), μ or μ_j (tentative message), m'_j (message fragment), μ'_j (tentative fragment), s_j (seed), s'_j (mask), ς'_j (tentative mask), h_j (hash of mask), η_j (tentative hash of mask), t_j (masking — XOR of message (or fragment) and mask), t (threshold of erasure code), z (nonce).

Randomness sampling

- Gen. (random sampling of elements of type $\cdot$, e.g., $\mathrm{Gen}_{\$\mathrm{ForCom}}$, $\mathrm{Gen}_{\mathrm{Seed}}$, $\mathrm{Gen}_{\mathrm{PairOpenings}}$)
- PRGen. (pseudo-random sampling of element of type $\cdot$, e.g., $\mathrm{PRGen}_{\mathrm{AuxiSeed}}$, $\mathrm{PRGen}_{\mathrm{BitString}}$, $\mathrm{PRGen}_{\mathrm{Exp}}$, $\mathrm{PRGen}_{\$\mathrm{ForCom}}$, $\mathrm{PRGen}_{\$\mathrm{ForLHT}}$, $\mathrm{PRGen}_{\mathrm{GC}}$, $\mathrm{PRGen}_{\mathrm{InKey}}$)

Randomness of hom. commitments

- α (*multiplier*, encoding of permutation bit π, used for connectors of input wires of $\mathrm{P_A}$)
- β (*multiplier*, encoding of 0, used for connectors of input and output wires of $\mathrm{P_B}$)
- ϕ (randomness of a BitCom of input of $\mathrm{P_A}$, produced by the intermediate BitCom scheme $\mathscr{B}_{\mathrm{FLA}}$; may be the same as σ when $\mathscr{B}_{\mathrm{FLA}}{=}\mathscr{B}_{\mathrm{A}}$, and/or the same as μ when $\mathscr{B}_{\mathrm{FLA}}{=}\mathscr{B}_{\mathrm{ConA}}$)
- γ (random encoding of 0, obtained via coin-flipping, used to permute an initial encoding σ into a final random encoding ρ)
- μ (intermediate encodings used to interface between outer encodings and connectors; related to $\mathscr{B}_{\mathrm{ConA}}$, $\mathscr{B}_{\mathrm{OT}}$ and $\mathscr{B}_{\mathrm{FLB}}$)
- ν (inner encoding used in connectors; related

to $\mathscr{B}_{\text{ConA}}$, $\mathscr{B}_{\text{OT}}$ and $\mathscr{B}_{\text{FLB}}$)
- u, v (tentative values for μ and ν, respectively, correct if P_A was honest)
- ρ (final encoding of input or output bit, using $\mathscr{B}_\text{A}$ or $\mathscr{B}_\text{B}$, after final random permutation)
- σ (initial encoding of input bits (before the S2PC), or adjusted encoding of output bits (after the S2PC), before the final random permutation; related to $\mathscr{B}_\text{A}$ and $\mathscr{B}_\text{B}$)
- ς (initial encoding of random preparatory bits, helpful to encode the final output bits; related to $\mathscr{B}_\text{A}$ and $\mathscr{B}_\text{B}$)

Other notation

Institutional or conventional acronyms

- CMU-ECE: Carnegie Mellon University, Electrical & Computer Engineering (Department)
- DOI: Digital Object Identifier
- FCCX: Federal Cloud Credential Exchange
- FCUL-DI: Faculdade de Ciências da Universidade de Lisboa, Departamento de Informática
- IACR: International Association for Cryptology Research
- IAP: Identity Assurance Principles (in the UK)
- ICTI: Information and Communication Technologies Institute
- NIST: National Institute of Standards and Technology
- NSTIC: National Strategy for Trusted Identities in Cyberspace (in the US)
- UK: United Kingdom
- US: United States (of America)

Language abbreviations

- abbr.: abbreviated or abbreviation
- a.k.a.: also known as
- Dr., Prof.: Doctor, Professor (academic titles)
- e.g.: *exempli gratia* (Latin), a.k.a. for example
- eds.: editors (of a bibliographic item)
- et al.: *et alia* (Latin), a.k.a. and others
- Fig.: Figure
- i.e.: *id est* (Latin), a.k.a. that is
- Illust.: Illustration
- Ph.D.: Doctor of Philosophy (academic degree), or Doctoral (adjective, e.g., in Ph.D. student), or path within a Doctoral program
- Proc.: Proceedings of the, or Proceedings on
- resp.: respective or respectively
- vol.: volume (of a bibliographic item)
- vs.: versus
- wrt: with respect to

Other symbols

- § (prefix to the number of a referenced "subsection" (division block below "Section" level, in a Chapter or Appendix) or its sub-divisions)

Chapter 1

Introduction

1.1 Secure two-party computation

Secure two-party computation is a general cryptographic functionality that allows two parties to interact as if intermediated by a trusted third party [Gol04]. A canonical example is *the millionaire's problem* [Yao82], where two parties find whether or not the first party is richer than the second, without any party revealing to the other any additional information about their amounts. One can envision applications of secure computation in cases where mutually distrustful parties can benefit from learning something from their combined data, without sharing their inputs [Kol09]. For example, two parties may evaluate in a privacy-preserving manner a data mining algorithm over their combined databases [LP02]. As a different example, one party with a private message may obtain an enciphering of the message, calculated with a secret key from another party (i.e., blind enciphering) [PSSW09]. One general solution to this type of problems is secure two-party evaluation of Boolean circuits, hereinafter denoted "S2PC" [Yao86]. The function to be computed is first encoded as a Boolean circuit, and the respective private inputs are encoded as bit-strings. Then, as a result of the S2PC, each party learns only the output of the respective circuit evaluated over both private inputs. Naturally, the term "private output" refers to privacy beyond what each party can infer from the learned output and the known input.

S2PC enables interactions with high security assurance, enhancing privacy and providing a fine-grained control of the information that parties leak when interacting toward an established

goal. For example: *Why should a millionaire disclose her wealth, if it is enough to just verify the predicate of whether or not it is larger than the wealth of someone else? Why should two hospitals share private data of their patients, if all they need is to learn a certain statistic of the combined data? Why should a central authority learn a cryptographic key and a plaintext block from two respective parties, if the underlying application just requires that the second party learns the respective enciphering of the plaintext?* Even if a *trusted third party* (TTP) may be hypothetically available to mediate these interactions, in practice it may be undesirable. For example: a budget restriction may disallow payments for services by a TTP; corruption may be a concern, e.g., one malicious party may try to coerce or bribe the TTP, or even stealthily affect an honest TTP, undetected by the other honest party; some interactions may require secrecy by design (e.g., by law), thus precluding any solutions that involve sharing of private inputs with any external party.

S2PC bypasses the problems inherent to a TTP by directly avoiding the TTP. This may also reduce the "attack surface" of the overall system, by reducing the number of parties that may be subject to external attacks. This is extremely important in the face of persistent threats where an external stealthy attacker may attempt to undetectably intrude one of the parties [BB12], to break some security goal, such as confidentiality, integrity or availability. In other words, even in the paradigm of intrusion-tolerant systems [VNC03], S2PC naturally fits as a technique for improving resilience of a system.

While S2PC may avoid trusted parties in certain interactions, the ability to *trust* a party (i.e., to assume the absence of certain types of adversarial behavior) may nonetheless be desirable, e.g., if it enables more efficient and simple interactions toward a desired goal. Naturally, the meaning of *desirable trust* is intended within a context of matching trustworthiness of the system upon which trust is placed. Rather than imposing that trust be a binary property (trusted vs. distrusted), it may be useful to consider a range of shades of trust. In this regard, S2PC may enable secure interactions based on (a controlled level of) trust, where otherwise it would not be advisable to "fully" trust the participants. In other words, S2PC can also serve as a trust-enabling tool, by reducing trustworthiness requirements to a level where it becomes appropriate to trust the parties involved in an interaction. For example, it may be appropriate to assume that certain subsets of parties will not maliciously collude, while also realizing that there is an incentive for certain individual parties to misbehave. If, in such a system, S2PC can be used to create resilience against individual malicious behavior, then it becomes appropriate

to base the security of the system on the trust that undesired collusions will not occur. As a concrete example, this dissertation considers brokered identification systems, where S2PC can be used to allow the presence of a central broker (a mediator) of authentications between other parties, but without letting security depend on full trust about said broker.

Despite the envisioned usefulness of S2PC, and theoretical protocols being known for about three decades [Yao86], widespread secure computation is still an elusive reality. There are indeed associated difficulties at different levels: (i) a direct implementation of general solutions often leads to prohibitive time and computational cost, due to large circuits; (ii) being a somewhat counter-intuitive concept (a possible rhetorical question being: *how can one compute a function over distributed inputs without sharing data?*), even security practitioners may remain oblivious to the benefits of secure computation, when reasoning about security goals; and (iii) it may be conceptually challenging to cope with technical subtleties associated with the goal of achieving provable security.

In the framework of secure computation, security is considered in the ideal/real simulation paradigm [Can00, Can01]. The intended functionality (e.g., S2PC) is defined in an ideal world, where a TTP mediates the communication and computation between the other two parties. By definition, anything possible in the ideal world is considered permissible (i.e., not a security violation), even when one of the parties (i.e., except the TTP) is corrupted by an adversary. The ideal world is an artifact that serves as reference to the design of a real protocol between two parties, in the real world where the TTP does not exist. The security of the real protocol is then assessed by comparing the external effects of its execution in the real world vs. the external effect of executions of the ideal protocol in the ideal world. Specifically, a protocol is deemed secure, i.e., *simulatable*, if it can be proven that any external observer (called environment) is not able to distinguish in which of the two worlds an execution has taken place. If this is the case, then the real protocol is said to *emulate* the ideal protocol.

The simulation paradigm is useful in allowing a separation of the process of designing a real protocol from the process of defining desirable security properties. Actually, *simulatability* becomes the notion of security that incorporates all aspects of security (e.g., privacy, correctness, independence, non-transferability) that could be optionally enumerated from an analysis of the ideal functionality. It also provides composability guarantees that are useful for the modular design of large protocols. Specifically, it allows the security of a protocol to be proven while replacing its internal components by respective ideal functionalities.

Proving security may be particularly challenging in the *malicious model*, where one party may maliciously deviate from the protocol specification in an arbitrary and computationally bounded way. The security model may be also characterized by cryptographic assumptions, such as computational infeasibility to solve some mathematical problems with a reasonable amount of time and computational resources. Throughout this dissertation, security is considered in a *static*, *active* and *computational* model; i.e., at most one party is corrupted at the onset of the protocol execution, the corrupted party may deviate from the protocol specification, and both parties are limited to probabilistic polynomial time computations.

A successful approach for S2PC, initiated by Yao's protocol [Yao86, LP09], is based on garbled circuits — cryptographic versions of the Boolean circuit that computes the intended function. By replacing bits by random bit-string keys, garbled circuits enable oblivious evaluation of the Boolean circuit; i.e., an evaluation where nothing about the intermediate bits is learned, but the output bits are obtained. The typical hiding properties of garbled circuits allow a malicious constructor (P_A) to build an incorrect circuit without the evaluator (P_B) detecting it. A cut-and-choose approach can solve this problem: P_A builds several garbled circuits, and then P_B *verifies* some for correctness and *evaluates* the remaining to obtain the information necessary to decide a correct output from a consistent portion of the evaluated circuits. This is a somewhat efficient approach for S2PC protocols with a constant number of communication rounds (i.e., independent of the number of gates or the depth of the circuit) [KSS12, FN13]. Developing more efficient S2PC protocols thus deals with investigating new designs, which may allow efficient executions and lend themselves to provable security, possibly through innovation in the way in which to integrate some of its sub-protocols and primitives.

Of particular relevance to S2PC is the integration of commitment schemes — protocols that allow one party (the sender) to become bound to a choice but without revealing it to the other party (the receiver), and later to reveal it in a convincing way. An integration of S2PC and commitments may increase the usability of the input and output of S2PC in a broader protocol, e.g., allowing efficient linkage of several S2PCs, ensuring that the inputs and outputs of different executions are linked in a proper way. Commitments are also useful in enabling two-party coin-flipping (letting two mutually distrustful parties decide a random string) and zero-knowledge sub-protocols useful in proving correct constructions and/or proving knowledge of hidden elements.

1.2 Vision, thesis, goals and contributions

Vision and thesis. This dissertation and the underlying research was guided by the vision that *S2PC has a great potential as a tool to resolve certain conflicts between the need for privacy and the utility of sharing information, with consequential societal benefits.* The envisioned applicability can extend from personal applications (e.g., two persons engaging in personally-customized secure computations) to applications within critical information infrastructures impacting a significant portion of society (e.g., nation-scale privacy-preserving brokered-identification systems for high-assurance authentication of citizens in myriad services). This vision is accompanied with the perception that the current state-of-the-art in S2PC still does not provide practical solutions to many theoretically solvable problems. Such observation motivated academic research with a focus on improving efficiency and applicability of S2PC. As a result, this dissertation presents the following thesis:

S2PC can be made more practical by means of innovative cryptographic techniques, namely by engineered use of commitment schemes with special properties, enabling more efficient protocols, with provable security and applicable to make systems more dependable.

Goals. To validate the thesis, this dissertation presents contributions toward more practical S2PC. The document was designed to meet three main goals.

As a first goal, it contains a complete description of a protocol for S2PC-with-commitments, i.e., apart from certain primitives that are considered as "black boxes" (e.g., garbled circuits, a collision-resistant hash function, a pseudo-randomness generator). The exposition (including techniques that improve the efficiency of S2PC, such as the forge-and-lose technique) may be useful for researchers non-expert in S2PC and that may want to attempt a computational implementation. It includes published results as well as new extensions and a revised analysis, enabling diversity across types of trusted setup (e.g., public-key infrastructure and common-reference string), intractability assumptions (e.g., "integer-factorization cryptography" and "discrete-log cryptography") and simulatability (e.g., with and without rewinding).

As a second goal, the dissertation evinces that the practicality of S2PC may improve as a result of research in cryptographic primitives in adjacent areas of independent interest. This is substantiated by a separate discussion with contributions on two-party coin-flipping and

commitment schemes with special properties (simultaneously extractable and equivocable).

As a third goal, the discussion conveys that S2PC is also useful at a conceptual level, as a tool to reason about the privacy and security of a system in ways that might otherwise be ignored. S2PC is a useful element to have in the conceptual toolbox of security practitioners, namely as a possibly counter-intuitive concept that clarifies the ability of a party to compute a function without needing to know the full input. The usefulness of S2PC is evidenced in this dissertation by exploring its distinctive applicability in a practical application example of contemporary interest (brokered identification). Concretely, even if S2PC of Boolean circuits might not be the most efficient solution to solve the problem at stake, it allowed considering a solution at the level of requirements design, i.e., not forfeiting a privacy goal in detriment of an operational goal. By facilitating the initial discovery of a conceptual solution to a problem of brokered identification subject to certain design constraints, S2PC opens the door for further research that may look for possibly more efficient solutions.

Contributions. The technical contributions are organized along three main topics.

First, the dissertation introduces a new S2PC protocol [Bra13], using *bit commitments* in an innovative way. It enables efficiency improvements in comparison to traditional cut-and-choose of garbled-circuit approaches, most notably reducing by a factor of approximately three the number of garbled circuits needed to achieve a target level of statistical security in a malicious setting. The underlying technique, called *forge-and-lose*, also increases the deterrence against certain types of malicious behavior, i.e., beyond simple detection, by leading the malicious party to lose the privacy of her input. The protocol achieves S2PC-with-Coms, i.e., also produces commitments (Coms) of the private circuit input and output of each party, thus enabling applicability improvements, e.g., linked executions of S2PCs in a malicious setting.

Then, the dissertation introduces a new construction [Bra16] that improves the efficiency of instantiating a primitive needed by the original proposal of the S2PC-with-Coms protocol, namely simulatable two-party coin-flipping. More specifically, it describes a universally composable commitment scheme for large bit-strings, obtained by a novel combination of weaker extractable commitments and equivocable commitments associated with short bit-strings, and achieving communication complexity asymptotically close to the optimal rate.

Finally, the dissertation considers the use of S2PC-with-Coms as a tool capable of enhancing privacy in two security-related systems currently being developed for nation-scale

use. The problem at stake is that of hub-based brokered identification of citizens, where a centralized online party (a "hub") mediates the communication between several entities in order to facilitate authentication at service providers. If the hub were allowed to learn certain persistent user-pseudonyms (as is the case in contemporary governmental proposals in some countries), it would be able to track users across their accesses to service providers, which could be abused as a vector of mass surveillance. This dissertation describes [BCDA15] how S2PC of a block-cipher can be used to promote privacy, by preventing the hub from learning those pseudonyms across diverse authentications, while nonetheless allowing a tradeoff with properties needed for foreseen auditability and selective forensic investigations. While S2PC is still not a mainstream technique used in real systems, the study of this application is on its own useful by exemplifying the potential of S2PC as a privacy enhancer.

The next three subsections provide an overview of the results.

Out of scope approaches. The S2PC protocol revised in this dissertation is based on a cut-and-choose of garbled circuits approach, allowing a constant number of communication rounds and compatible with very efficient garbling schemes. Outside the scope of this dissertation there exist other promising S2PC approaches (some are briefly mentioned in the next Chapter), some of which allow different tradeoffs, sometimes asymptotically more efficient but sometimes impractical under common parametrization ranges.

Depending on the application context, the output learned from one or several S2PC interactions may pose a threat to privacy when combined with external information. Thus, an important complementary aspect to the design of S2PC protocols relates to deciding whether or not a function (or probabilistic functionality) is "safe" to compute, from a privacy-preserving perspective. These considerations are more closely related to the area of *differential privacy* [Dwo06], not discussed in this dissertation.

Another complementary aspect of secure computation is that of hiding from (at least) one party the function being computed [AF90]. This is commonly called *private function evaluation* and can be achieved directly via S2PC of a universal circuit [Val76, KS08c, KS16], where the actual evaluated circuit is specified as part of the private input of the universal circuit, or more specialized techniques [MSS14].

1.3 S2PC-with-Coms and the forge-and-lose technique

Traditional cut-and-choose of garbled circuits. The two parties in the interaction are denoted P_A (Alice) and P_B (Bob). In a traditional cut-and-choose of garbled circuits (e.g., [Pin03]), P_A builds several garbled circuits and then P_B (based on auxiliary information provided by P_A) *checks* the correctness of a random subset of them. If all checked circuits are correct, then P_B proceeds (based on other auxiliary information provided by P_A) to *evaluate* the remaining circuits to decide the circuit output from a consistent majority of the results. The majority criterion is needed because P_A cannot complain even if it detects inconsistencies (e.g., different outputs) — such complaint would jeopardize the privacy of P_B, because P_A could have prepared a few circuits for *selective failure* (e.g., error only if the first bit of P_A is 0). Thus, a successful cheating requires that P_A builds enough *bad* circuits (i.e., at least half of the number of circuits selected for evaluation) and be *lucky* that all of them are selected for *evaluation*. Specifically, the multitude of circuits induces a negligible probability with which a malicious P_A can make an honest P_B accept an incorrect output. In this protocol structure, an optimal cut-and-choose partition should have a fixed proportion of challenge types, of about three fifths for check and two-fifths for evaluation [SS11]. Concretely, at least 123 circuits [Bra13] (with 74 for check and 49 for evaluation) are needed to reduce to less than 2^{-40} the probability of successful cheating by means of building incorrect circuits, i.e., to obtain the typical benchmark of 40 bits of statistical security.

A new success criterion — improving efficiency. The traditional cut-and-choose was a reference point to improve the *efficiency* of S2PC. This dissertation describes the *forge-and-lose* technique, introduced in [Bra13], which enables reducing by an approximate multiplicative factor of three the number of circuits, which represents the main communication cost of cut-and-choose based S2PC protocols, for circuits of practical size. For example, 40 bits of statistical security can now be achieved with between 40 and 44 garbled circuits. This is achieved with a new criterion for successful evaluation, only requiring that at least one *evaluation* circuit is correct, instead of a majority. Thus, a successful cheating would now require P_A to guess the exact cut-and-choose partition in advance. As a result, each circuit provides an additive contribution of about one bit of statistical security.

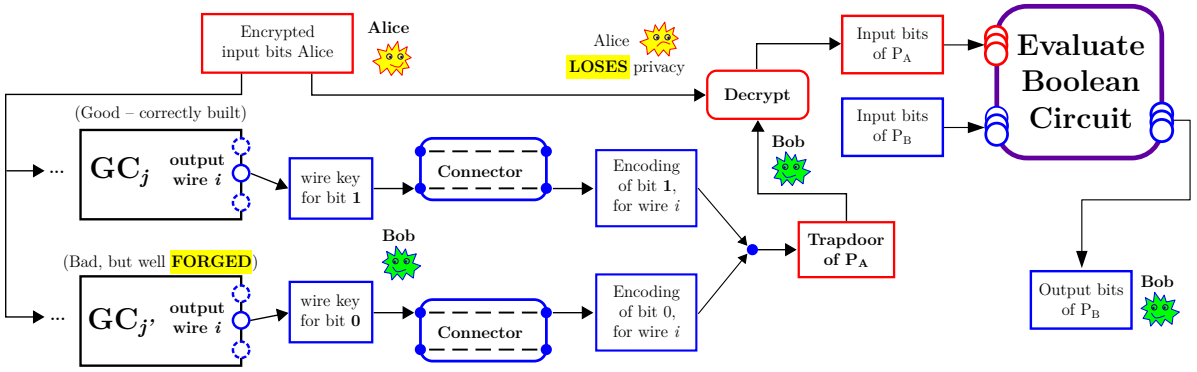


Illustration 1.1: Forge-and-lose. Evaluation path followed by P_B (the evaluator), if two different garbled circuits (e.g., GC_j, $GC_{j'}$) built by a malicious P_A and selected by P_B for *evaluation* lead to valid encodings of different bits in a same wire index (i).

The forge-and-lose construction. If P_A maliciously builds several incorrect garbled circuits, it may happen with a noticeable probability that all selected for check are correct but some selected for evaluation are inconsistent. Specifically, an evaluated circuit that in the view of P_B is seemingly correct in isolation may actually be *bad*, in the sense of leading to an output that is inconsistent to what would be obtained with a *good* circuit. Each such bad circuit, a.k.a. (bad but well-)*forged* circuit, will have at least one output wire index (i) whose output may differ from the output in the same wire index of a good circuit. The forge-and-lose technique leverages these inconsistencies to the benefit of the honest P_B, making any such pair equivalent to a trapdoor that is able to decrypt the input of P_A (which is verifiably encrypted as part of the protocol). In summary, if P_A *forges* some elements then it risks *losing* the privacy of her inputs — thus the name "forge-and-lose". The technique is illustrated at high level in Illustration 1.1, showing the path whereby two inconsistent outputs allow P_B to obtain the input of P_A and then use it to evaluate the intended Boolean circuit. The loss of privacy by P_A is not a violation of security, but rather a disincentive against malicious behavior. In the new structure, for P_A to succeed with a malicious forging it must guess in advance the exact cut-and-choose partition, in order to induce that all circuits selected for check are *good* and all selected for evaluation are *bad*.

BitCom approach. At the heart of the devised protocol is the use of *bit commitments* (BitComs) with trapdoor and other special properties. In a BitCom scheme, a party is able to become *bound* (i.e., committed) to a bit that is nonetheless *hidden* from the other party; the committed party also learns additional information (a.k.a. the "randomness" or "encoding")

that in a later *open* phase (if so desired) allows revealing the committed bit, in a convincing way, to the other party. A *trapdoor* is a secret that, under special circumstances and depending on the type of BitCom, may be disclosed to allow *extraction*, i.e., finding in advance what is the committed bit, without an *open* phase, and/or *equivocation*, i.e., opening any intended bit from a BitCom in the open phase. For the purpose of the forge-and-lose technique, P_A builds extractable (Ext) BitComs for her input bits, as well as equivocable (Equiv) BitComs for the output bits of P_B. Then, in each evaluated garbled circuit, each output wire key leads P_B to learn an encoding of a bit-opening of the Equiv-BitCom associated with the respective wire index. If a *bad* circuit is well forged then it will have at least one output wire key leading to a valid bit-encoding different from the one obtained from a *good* circuit. In special types of trapdoor Equiv-BitComs, different bit-encodings corresponding to the same BitCom can be combined into a trapdoor of another Ext-BitCom scheme. In the forge-and-lose technique the BitComs schemes are defined in a way that the trapdoor can be used as a key to decrypt the input of P_A from the respective Ext-BitComs. Overall, the use of BitComs in the protocol extends beyond the forge-and-lose technique. For each type of wire (input of P_A, input of P_B, output of P_B), the BitCom approach integrates in the cut-and-choose approach a statistically verifiable connection between BitComs and wire-keys, using a construction denoted *connector*. The connectors leverage certain homomorphic properties of the underlying commitments, namely the ability to produce and verify a BitCom of the XOR (i.e., sum modulo 2) of the bits committed by other BitComs.

S2PC-with-Coms — improving applicability. The new protocol achieves S2PC-with-Coms, as depicted in Illustration 1.2. Besides the output of a regular S2PC of a Boolean circuit, each party also receive commitments (Coms), possibly BitComs, of the private input and output of both parties, and the private randomness needed to *open* the Coms of her own input and output. The applicability goes beyond that of a S2PC of the underlying Boolean circuit, as it also allows parties to use the Coms as part of a larger protocol. For example, S2PC-with-BitComs enables a natural solution to linkage of several S2PCs, efficiently ensuring that the input used in a S2PC satisfies intended relations with the input and output of other S2PCs. As another example, this dissertation also discusses the usefulness of S2PC-with-Coms in an application (brokered identification) where Coms are externally useful.

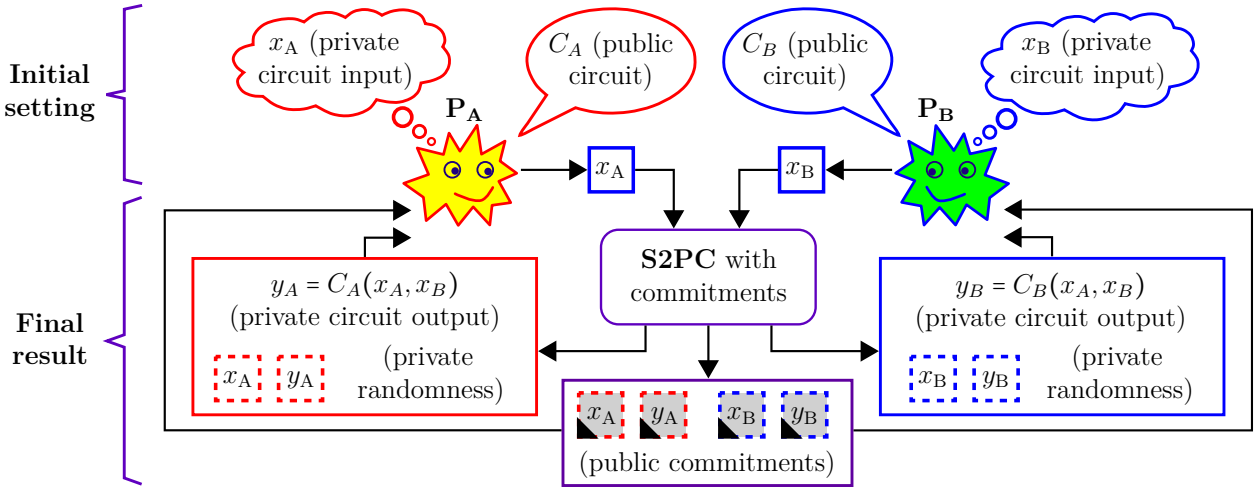


Illustration 1.2: Secure Two-Party Computation with commitments. Legend: P_A and P_B (the names of the two parties); x_p, y_p and C_p (the private circuit input, the private circuit output and the public circuit specification of party P_p, respectively, with p being A or B); ▩ (commitment of the variable inscribed inside the dashed square); ⬚ (randomness needed to open the respective commitment). Colors red, blue and purple are related with P_A, P_B and both parties, respectively.

Contemporary results. In respect to the three-fold reduction of number of garbled circuits, a comparable result was also achieved by other authors in independent contemporary research, by also just requiring that at least one evaluation garbled circuit is correct [Lin13, HKE13]. They follow different approaches, use different concrete primitives, do not consider the aspect of linkage and do not allow the same tradeoffs. For example: [Lin13] is inherently interactive after the first evaluation stage, whereas in [Bra13] the recovery path does not require further interaction upon being able to evaluate garbled circuits. [HKE13] requires use of a verifiable secret sharing scheme and both parties executing in parallel the same number of garbled circuits, and the recovery mechanism does not lead the malicious party to lose her privacy.

1.4 Simulatable coin-flipping and commitment schemes

Coin-flipping and commiments. To emulate an ideal random selection of BitComs in the new S2PC-with-BitComs protocol, the two parties need to execute, as a sub-protocol, a simulatable two-party coin-flipping. This is a probabilistic functionality that allows two mutually distrustful parties to agree on a common random bit-string of a certain *target* length. A protocol for two-party coin-flipping ("by telephone") was early proposed by Blum [Blu81]. It allows both parties to provide and combine independent *contributions* so that the

output bit-string of an honest party is indistinguishable from a random bit-string, even if the other party is malicious. The protocol uses the fundamental notion of commitment scheme (Com-scheme), allowing one party (P_1) to *commit* her own contribution before knowing anything about the contribution of the other party (P_2), but *hiding* it until the contribution of P_2 is revealed, and *binding* P_1 to only being able to *open* the committed value. The solution, in the form of a coin-flipping into a well (where one party learns the result before the other), sets the basis for what is hereafter denoted as the *traditional template*:

- **Step 1.** (*Commit* phase) P_1 commits to a contribution, but hiding it from P_2.
- **Step 2.** P_2 selects and sends his random contribution to P_1.
- **Step 3.** (*Open* phase) P_1 reveals her contribution to P_2 in a convincing way.
- **Step 4.** Each party outputs a combination of both contributions.

Simulatability. The simulatability of a coin-flipping protocol within the traditional template depends on the number of coins flipped in parallel, i.e., the length of the contributions, and on the type of Com-scheme. When flipping a single coin, any hiding and binding Com-scheme is enough if rewinding is allowed in the simulation [Gol04, §7.4.3.1]. Conversely, when doing parallel flipping of coins in number at least linear in the security parameter, or when considering a setting without rewinding, simulatability is facilitated by using Com-schemes with special simulatability properties, namely *extractability* (Ext) and *equivocability* (Equiv). In an *extractable commitment* (Ext-Com) scheme [SDCP00], a *simulator* is able to *extract* a contribution that has been committed by another party, in apparent conflict with the *hiding* property. In an equivocable commitment (Equiv-Com) scheme [Bea96a], a *simulator* is able to *equivocate* the opening to any contribution, namely to a value different from what had been committed, in apparent conflict with the *binding* property. The conflict is only apparent, as in comparison with a real party the simulator has extra power, such as capability to rewind the other party in the simulated execution, or knowledge of secret information (a trapdoor) obtained from some specially selected setup.

A main technical challenge is to obtain, in an efficient way, the combination of extractability (Ext) and equivocability (Equiv), as needed by the simulator to anticipate and control the outcome of a coin-flipping and/or commitment opening. In the initial description of the S2PC-with-Coms protocol [Bra13], an initial suggestion of simulatable sub-protocol for coin-flipping required a number of exponentiations linear in the number of input and output bits of both

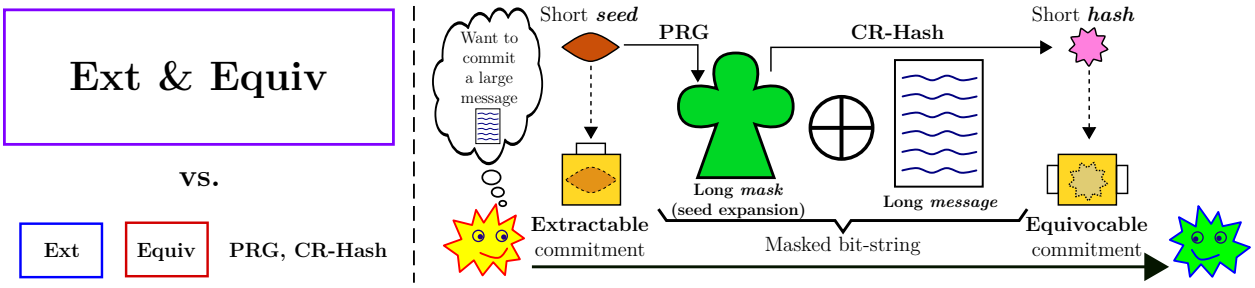


Illustration 1.3: Initial intuition for an efficient Ext&Equiv-Com scheme. Legend: Ext = extractable; Equiv = equivocable; PRG = Pseudo-randomness generator; CR-Hash = collision-resistant hash function. (Ext-Com — like a vault with a single door); (Equiv-Com — like a vault with several openings). The construction on the right side does not work, because opening the Ext-Com prevents equivocability, but it hints at how to achieve isolated extractability based on an Ext-Com of a short seed, and isolated equivocability based on an Equiv-Com of a short hash.

parties, and a linear communication complexity with a relatively large constant multiplicative factor (in respect to the target length). In comparison, further research [Bra16] presented in this dissertation improves efficiency (lower asymptotic communication complexity and a number of exponentiations independent of the target length) even without rewinding.

Simulatable two-party coin-flipping is also a problem of independent interest. A technical motivation relates to the design of larger cryptographic protocols, e.g., where it may allow the joint decision of a large *common reference string* needed as setup condition of follow-up sub-protocols [CR03]. It can also be useful for protocols whose probabilistic output needs to directly depend on random bit-strings, as exemplified with S2PC-with-Coms.

Contribution. An initial intuition for efficiency (depicted in Illustration 1.3) comes from two observations: (i) the *extractability* of a large string can be reduced to the extractability of one short seed, which can be expanded with a pseudo-randomness generator (PRG) into a long *mask* (a bit-string indistinguishable from random) that is then used to mask (encrypt with a one-time-pad) the large string being committed; (ii) the *equivocability* of a large string can be reduced to the equivocability of a short collision-resistant hash (CR-Hash) of whatever large string (e.g., the mask) the simulator wants to equivocate. Given the high efficiency of standardized PRGs [BK15] (e.g., based on block or stream ciphers) and CR-Hash functions [Nat15], the mentioned constructions would also be highly efficient. However, a simple triplet composed of a masked bit-string, an Ext-but-not-Equiv-Com of the seed of the mask, and an Equiv-but-not-Ext-Com of a hash of the mask does not result in an Ext&Equiv-Com of the

(unmasked) bit-string. For example, opening the Ext-Com would disallow equivocability.

Out of two newly developed protocols (in different simulatability settings) [Bra16], this dissertation describes the new universally-composable commitment scheme — Ext&Equiv in a simulatability setting that does not allow rewinding. It efficiently and securely combines the two separate Ext and Equiv properties, associated with a few commitments of short seeds and hashes (in number independent of the target length), into a unified property extensible to a much larger string, thus amortizing the cost of the base commitments. The protocol can be parametrized to achieve asymptotic communication rate arbitrarily close to 1, i.e., in order to commit or open a bit-string of asymptotically increasing length, the number of bits exchanged is larger than the bit-string by a multiplication factor that can be made arbitrarily close to 1. Computationally, the protocol requires collision-resistant hashing, pseudo-randomness generation and erasure encoding of a string of size linear in the target length.

The separate Ext and Equiv commitments for short strings can also be instantiated with a full-fledged Ext&Equiv-Com. In this case the construction represents a UC commitment extension, where a few (*commit* and *open*) calls to an Ext&Equiv-Com scheme for short bit-strings enable an Ext&Equiv-Com (*commit* and *open*) of a polynomially larger size.

Contemporary results. Other recent works [GIKW14, DDGN14, CDD+15] also focus on universally composable commitment schemes with communication complexity asymptotically amortized to rate 1. They explicitly use oblivious transfer as an ideal functionality in a hybrid model. In contrast, the distinct design described in this dissertation avoids explicit use of oblivious transfer, and instead uses base Ext-Com and Equiv-Com schemes (besides a PRG and CR-Hash function), thus enabling different tradeoffs in practical instantiations. At the cost of more interactivity, the Equiv-Com scheme can be based on an Ext-Com scheme. Based on other assumptions (common reference string and non-programmable random oracle) the protocol can be made non-interactive.

1.5 Privacy-preserving brokered identification

S2PC has the potential to solve apparent conflicts between privacy and the utility of sharing information. As a concrete example, this dissertation shows that S2PC-with-commitments can be applied to a practical and contemporary problem: enabling privacy-preserving nation-scale

brokered identification [BCDA15]. This constitutes an application example within a complex system with more than two parties, exhibiting the potential societal impact of the presented theoretical contributions. Here, S2PC can be used to enhance the privacy guarantees of citizens, notably creating resilience against a possible vector of mass surveillance.

Brokered identification. The main goal is to allow a user to authenticate to myriad service providers, without the user having to directly establish credentials with each service provider. This can be achieved by leveraging a pre-established relation between the user and an identity provider that can vouch for the user identity. In *privacy-preserving* brokered identification, the complementary goal is to prevent the intervening parties (such as identity providers) from tracking the activity of users across successive accesses to service providers.

A possible type of brokered identification involves an online central entity, called a *hub*, brokering the interaction between users, service providers and identity providers. Such centralized mediation is the basis of two concrete and contemporary systems being developed for nation-scale use: the Federal Cloud Credential Exchange [Uni13] (FCCX, recently re-branded as Connect.Gov) in the United States (US), and Gov.UK Verify [Ide13] in the United Kingdom (UK). As a broker, the role of the hub is to ensure interoperable identification and authentication, while seemingly offering desirable privacy and security guarantees, such as hiding from the identity provider the user activities across service providers. However, in the two analyzed systems, the hub has a wide-tracking capability, on top of other problems such as attribute visibility and capability of impersonation. The identified privacy and security problems are in sharp opposition to the guidelines, requirements and/or principles that the systems should be aligned with, given the underlying strategies that they are meant to follow, namely: the National Strategy for Trusted Identities in Cyberspace (NSTIC) [The11] in the US, and the Identity Assurance Principles (IAP) [Pri14] in the UK.

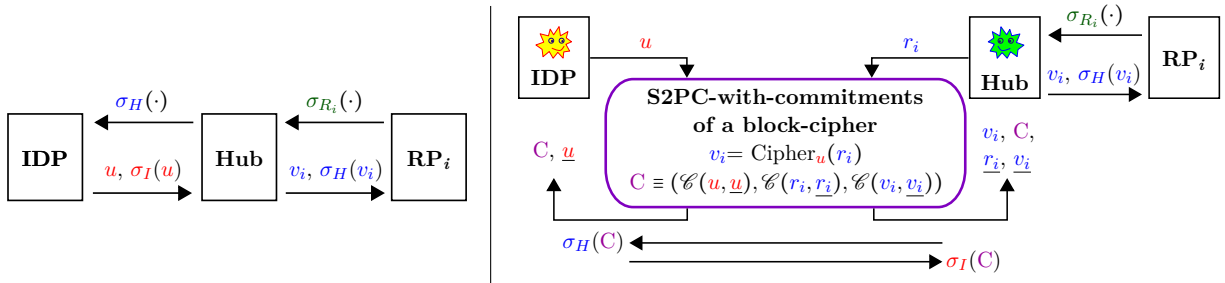
In respect to brokered identification, the results reported in this dissertation are focused on a specific privacy property, here called "weak pseudonym-unlinkability by the hub." This aspect of *unlinkability* considers the inability of the hub to infer whether two identification transactions, by the same user but at different service providers, were performed by the same user or by different users, even if no other private identifiable information is known about the user(s). While an identification transaction is composed of many identifiers and metadata, the essence of the problem discussed herein is related only to user-pseudonyms,

i.e., identifiers that in isolation do not reveal any personal data about a user, but which may constitute "relationship data" when linked across several events. It is shown that S2PC with commitments can be used to enable such unlinkability property while at the same time still enabling the overall system to operate "in a secure and auditable manner." Following standard notation, service providers are hereafter denoted as *relying parties* (RPs), since they will rely on identity assertions provided by someone else (the hub) about the user.

Pseudonym unlinkability. The problem and solution are illustrated in Illustration 1.4.

Illustration 1.4a shows that in the current systems (FCCX and Gov.UK Verify) the user pseudonym (u) defined by the identity provider (IDP) is first sent to the hub and only then transformed into a user pseudonym (v_i) to be sent to a respective RP. The transformation is left implicit, as it is different in Gov.UK Verify (where another entity is involved) and FCCX. The later user pseudonym is persistent and different for each RP. In other words: across several transactions of the same user (u) trying to authenticate to the same RP_i, the hub will always send the same user pseudonym (v_i) to RP_i, so that RP_i can find the same local user account; each RP receives a different user pseudonym. This pseudonym transformation prevents RPs from learning a global persistent identifier (u) of the user. If assuming that the user pseudonym for each RP is pseudo-random (or randomly selected the first time and then memorized), then it follows that it does not reveal identifiable information about the user, and it does not allow a set of colluding RPs to link user transactions based on these pseudonyms. However, an essential problem remains — the hub always sees the user pseudonym (u) defined by the IDP, which is the same whenever the same user authenticates via the same IDP. This poses a privacy shortcoming, in the sense that the hub gains a wide ability to *link* all identification transactions associated with each user.

It could seem at first glance that the described *linkability* is a "necessary evil" in order to allow an adequate transformation of the user pseudonym (u) received from the IDP into a new user pseudonym (v_i) to send to each RP_i, without letting the IDP know which RP is involved. However, S2PC enables an alternative elegant solution, allowing the needed pseudonym transformation, while hiding the initial user pseudonym (u) from the hub. In the right side, Illustration 1.4b shows how this can be achieved based on a S2PC of a block-cipher. Specifically, the user-pseudonym v_i to be sent to RP_i is obtained as a ciphertext resulting from using the user pseudonym u defined by the IDP as a key to encipher the RP pseudonym



(a) In FCCX and Gov.UK Verify **(b) New proposal**

Illustration 1.4: Linkability vs. weak-unlinkability of user-pseudonyms by the hub. Legend: u (user pseudonym determined by IDP); r_i (pseudonym of RP$_i$, determined by Hub); v_i (user pseudonym to send to RP$_i$); $\mathscr{C}(\cdot,\underline{\cdot})$ (commitment of $\cdot$, using randomness $\underline{\cdot}$); $\underline{u}$, $\underline{r_i}$, $\underline{v_i}$ (randomnesses associated with commitments of u, r_i, v_i); σ_H (signature by Hub); σ_I (signature by IDP); σ_{R_i} (signature by RP$_i$). Several metadata elements (e.g., session identifiers) are left implicit.

r_i defined by the hub. By definition a block cipher has a pseudo-random output when its key (in this case the user-pseudonym u defined by the IDP) is also pseudo-random. Since the computation is made via a S2PC, the hub does not get to learn a common user pseudonym across two executions of the same user across different RPs.

Auditability. The design of a complex privacy-preserving brokered identification system is also subject to auditability considerations. For example, it may be required that the hub be able to prove, in certain audit actions, that the user pseudonym v_i sent to RP$_i$ in a given transaction was indeed obtained upon interaction with an IDP, namely by showing a related cryptographic signature by the respective IDP. Similarly, the IDP must be able to prove that it only signed said material as a result of an interaction with the hub. In order to enable these auditability properties, a more intricate solution is required (still depicted in Illustration 1.4b). Besides the already described block-cipher output received by the hub, both the IDP and the hub also receive commitments of the inputs and outputs. The commitment hides the inputs and outputs of each party from the complementary party, but allows each party to open (or prove something about the values committed by) the commitments respective to her private input and output. By having each party send a signature of the commitments to the other party, each party is later able to prove the needed relations during an audit action. This can be based on the already mentioned S2PC-with-BitComs protocol described in the dissertation, but a complementary solution is also described, built on top of any black-box S2PC protocol. The different solution slightly augments the circuit that needs to be evaluated, but otherwise leaves the computation of cryptographic commitments outside of the actual S2PC module.

A complete solution for privacy-preserving brokered identification, resilient to a compromised hub, requires integrating other properties, e.g., a stronger form of pseudonym unlinkability (with the hub not learning any persistent user-pseudonym, not even the one needed by the RP), as well as attribute privacy and authenticity against (i.e., not allowing user-impersonation by) a malicious hub. Nonetheless, the example of weak pseudonym-unlinkability across RPs already highlights the applicability of S2PC, demonstrating that two apparently conflicting requirements may actually be satisfied simultaneously, while allowing a desirable level of auditability. Even if more efficient solutions may exist, S2PC as an ideal functionality is useful to show conceptual feasibility and thus can stand as a starting point for research of more specialized solutions (e.g., other than S2PC of a Boolean circuit).

1.6 Organization

The remainder of this dissertation is organized as follows. Chapter 2 introduces the necessary background notions, namely some cryptographic assumptions, the ideal/real simulation paradigm, commitment schemes, the cut-and-choose of garbled circuits approach for S2PC, and some oblivious transfer protocols. Chapter 3 describes the protocol for S2PC-with-Coms, including the forge-and-lose technique, with a revised analysis and improvements in comparison with the original paper. Chapter 4 presents a new efficient universally composable commitment scheme (usable for two-party coin-flipping). Chapter 5 analyzes two brokered identification systems, showing that S2PC can play the role of privacy enabler. Chapter 6 concludes with a review of the goals proposed in this dissertation and enumerates open problems. Appendix A contains details about zero knowledge (ZK) sub-protocols. Appendix B provides a low-level description of the S2PC-with-Coms protocol. Appendix C describes further details about simulatable commitments and two party coin-flipping.

Chapter 2

Background

This chapter discusses background material useful for the understanding and instantiation of the contributions presented in this dissertation. The goal is to convey the needed intuition about used primitives and concepts, serving as a technical introduction, helping to define the scope of contributions and initiating a comparison with related work. It does not intend to be a tutorial of cryptography nor to discuss aspects in full formality.

Section 2.1 reviews background material about standard cryptographic assumptions and primitives, namely associated with *integer-factorization cryptography* (IFC) and *discrete-log cryptography* (DLC). Section 2.2 overviews the ideal/real simulation paradigm, used to prove security (a.k.a. simulatability) of protocols. Section 2.3 reviews basic notions about commitment schemes, including BitCom schemes, homomorphic and pseudo-homomorphic properties, extractable and equivocable instantiations. Section 2.4 describes at high level the basic building blocks of the traditional cut-and-choose of garbled-circuits approach, including nuances on how to decide the cut-and-choose partition, and the application of the random-seed checking technique. Section 2.5 describes two BitCom-based constructions of oblivious transfer useful for S2PC. Zero knowledge sub-protocols are discussed in Appendix A.

2.1 Cryptographic assumptions and primitives

Within the *standard model* of cryptography, some mathematical problems are assumed to be computationally intractable, i.e., infeasible to solve in a practical amount of time with a practical amount of computational resources. The assumptions herein are taken at the light

of current state-of-the-art computability, not in the sense of conjectures.

Informally, the computational security parameter κ, specified in number of bits, denotes a value whose respective exponentiation over a base 2 represents a prohibitive number of computational operations. Similarly, the number of bits σ of statistical security is the additive inverse of the logarithm base 2 of the maximum error probability, i.e., for which a malicious party can break some security property in a single execution (e.g., make the other party accept an incorrect output) and/or the maximum advantage in a game of distinguishing real protocol execution transcripts from transcripts of an ideal execution.

This section starts by briefly mentioning base cryptographic primitives that are available based on highly-efficient standardized constructions and are directly assumed to be secure: a pseudo-randomness generator (PRG) based on block-ciphers, and a collision-resistant compressive hash function (CR-Hash) (§2.1.1). Then it describes number-theoretical assumptions and instantiations, suggestively labeled as being of integer-factorization cryptography (IFC) type or discrete-log cryptography (DLC) type. The IFC case is considered here with a concrete instantiation based on Blum integers (§2.1.2). The DLC case is itself sub-instantiable by regular multiplicative groups modulo a large prime number, a.k.a. finite-field cryptography (FFC), and by elliptic curves over finite fields, a.k.a. elliptic-curve cryptography (ECC) (§2.1.3). (The acronyms IFC, FFC and ECC are for example used in [Bar16].)

2.1.1 Concrete cryptographic primitives

A *negligible function* is a positive function approaching zero faster than the inverse of any positive polynomial, asymptotically as its real pre-image approaches infinity [Bel02]. *Negligible probability* denotes a probability that is a negligible function of an implicit security parameter. *Overwhelming probability* denotes a probability whose distance to certainty is negligible in the implicit security parameter. *Noticeable probability* denotes a probability higher than the inverse of some positive polynomial, asymptotically as the security parameter increases, i.e., such that any polynomial number of repetitions yields an overwhelming probability.

The usefulness of polynomial-based complexity for defining certain properties stems from the closure properties of polynomials. For example, if an event has a negligible probability of occurrence (e.g., a randomly selected element having a certain property), then it follows that the probability of occurrence remains negligible even after a polynomial number of repetitions. Asymptotic-based definitions are useful for security reductions, but do not hint at concrete

parameters to use. In order to estimate the concrete complexity of possible implementations of protocols this dissertation assumes some practical choices based on standardized parameters.

PRG. A pseudo-randomness generator receives a fixed-length input seed, assumed to be random (or pseudo-random) and is then able to pseudo-randomly generate (PRG) an arbitrary number of elements from a specified set, e.g., bit-strings of a certain length. The input length is assumed to be proportional to the computational security parameter, and in practice is always used as twice its size. For example, for 128 bits of security, AES-256 is used as the base cipher for generation of pseudo-random elements, guaranteeing a *birthday bound* of about 128 bits of security in respect to the probability of ever repeating the same random seed. A PRNG is secure if any probabilistically polynomially time bound adversary has a negligible advantage in a game of distinguishing PRG outputs from actual random outputs.

CR-Hash. A collision-resistant hash function (CR-Hash) maps the domain of arbitrary-length bit-strings into the set of bit-strings of some fixed length, and satisfies the property of being computationally infeasible to find collisions, i.e., two inputs with the same hash. The CR-Hash must have output length at least twice of the computational security parameter, lest its collision resistance would break under a generic *birthday attack* (i.e., generating many pairs of random pre-images and respective hashes until finding two distinct pre-images with the same hash). Whenever mentioning CR-Hashes it will remain implicit that one-wayness is also intended, namely that upon receiving the hash of an input selected uniformly at random, from the set of bit-strings of sufficiently large length, it is infeasible to find a respective pre-image. In practice, it is assumed that SHA-256 delivers the intended properties of collision resistance and one-wayness (roughly) up to 128 bits of security.

A formal definition of collision resistance and one-wayness usually considers a family of hash functions, parametrized by a security parameter, such that asymptotically as the security parameter grows it is computationally infeasible to obtain collisions. However, the use of CR-Hashes in this dissertation is very concrete and fixed at setup. As opposed to asymptotic-based security, *concrete security* measures the concrete computational resources (e.g., number of calls to a certain primitive) needed to break a cryptographic assumption based on the ability to break a security property. Given the compressive nature of the CR-Hash (e.g., outputs with 256 bits), theoretical collisions certainly exist (e.g., pairs of values with the same CR-Hash), but for a concrete hash function (such as currently for

SHA-256) they are assumed to be unknown and infeasible to compute. In an eventual future where a single collision is disclosed, or where theoretical advances shed reasonable suspicion that the practical security of the hash may be breakable, the CR-Hash function must change to adapt to the respective state-of-the-art. Whenever explicitly mentioned, SHA-256 may also be used as a cryptographic hash assumed to have a type of pseudo-random property and some correlation intractable properties. For example, it may be assumed that the output is pseudo-random whenever the input is unpredictable, e.g., whenever prefixing to the input a bit-string (a salt) with a large enough known number of bits selected uniformly at random.

Ideal primitives. Several instantiations of protocols are described in a hybrid model with access to ideal functionalities, as better discussed in Section 2.2 when reviewing at high level the ideal/real simulation paradigm. For example, a protocol implementing a commitment scheme with special properties (extractability and equivocability), and designed for the domain of large bit-strings, may be defined and proven secure based on the use of a few ideal commitments for shorter bit-strings. The ideal functionalities may then be instantiated based on other ideal functionalities, e.g., a common reference string and/or a (non-programmable) random oracle, and/or additional real concrete primitives. The relevance of definitions based on ideal functionalities is arguable when there exist no concrete instantiations to realize them, or when there exist no provably secure instantiations based on standard assumptions [Dam07]. A special case is the programmable random oracle functionality [FS87, BR93], for which it is proven that any replacement by a concrete function fails to provide all properties of the oracle [CGH04, Nie02]. It is thus relevant to consider which properties of the ideal functionality are required to prove security, to understand if there may exist a real cryptographic primitive adequate as a concrete replacement.

Non-programmable random oracle (NPRO). A NPRO may be understood as a black-box providing access to the evaluation of a single random function, which no party (or simulator) has access to program [Nie02]. It is a function decided once and for all, before any protocol execution, and whose output can only be determined upon providing the respective input to the black-box. A NPRO may be suitable to support a soundness property that depends on the inability to compute in advance a value (a pre-image) that is related in a particular way with an unpredictable value (the respective NPRO image). For example, it can be used, with care, to enable non-interactive versions of typical Σ-protocols, e.g.,

zero-knowledge proofs of knowledge (ZKPoKs) with a commit-challenge-response structure and having a special soundness property (see §A.1.2 in Appendix). The transformation for non-interactivity consists on defining the Σ-*challenge* element to be the NPRO image of an *equivocable commitment* of a tuple of elements that define the problem context and include the Σ-commit element [Lin15]. The intuition for soundness is that the NPRO guarantees that at the time of selecting a new Σ-commit element the corresponding Σ-challenge is unpredictable (i.e., assuming that the protocol parameters are being used for a first time), which makes it similar to being decided at random. Naturally, if the cardinality of the challenge space is within the reach of a brute-force attack (e.g., if the statistical security parameter is too small to be suitable as a computational security parameter) then the transformation is unsuitable to support soundness. The above reasoning is able to isolate soundness from other security properties because, in the mentioned example, the NPRO properties are really only required to prove soundness, whereas other properties such as zero-knowledge (including non-transferability) depend on the use of an (equivocable) commitment scheme to prepare the pre-image of the NPRO, which for example semantically hides the respective input.

Cryptographic hash vs. NPRO. In comparison with certain uses of a full-fledged (programmable) random oracle, the use of a NPRO in certain protocols leaves open the possibility of replacing the oracle by a concrete function with certain correlation-intractability properties sufficient to prove security of the protocol. In the example of ensuring soundness of a non-interactive version of a concrete Σ-protocol, an appropriate replacement of the NPRO by a concrete cryptographic hash function must be such that it does not increase the advantage of non-interactively producing a valid commit-*challenge*-response triplet. Once replacing the NPRO by a concrete hash function, the problem of finding a valid commit-challenge-response triplet can be reduced to that of finding, for a particular function that depends on the hash function and the protocol at stake, an input-output pair that is correlated in a specific way that also depends on the protocol. Specifically, in the non-interactive transformations considered in this dissertation it is assumed that there is a concrete cryptographic hash function (e.g., SHA-256) that is not only collision resistant, one-way and pseudo-random (whenever the input is unpredictable) but also correlation intractable in a specific way related to the protocol that uses it.

From a definitional standpoint, the arguable problem of the mentioned NPRO replacement is that the requirement of a specific correlation intractability per protocol is somewhat circular,

in the sense that the proof of security of the protocol depends on the hash function being assumed to satisfy some specific properties whose definition is based on the protocol itself. While it remains an open problem to prove that a concrete cryptographic hash (e.g., SHA256) may securely replace a NPRO in the concrete applications considered herein, at the light of current state-of-the-art this is an arguably acceptable tradeoff, as it allows a modularization of security concerns and allows understanding how a protocol may fail (i.e., by breaking explicit assumptions). It would indeed be very surprising to show that a standardized particular cryptographic hash function (e.g., SHA-256) could introduce, in the particular considered protocols, a flaw not existent when using instead a NPRO — any such example would represent finding a new undesirable property of the concrete standardized cryptographic hash.

It is worth emphasizing that the above reasoning is applied for particular protocols that allow specifying a concrete intractability assumption about the function replacing the NPRO. The reasoning would provably fail for cases where security relies on the ability to program the output of a random oracle. Particularly, properties like deniability (a.k.a. non-transferability, an aspect of zero-knowledge), which (in non-rewinding simulations) require a kind of programmability, are in this dissertation supported based on other ideal functionalities (weaker than a programmable random oracle), such as equivocable commitments (for which it is possible to program, i.e., equivocate, the committed value), e.g., which can on its turn be based on a (programmable) common reference string.

The NPRO is used to enable non-interactive sub-protocols but can be avoided in interactive versions thereof. More considerations are elaborated in §A.1.4 in Appendix.

2.1.2 Integer-factorization cryptography (IFC)

A Blum integer [Blu81] is a positive integer composed only of the product of two prime powers, where each prime is congruent with 3 modulo 4 and each power has an odd exponent. Within the use cases of this dissertation, it is enough to consider a Blum integer as equal to the product of two primes. Blum integers can thus be produced by selecting two random primes congruent to 3 modulo 4 and then multiplying them. Once the number of bits (a.k.a. the size) of a prime is decided, selecting a respective random prime can be done somewhat efficiently. First, because the probability that a random integer of a given size is prime is approximately inversely proportional to the size of the prime. Second, because checking primality can be done with a number of modular exponentiations linear in the statistical

Table 2.1: Concrete sizes for IFC, FFC and ECC parameters

A	B	C	D	E	
Security parameter (κ)	Type of elements	IFC	DLC		1
			FFC	ECC	2
256	Group elements	15,424	15,424	512	3
	Pre-images	15,424	512	512	4
128	Group elements	3,248	3,248	256	5
	Pre-images	3,248	256	256	6
96	Group elements	1,776	1,776	192	7
	Pre-images	1,776	192	192	8

The case $\kappa = 96$ is not intended for long term security, but may be useful when a security property only needs to hold in the short-term, e.g., binding of a commitment produced and opened within a brief protocol execution (e.g., see §B.4.1.3). In IFC the pre-images are square-roots; in DLC they are integer exponents. Future progress in theory, algorithms and computer technology may change the adequate lengths (and, hypothetically, even invalidate some intractability assumptions).

security parameter, e.g., based on the Miller-Rabin probabilistic primality test [Rab80].

Integer-factorization (IF) intractability assumption. Given a large enough Blum integer selected uniformly at random, restricted to be the product of two primes of equal size, it is here assumed to be computationally infeasible to find its prime factors.

Decisional quadratic-residuosity (DQR) intractability assumption. Given a large enough Blum integer with unknown factorization, it is here assumed to be infeasible to distinguish with noticeable advantage whether an element was selected as a random square or as the additive inverse of a random square.

Parameters. Table 2.1 (based on [SBC+12, Table 7.2]) suggests Blum-integer sizes for diverse security parameters. For example, for 128 and 256 bits of (symmetric) security, it is here assumed that it is enough to use Blum integers selected with 3,248-bits and 15,424 bits, with the primes being randomly selected subject to having the respective half length.

Correctness of Blum integers. It is an interesting open problem whether or not there exists an efficient procedure to decide whether a large random integer (equal to 1 mod 4, not

a square and not a prime power) is a Blum integer or not. Not knowing such a procedure, certain uses of Blum integers in this dissertation require that a party that knows its prime factors proves to another party that the integer is correct and/or that it knows its factors, but without revealing anything else. This is done via a zero knowledge proof (ZKP) of correctness. Actually, even in a context where a Blum integer is selected by a trusted setup, subtle security reasons (simulatability) require that a party knowing the Blum integer trapdoor (its prime factors) provides a zero-knowledge proof of knowledge (ZKPoK) of correct factorization (e.g., [vdGP88]). Appendix A.2 describes problems in case of using a non-Blum integer and analyzes suitable ZK sub-protocols.

2.1.2.1 Other properties of Blum integers

Given the use of Blum integers in the original forge-and-lose technique, and also its special properties that enable a 2-out-of-1 oblivious transfer for the input wires of P_B, it is useful to review here a few background properties. A more in depth analysis of these and other number theoretical properties can be found in the literature (e.g., [NZM91]).

Class of a group element. The set of non-negative integers which are co-prime to and lower than the Blum integer, together with the operation of multiplication modulo the Blum integer (simply denoted *multiplication* or group multiplication), forms a multiplicative group. The group elements are also denoted as residues. For a fixed Blum integer, the Jacobi Symbol is a completely multiplicative function that maps any group element into 1 or −1. It can be computed efficiently without knowing the factorization of the Blum integer. Since the Jacobi Symbol is multiplicative (over $\{-1, 1\}$), but S2PC is more focused on bits (0 and 1), it is easier to consider the *class* of an element as the respective homomorphism to XOR (i.e., additive over $\{0, 1\}$) — class 0 means Jacobi Symbol 1, and class 1 means Jacobi Symbol −1.

Blum integers have interesting properties that distinguish them from other integers. In particular, they are the only moduli that simultaneously satisfy the following 2 properties:

- *Each quadratic residue has exactly 4 distinct square-roots, two of which for each possible Jacobi Symbol (*−1 *and* 1*)*, i.e., for each class (0 and 1).
- *The residue* −1 *is non-quadratic and has Jacobi Symbol 1 (i.e., class 0).*

Proper elements. From the above properties, it follows that the additive inverse of any residue is a different residue with the same square and the same class. Two group elements trivially correlated in this manner have different least significant bit, i.e., one is even and the other is odd (when represented as a non-negative integer lower than the modulus). When dealing with square roots it is useful to define a *proper* square root of a given class, out of the two possible with the same square and same class. For example, it may be assumed that a *proper* element is the minimum between itself and the modular additive inverse. Other options could be to choose the odd one, or the one whose least significant bit is equal to its class. It is trivial to adjust group multiplication to only operate on *proper* elements.

The knowledge of the prime factorization of a Blum integer is computationally equivalent to the knowledge of a non-trivial square-root of 1 (i.e., with class 1, i.e., Jacobi symbol −1) in the respective multiplicative group. Any residue multiplied by this value is converted into a new residue with different class but same square.

If a party knowing the factorization of a Blum integer is able to assume that for the other party it is infeasible to find the factorization and to decide quadratic residuosity of random residues, then the modulus can be used as an instantiation basis for several components of the S2PC-with-Coms protocol defined in this dissertation.

It is useful to notice three other properties related to square-roots: (i) the knowledge of two square-roots of different class enables an efficient discovery of the prime factorization of the Blum integer (upon a simple computation of a modular sum and the computation of a greatest common divisor, e.g., using the Euclidean algorithm); (ii) the knowledge of the prime factorization enables efficient discovery of all four square roots of any quadratic residue; (iii) for each quadratic residue there is only one square-root that is itself a quadratic residue — this is called the *principal* square-root.

2.1.3 Discrete-logarithm cryptography (DLC)

Another type of cryptographic instantiation considered in this dissertation relates to cyclic groups, i.e., groups that can be generated from a single element (a *generator*), by repeated application of the group operation. In multiplicative notation the group set is defined as the set of powers of the generator. Correspondingly, the *discrete logarithm* of any group element is the least non-negative integer exponent that produces said group element as a result of exponentiating the generator. The number of elements in the group is denoted the group order.

Discrete logarithm (DL) intractability assumption. For certain group representations, i.e., once fixed a base generator and specifying the group operation (multiplication), it is here assumed to be infeasible to compute the discrete logs of random group elements.

The group representation is relevant because for any cyclic group there is a group representation for which the assumption does not hold. For example, any finite cyclic group is isomorphic to the additive group of integers modulo the group order. Here, in additive notation, modular multiplicative inverses — the analogous of discrete logs in multiplicative notation — can be computed efficiently, e.g., based on the extended euclidean algorithm.

Decisional Diffie Hellman (DDH) intractability assumption. Given a reference triplet composed of the generator and two respective random powers (i.e., with unknown random exponents), it is hard to distinguish another random group element (i.e., with unknown random exponent) from a group element whose discrete log is the product of the exponents of the reference triple.

The DDH assumption is strictly stronger than the DL assumption, in the sense that an algorithm for computing discrete logarithms can be converted to one that solves the DDH problem, whereas the converse is not known to be true for several groups of interest. The two assumptions are assumed to be true for two types of (properly instantiated) groups:

- **FFC (finite field cryptography).** Cyclic subgroups over finite-fields represented over the integers modulo a large prime number, e.g., after selecting a large random prime equal to one plus the double of another prime, selecting the group generated by any quadratic residue different than one. In order to allow shorter exponents (e.g., with 256 bits), the prime can be selected as one plus the double of a product of two odd primes, where one of the primes is of the desired length (e.g., 256 bits) The group is then defined as an element with verifiable prime order of the given size.
- **ECC (elliptic curve cryptography).** Cyclic subgroups of elliptic curves over finite fields. While the theory of ECC is out of the scope of this dissertation, their high level properties with respect to DLC assumptions (DL and DDH) are identical to FFC, and the treatment of the set of exponents is similar (i.e., non-negative integers less than the group order). It is thus enough to mention here that, for the same security parameter and same DLC assumptions, ECC has the advantage of allowing shorter representations, with each group element and exponent having up to twice the size of the security parameter.

Parameters. Suggested group-element sizes and key-sizes (e.g., [SBC+12, Table 7.2]) can be found for diverse values of security parameter. For appropriate ECC curves the size of the group elements and pre-images (exponents) are the double of the security parameter (i.e., 256 and 512 bits), respectively for 128 and 256 bits of (symmetric) security. For FFC the sizes of group-elements are as for Blum integers (i.e., 3,248 and 15,424 bits), but the size of exponents is as in ECC. Instantiations based on FFC are thus likely to induce higher communication complexity than instantiations based on ECC. For 128 bits of security, the size of group elements is between 12 and 13 times larger, and for 256 bits the factor is about 30. However, when certain protocols can be adjusted to have most of the communication burden associated with exponents, instead of based group elements, then FFC may require approximately the same communication as ECC. Since FFC has (arguably) a simpler description and theory in comparison with ECC, in practice there are uses for which it may be adequate to use FFC, namely if/when it is the only DLC instantiation supported in a certain system.

Conversely, ECC may be preferable when there is an available implementation. For example, Curve25519 (over the prime field with a modulus $2^{255} - 19$) is regarded as a choice allowing secure and highly fast implementations [Ber06]. The order of the group has size equal to 252 bits, thus allowing the space of exponents to encode bit strings with up to 252 bits. Other curves could be selected to allow committing strings up to 256 bits, but a practical tradeoff between efficiency (existing implementations) and test-of-time based security may, at implementation time, lead to different decisions. For example, if an available ECC implementation allows only committing bit-strings up to 252 bits, then an application that requires committing a CR-Hash, where the hash would typically have 256-bits, may consider truncating the hash to 252 bits, still assuming it is collision resistant. Conversely, if a 256-bit input to a circuit needs to be committed in an extractable way (e.g., using an ElGamal commitment), then the 256 bits need to be preserved. In this case a larger group is required, e.g., either via a different curve (with just a slight increase in representation size, e.g., 264 instead of 256 bits), or by using the same curve twice, committing the 256 bits into a vector with two commitments, each committing to 128 bits.

Remark 2.1 (Parameters may change with time). The adequate lengths to propose for IFC and DLC may change based on progress in future developments in theory, algorithms and concrete results in benchmarking challenges. For example, integer factorization of 768-bit integers is already within the reach of "academic effort" [KAF+10], and 1024-bit integers are

no longer recommended as secure [Bar16]. Hypothetically, some intractability assumptions may also be falsified if/when new algorithms are discovered or other computation paradigms (e.g., quantum computers) are implemented with the ability to solve in feasible time the underlying mathematical problems.

2.2 Ideal functionalities

This section reviews at high-level the ideal/real simulation paradigm (§2.2.1), describes types of global and local trusted setup (common reference string and public-key infrastructure) (§2.2.2), and defines ideal functionalities for commitments (§2.2.3) (including properties and notation) and S2PC-with-Coms (§2.2.5).

2.2.1 The ideal/real simulation paradigm

In the ideal/real simulation paradigm [Can00, Can01], a real protocol Π is considered secure, a.k.a. *simulatable*, if it *emulates* a respectively intended *ideal functionality* $\mathcal{F}$. The ideal functionality defines the behavior (rules of interaction) of a *trusted third party* (TTP) that in an ideal world mediates the communication and computation between the two regular parties. For example, in an ideal S2PC, where the TTP mediates all communication and computation, it is intuitive that the following properties are satisfied: *privacy* of inputs and output of each party; *correctness* of the computation; *independence* of inputs decided during a computation i.e., except for apriori externally determined correlations; *non-transferability* (a.k.a. deniability), i.e., inability of a party to externally prove that her output was obtained from an actual computation with another party, except what may be proven directly from the private input and private output. Thus, if it is proven that a real S2PC protocol emulates an ideal S2PC functionality, it follows that it satisfies all above-mentioned properties.

Adversarial model. Proving that a real protocol is secure against a class of adversaries amounts to show that for any allowed adversary in the real world, denoted *real adversary* $\mathcal{A}$, there is a respective allowed adversary in the ideal world, denoted *simulator* $\mathcal{S}$, that induces in the ideal world a distribution of outputs that is indistinguishable from the respective distribution in the real world. This dissertation considers static and computational active (a.k.a. malicious) adversaries — the adversary must choose in the start of the execution which

party to corrupt, and then it controls said party to deviate from the protocol specification, in an arbitrary way but computationally bounded to probabilistic polynomial time.

In the setting of multi-party computation, an adversary (in the real or ideal world) is defined as a separate entity that corrupts a regular party, taking control of the actions and internal states of said corrupted party. In the setting of two-party computation with static corruptions, when considering the case where at most one party is corrupted at the onset of the protocol execution, it is possible and simpler to equate the adversary to the corrupted party. Thus, hereafter the expression *malicious party* is used to denote the combination of the corrupted party and the corrupting adversary. A malicious party may behave arbitrarily, possibly attempting to disrupt the joint output distribution of an interaction. In particular, while the output of an honest party is as prescribed by the protocol, the output of the malicious party may include her complete view of the execution, including her internal states of memory, as well as any efficiently computable transformation thereof.

In any of the two worlds (ideal and real), an external entity denoted *environment* ($\mathcal{Z}$) interacts with the regular parties. It activates the execution of the protocol by defining possibly-correlated inputs of the parties, observing their final outputs and possibly interacting with the adversary during the protocol execution. However, $\mathcal{Z}$ does not have direct access to the ideal functionality $\mathcal{F}$ in the ideal world. In the ideal/real simulation paradigm, security (i.e., simulatability) is expressed as the condition that $\mathcal{Z}$ is not able to distinguish whether (i) it is interacting with parties and the real adversary $\mathcal{A}$ in a real world protocol execution, or (ii) interacting with the parties and the simulator $\mathcal{S}$ in the ideal world. This is proven by showing that there is a simulator $\mathcal{S}$ that, when given black-box access to $\mathcal{A}$, is able to induce in the ideal world an output distribution indistinguishable from the distribution of a real world execution with $\mathcal{A}$. $\mathcal{S}$ achieves this by performing a controlled internal simulation of the real world protocol execution, to learn how the real adversary would behave.

Illustration 2.1 depicts the relation between the ideal world, where an internal simulation takes place, and the real world, for the case of corruption of $\widehat{\mathrm{P}}_{\mathrm{A}}$. Any message sent from $\mathcal{Z}$ to $\mathcal{S}$ in the ideal world is relayed to $\mathcal{A}$ in the simulation. Correspondingly, any message sent from $\mathcal{A}$ to $\mathcal{Z}$ (which in the simulation is impersonated by $\mathcal{S}$) is relayed by $\mathcal{S}$ to $\mathcal{Z}$. In the simulation, $\mathcal{S}$ also impersonates the non-corrupted parties, such that $\mathcal{A}$ *believes* to be in the real world, i.e., not able to distinguish the simulated execution from a real (not simulated) one. The ability of $\mathcal{S}$ to setup the simulation gives $\mathcal{S}$ some extra power in comparison with a

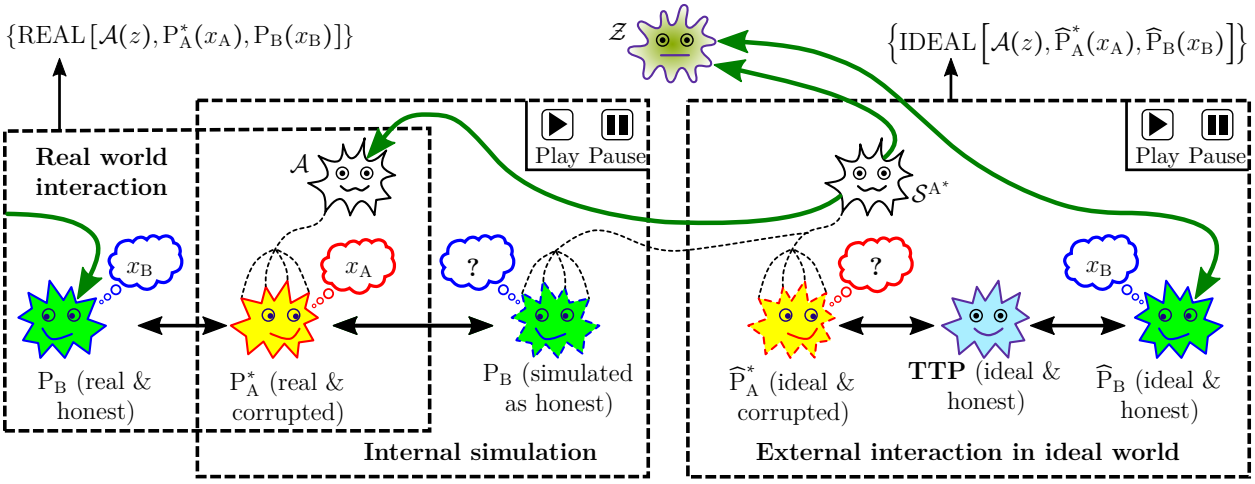


Illustration 2.1: Ideal/real simulation example with statically corrupted $\mathbf{P_A}$. In the ideal world (right side), the simulator is denoted as $\mathcal{S}$. It may also be represented as $\mathcal{S}^{\mathrm{A}^*}$ to denote that it has black-box access to the real-world adversary $\mathcal{A}$. In the ideal world, any messages sent from the environment $\mathcal{Z}$ to $\mathcal{S}$ are relayed to the black-box adversary $\mathcal{A}$ of the real world (left side) in an internal simulated execution. If $\mathcal{A}$ corrupts P^*_A in the simulated execution, then $\mathcal{S}$ corrupts the ideal $\widehat{\mathrm{P}}^*_\mathrm{A}$. In the ideal world, the ideal-and-honest $\mathrm{P_B}$ has the ability to privately communicate with $\mathcal{Z}$. The communications between $\mathcal{Z}$ and $\widehat{\mathrm{P}}_\mathrm{A}$ are intersected and managed by the adversary. $\mathcal{S}$ does not have direct access to the input (x_B) of the ideal $\widehat{\mathrm{P}}_\mathrm{B}$, so it simulates an arbitrary (yet hidden) input for the impersonated $\mathrm{P_B}$ (simulated as honest). In the simulated execution, $\mathcal{S}$ impersonating $\mathrm{P_B}$ extracts the input of the corrupted P^*_A, and then uses the extracted input as the input of the ideal-and-corrupted $\widehat{\mathrm{P}}^*_\mathrm{A}$ in the ideal execution. $\mathcal{S}^{\mathrm{A}^*}$ is able to control the timing of message delivery, but in the simulatability setting in this dissertation is not allowed to rewind the adversary. The goal of $\mathcal{S}$ is to ensure that the probability distribution of the view (IDEAL) of the environment in the ideal world is indistinguishable from the distribution of the view (REAL) in the real world, for any pair of inputs (and for any interaction between $\mathcal{Z}$ and the adversary during the execution).

regular party, which may be essential to enable an indistinguishable execution.

In the case of S2PC the simulator impersonating an honest party in a simulated execution needs to be able to extract the circuit input of the malicious party in the simulated execution (x_A in the Illustration), so that it can send it to the ideal functionality $\mathcal{F}_{\mathrm{S2PC}}$ in the ideal world. Then, upon receiving a circuit output from $\mathcal{F}_{\mathrm{S2PC}}$ in the ideal world, the simulator plays the remainder of the simulated execution in a way that induces the malicious party to receive said output. Based on what $\mathcal{S}$ learns about how $\mathcal{A}$ would behave in the real world (including if it aborts), $\mathcal{S}$ decides how to behave in the ideal world, in order to induce in the ideal world a *global output* (i.e., the joint output of all the parties) with a probabilistic distribution indistinguishable (by $\mathcal{Z}$) from the one that $\mathcal{A}$ would induce in the real world.

Achieving simulatability is useful for the modular design of larger protocols, as it guarantees

security under some type of *composition* operation, e.g., non-concurrent sequential composition [Can00] (a.k.a. the stand-alone setting) or *universal composability* [Can01], depending on the type of achievable simulation — *with-rewinding* or *one-pass*, respectively. In a standalone setting, $\mathcal{Z}$ interacts with the parties and the adversary only in the beginning and in the end of the protocol, by means of delivering their input and receiving their output. In the universal composability framework, $\mathcal{Z}$ is also able to interact with the adversary throughout the protocol execution. Further details about composability are left outside of the scope of this dissertation, except for noticing that it allows a simpler analysis of the security of larger protocols. Essentially, proving the security of a large protocol made of several modular components can be proven by: (i) proving the security (simulatability) of the modular components; and then (ii) proving the security of the larger protocol when instead of its components it uses respective ideal functionalities.

A note on *rewinding*. A traditional simulation technique in the standalone setting involves *rewinding*, denoting the ability of the simulator $\mathcal{S}$ to rewind the state of the real adversary $\mathcal{A}$ in a simulated execution, without the adversary "understanding" that it has been rewound. The notion is intuitive when considering adversaries that can be virtualized as a computer algorithm execution, accessible in a black-box manner, with recoverable past states (e.g., by taking snapshots of the state of the party across state transitions), and interacting with the external environment $\mathcal{Z}$ only at the beginning and end of a protocol execution. Upon rewinding, $\mathcal{S}$ can change the execution path, and thus learn how $\mathcal{A}$ would react to different contexts. In contrast, in settings (such as in the UC framework) where the environment $\mathcal{Z}$ may interact with the adversary at arbitrary moments of the protocol execution (e.g., during concurrent protocol executions), rewinding of $\mathcal{A}$ would be detectable by $\mathcal{Z}$ and is thus not allowed in a simulation. In this case, *one-pass simulatability* is required and the power of $\mathcal{S}$ must come from elsewhere. In the UC framework, the extra power is often obtained in the scope of a *hybrid world*, where some primitives or sub-protocols are replaced by respective ideal (sub-)functionalities, which $\mathcal{S}$ is able to impersonate to gain the needed extra power. A usual choice is the "common reference string" (CRS) ideal functionality, allowing each party to access the same string assuming that it is randomly decided from some distribution and associated to a particular S2PC execution. (The notion of CRS was used prior to the UC framework, e.g., to allow simulatability of non-interactive zero-knowledge proofs [BFM88], where the non-interactive nature of the protocol also makes rewinding useless.

This dissertation is focused on one-pass simulatability, i.e., simulatability without rewinding. Still, since the definition of protocols and the proof of security identifies the crucial aspects where $\mathcal{S}$ needs special power (e.g., extractability of committed values and/or equivocability of opened values, respectively), it is easy to simplify some sub-protocols in settings where it may be possible to assume that rewinding is possible and undetected by $\mathcal{Z}$.

Syntactical considerations. There are many technical subtleties associated with the universal composability framework, which was devised to tackle a wide generality of settings for cryptographic protocols, e.g., concurrent executions, unbounded number of parties, adaptive corruptions, synchronous vs. asynchronous models. This complexity is recognized in the literature, and over time several refinements have been made to the model to deal with respective technicalities. For more restricted settings (e.g., with a fixed number of parties and with known identities, and considering static corruptions), several simplifications are possible without compromising compositional guarantees [CCL15, Wik16]. Such simplifications are used implicitly throughout this dissertation, which is focused on the two party setting and a static corruption model. For example, all communication is assumed to be ideally authenticated, without explicitly mentioning every time that this would occur in a setting with ideally authenticated channels (which could otherwise be modeled in a hybrid setting with access to a respective ideal functionality).

Messages sent between each party and the ideal functionality are composed of a public header and an optional private content (e.g., notation as in [CLOS02]). The public header is composed of: a *message-type identifier*, e.g., a text-string suggestive of the role of the message in the protocol; a session identifier *sid*, so that in concurrent executions each party knows to which session the message refers to; a sub-session identifier (e.g., *cid* for commitment identifier), so that in a modular construction an ideal functionality can be called several times (e.g., for multiple commitments) within the same session, instead of requiring a different copy of the functionality for each call (e.g., see [CR03]); identifiers of the parties associated with the message (e.g., sender and receiver), so that the message can be properly forwarded and processed; additional contextual information that does not need to be private, e.g., the length of a bit-string being committed, or the Boolean circuit being evaluated by a S2PC. The private content can include for example the actual private value being committed or the private circuit input of a S2PC.

The communication model allows the adversary to read the public content of every message

exchanged between the regular parties and the ideal functionality, and to delay or block messages, but does neither allow changing message content nor reading private content. For example, [CCL15] considers a communication model with a star configuration, with a router at the center that informs the adversary of every message, and allowing a decision to block or delay, but not revealing the private content nor allowing content modifications.

2.2.2 Types of trusted setup

Simulatable S2PC and commitment schemes cannot be realized in the plain model [CF01] (i.e., without assuming some kind of trusted setup) when simulation does not allow rewinding, namely in the universal composability framework. However, by making available an extra ideal functionality that provides a trusted setup it is possible to enable simulatability.

Local trusted-setups. A popular trusted setup is a local "common reference string" (CRS), made available by a respective ideal functionality $\mathcal{F}_{\text{CRS}}$ that, within the *local* context of a protocol execution, gives both parties access to the same reference string, obtained randomly from some probabilistic distribution. The parties (e.g., P_A and P_B) involved in a computation are able to call the ideal functionality $\mathcal{F}_{\text{CRS}}$ by sending her the context of the execution, e.g., the session (and subsession) identifiers *sid* and the identity of the intervening parties, and then receiving back the CRS, e.g., defining the public parameters of a commitment scheme.

If the trusted setup functionality $\mathcal{F}_{\text{CRS}}$ is indeed only available within the *local* scope of an execution session (namely not accessible by the external environment $\mathcal{Z}$), then it is susceptible to impersonation by the simulator $\mathcal{S}$ in a simulation. This impersonation allows $\mathcal{S}$, in a simulated execution, to provide as CRS a random looking string to the real world adversary ($\mathcal{A}$), but with the string actually being selected in a special manner, e.g., with an associated secret trapdoor value known only to $\mathcal{S}$. For example, if the parties then use the CRS to define the public parameters of a commitment scheme used within the protocol execution, $\mathcal{S}$ may be able to use the extra knowledge (the trapdoor) to extract committed values or equivocate their opening, whereas a regular party would not be able to do so.

As another example, a local "public key infrastructure" (PKI) ideal functionality $\mathcal{F}_{\text{PKI}}$ could provide to each party a respective secret parameter and make available to all parties the public parameters of all parties. In this case, a simulator $\mathcal{S}$ impersonating the local PKI

would be able to decide the PKI parameters such that it would know the secret parameters of every party. (The protocol described in this dissertation does not make use of a local PKI, but may take advantage of a global PKI — considered ahead).

Even if a local setup might be somewhat impractical to instantiate, it allows defining a protocol with security reducible to the trustworthiness of said setup, thus allowing future considerations on how to replace it by another setup. For example, a protocol whose simulatability depends on a trusted local CRS can still be securely used, within a larger protocol, after replacing the local CRS ideal functionality by a respective simulatable coin-flipping protocol that jointly decides the CRS.

Global trusted-setups. In contrast to local trusted setups, a global trusted setup may correspond to a more readily available setting, e.g., a public-key infrastructure that is pre-existent and remains constant across executions of different protocols. Since the parameters of a global setup (e.g., global CRS or global PKI) are explicitly accessible by the environment, the simulator $\mathcal{S}$ needs to use said parameters in a simulation, lest it would directly induce a distinguishable outcome (i.e., break simulatability). This means that the simulator does not have in advance the knowledge of useful trapdoors associated with the global setup (i.e., beyond what an honest party would), and thus the setup does not provide a needed super power to the simulator [CDPW07]. In summary, simply replacing a local setup by a corresponding global setup could lead the simulator to lose its ability to properly simulate a protocol execution. As a corresponding example, by making public the value of a local CRS, a zero-knowledge proof (i.e., its transcript) based on said CRS could become transferable to anyone that would "believe" the CRS was properly selected without interference from the simulator [Pas03]. In fact, a global-CRS setup alone is not sufficient to realize the S2PC functionality [CDPW07].

Even when a global setup does not endow $\mathcal{S}$ with the needed extra power for a simulation, a protocol execution (possibly aided by another setup) may still allow the simulator to access some secret information made available to the parties. For example, if upon receiving a trapdoor from a global PKI the respective party gives a zero-knowledge proof of knowledge (ZKPoK) of said secret trapdoor, then it follows (by definition of ZKPoK) that the simulator is able to extract said trapdoor. On its turn, the extraction from the ZKPoK must necessarily be empowered by some other local setup (if rewinding is not allowed), e.g., a local CRS setup that during the ZKPoK gives an edge to the simulator.

Considered trusted-setups. As mentioned, there are pros-and-cons of an entirely global setup, e.g., practicality of setup instantiation vs. loss of non-transferability. Considering this contrast, this dissertation proceeds with a design decision of specifying its main protocol (S2PC-with-Coms) based on a setup composed of a global part and a local part. Specifically, the global setup is used to define the parameters of commitment schemes (also so that they are externally meaningful, e.g., for linked executions in multi-party settings), whereas a local setup is considered to allow the simulatability of needed sub-protocols, e.g., non-interactive (NI) ZKPs, NI ZKPoKs and coin-flipping.

The following two main types of setup will be considered:

- **Global PKI ($\mathcal{G}_{\mathbf{PKI}}$) with Local CRS ($\mathcal{F}_{\mathbf{CRS}}$).** The parties have access to a global PKI (GPKI) and a local CRS. A global PKI associates with each party identity P_p a pair composed of a public parameter and a secret parameter. For an IFC instantiation, the public parameter may be a Blum integer N_p and the secret parameter — the trapdoor t_p — be the respective integer factorization. For a DLC instantiation, the public parameter associated with each party P_p may be a pair of generators (g_0, g_p), for simplicity assuming that the first (g_0) is equal for everyone, and with the secret parameter being the discrete log of the second generator (g_p) base the first generator. As part of a protocol execution in the real world, each party is able able to access the ideal global-PKI functionality $\mathcal{G}_{\mathrm{PKI}}$ to retrieve her own public-secret pair and obtain the public parameters of other parties. A different PKI setup could keep the trapdoors hidden even from each respective party, but for concreteness it is hereafter assumed that each party learns a respective trapdoor — for example, in the IFC case this will enable more efficient ZKPoKs (e.g., a single ZKPoK of trapdoor instead of one ZKPoK of opening of each BitCom). In order to empower the simulator $\mathcal{S}$, an ideal local-CRS functionality $\mathcal{F}_{\mathrm{CRS}}$ is also made available, which $\mathcal{S}$ can impersonate when in the ideal world creating a simulation of the real world.
- **Global CRS ($\mathcal{G}_{\mathbf{CRS}}$) with Local CRS ($\mathcal{F}_{\mathbf{CRS}}$).** The parties have access to a global CRS (GCRS) and a local CRS. The global-CRS functionality ($\mathcal{G}_{\mathrm{CRS}}$) provides a common reference string available to everyone, useful to determine the parameters of a commitment scheme that every party uses when outputting commitments to $\mathcal{Z}$. A global CRS could also be used to define different parameters for each party, e.g., based on the CRS and the (independent) identity of the parties, but for concreteness it is hereafter assumed that whenever there is a global trusted CRS the (outer) commitment schemes of both parties are the same. For an IFC instantiation, the public parameter may be a Blum integer of

unknown factorization. Since it is an open problem how to verify correctness of a Blum integer without knowing its factorization (i.e., or without a non-interactive transferable proof generated by someone that would have known the factorization), the correctness of the Blum integer depends on the trustworthiness of the trusted setup functionality. For a DLC instantiation the public parameter may for example be a pair of generators with unknown discrete log vis-a-vis each other. The local-CRS defines other parameters useful to empower the simulator in a simulation.

Alternatives to a global setup. The global setup provides an intuitive characterization that enables avoiding concerns about maliciously selected commitment scheme parameters. Still, in the main (S2PC-with-Coms) protocol defined later in the dissertation the environment directly provides to the parties the commitment scheme parameters. Such description can be perceived as a sub-protocol of a larger protocol where the parties would fetch the parameters from the trusted setup and would then (in the role of environment) provide those parameters to the sub-protocol. The advantage of such description is that it enables more generalized cases, e.g., where the parameters can also be defined by different secure protocols at a higher level. As a possible disadvantage, it assumes that the parameters of both schemes are immediately known to the initiating party, whereas in practice a protocol with a minimal number of rounds can be initiated without knowledge of the non-initiating party.

The case of the environment providing malicious parameters can also be considered, and in some cases be mitigated. For example, in respect to consistency of parameters across the two parties, the ideal functionality and the parties can explicitly confirm that they agree with the parameters of each other, respectively in the ideal and in the real world. In respect to correctness, in a GPKI setting with known trapdoors the parties can verify correctness of their own parameters and prove it to the other party, whereas in the ideal world they would send the trapdoor to the ideal functionality who could also verify correctness. In a DLC-based global-CRS it may be possible (depending on the group structure and representation) for each party and the ideal functionality to directly verify that the parameters are correct. Conversely, in case of an IFC-based global-CRS containing a Blum integer, a malicious $\mathcal{Z}$ could instead potentially provide a non-Blum integer (computationally) indistinguishable from a Blum integer. This could lead the parties to compute insecure commitments. In all cases, $\mathcal{Z}$ may possibly know the trapdoor of the used commitment schemes, e.g., if the parameters were based on a CRS decided by $\mathcal{Z}$.

Another alternative to a DLC-based global CRS setup (assuming non-interactive verifiability of generators) is to let concrete public parameters be defined as *nothing-up-my-sleeves* parameters [BCC+15] inherent to the protocol specification, i.e., hard-coded in the specification of the real protocol and the ideal functionality, e.g., based on Curve25519. The security of the protocol (for each security parameter value) would thus be reducible to the infeasibility of finding a concrete discrete-log. Yet another alternative is mentioned in Remark 2.8, in the context of a concrete application. The above-mentioned alternatives are not further analyzed in this dissertation, and instead the global trusted setup is considered. An interesting problem discussed by Nielsen and Strefler may occur with a pathological correlation between the parameters of a global setup and the definition of a protocol [NS14]. This is ignored in this dissertation, by letting the protocol definition be independent of the public parameter values.

2.2.3 Ideal commitments

A commitment scheme is a protocol with two parties, denoted *sender* (or committer) and *receiver*, and two phases, denoted *commit* and *open*. In the commit phase, the sender becomes *bound* to a value that is *hidden* from the receiver — correspondingly, the scheme is said to have *binding* and *hiding* properties. In the open phase, the sender reveals the committed value, in a convincing way, to the receiver. Real commitment schemes can be instantiated as concrete protocols (e.g., [Blu81, Nao91]), with the hiding and binding properties explicitly defined in respect to the capabilities of an adversary. In contrast, an idealized definition of a commitment scheme allows a greater level of abstraction, from which the hiding and binding properties can be derived, and helping to conceptualize other properties.

2.2.3.1 Ideal functionality

Ideal simple multi-Coms ($\mathcal{F}_{\text{MCom}}$). Ideal functionalities for commitments have been defined in prior work, e.g., see [CF01, Fig. 3] and [CLOS02, Fig. 4] for multiple BitComs and [DN02, §4.2] for multiple commitments with a multi-valued domain with more than two elements. Essentially, once the ideal functionality $\mathcal{F}_{\text{MCom}}$ receives a request to commit a value, it sends a receipt to the receiver, without disclosing the committed value. Then, when receiving an open request, the ideal functionality reveals the value that had previously been committed. All messages within an execution contain a message identifier (e.g., `commit`, `receipt`, or `open`), and the execution context composed of the session and sub-session

identifiers and the identity of the sender and receiver. Some messages also contain a private component (e.g., the actual value being committed). The sub-session identifiers *cid* allow distinguishing multiple commitment executions via the same ideal functionality initialized with a certain session identifier *sid*. Duplication of commitments via replay attack is assumed infeasible by means of an implicit mechanism that ensures uniqueness of identifiers, e.g., based on time and/or cache and/or stateful counters, but the details of such mechanism is left outside the scope of the protocol definition and is assumed to be ensured at a higher level.

Herein, the ideal simple multi-Coms functionality $\mathcal{F}_{\mathbf{MCom}}$ is defined as a special case of the generalized multi-Coms functionality $\mathcal{F}_{\text{GMCom}}$ defined ahead, when requiring the opened value to be the same as the respective committed value, i.e., after applying an identity function. The functionality explicitly handles the possibility of early abort, and also allows specification of a family of commitable domains, as in the generalized case.

Ideal generalized multi-Coms ($\mathcal{F}_{\mathbf{GMCom}}$). In this dissertation it is useful to consider a generalized version of commitments, allowing the sender to ask for the opening of the result of applying a function f to the committed value, where the function can be selected from an implicitly specified family of functions. This is usable in a corresponding generalized version of coin-flipping into-a-well, useful for S2PC-with-Coms, e.g., where one party commits to (the knowledge of) a vector of square-roots but in the open phase only reveals a respective vector of modular squares, or where it commits to (the knowledge of) an exponent but in the open phase only reveals a respective modular power. Even if the function f to be applied is publicly known at commit phase, a simulator corrupting the sender is still able to extract the pre-image of the value that may later be opened to the receiver. Thus, from a simulatability standpoint, using the generalized commitment functionality $\mathcal{F}_{\text{GMCom}}$ to commit to a pre-image and then open the image is in general different from committing directly the f-image using a simple commitment functionality $\mathcal{F}_{\text{MCom}}$ (that might not allow extraction of the pre-image). In the ideal world, the parties do not need to execute any kind of computation, as in essence the *commit* and *open* phases are intermediated by the ideal functionality $\mathcal{F}_{\text{MCom}}$, fully trusted to store a value transmitted by the sender (P_S), and later, when allowed by P_S, to reveal a function thereof to the receiver (P_R).

The ideal general multi-Coms functionality $\mathcal{F}_{\text{GMCom}}$ in Figure 2.1 is also parametrizable by a specification $\mathbb{G}$ of a family of committable domains $\mathbb{G}_\ell$. This allows committing each time to elements from different sets (e.g., bit-strings of specifiable length ℓ, or residues from

Ideal functionality $\mathcal{F}_{\text{GMCom}}$

Implicit parameters. Let $\mathbb{S}$ denote a specification of a family $\equiv \left\{\mathbb{S}_\ell : \ell \in \mathbb{N}\right\}$ of finite sets $\mathbb{S}_\ell$, with each such set being indexed by an integer ℓ, and containing only elements representable in size polynomial in ℓ, and with membership in each set being efficiently decidable without auxiliary information. (Each set $\mathbb{S}_\ell$ is a possible domain of committable values.) Let $\mathbb{F}$ denote a specification of a family $\{F_{\ell,f} : (\ell, f) \in \mathbb{N}^2\}$ of functions $F_{\ell,f}$, with each such function being indexed by a pair (ℓ, f) of integers, being efficiently computable (i.e., in time polynomial in the size of its indices ℓ and f), and having domain containing $\mathbb{S}_\ell$. Let $\kappa \equiv 1^\kappa$ denote a computational security parameter.

Procedure. $\mathcal{F}_{\text{GMCom}}$ activated with session identifier *sid*, and (possibly implicitly) parametrized by $\mathbb{S}$, $\mathbb{F}$ and κ, proceeds as follows when running with parties P_1, ..., P_n and adversary $\mathcal{S}$:

- **Phase 1 — Commit phase.** Upon receiving a message (`commit`, *ctx*, ℓ, m) from $\mathrm{P_S}$ (the sender): if $ctx \equiv (sid, cid, \mathrm{P_S}, \mathrm{P_R})$, where *sid* is the session identifier with which $\mathcal{F}_{\text{S2PC}}$ was activated, and $\mathrm{P_S}, \mathrm{P_R} \in \{P_1, ..., P_n\}$, if $\ell \in poly(\kappa)$, and if $m \in \mathbb{S}_\ell$, and if a tuple (`commit`, *ctx*, ...) has *not* yet been stored, then store the tuple (`commit`, *ctx*, ℓ, m) and send the message (`receipt`, *ctx*, ℓ) to $\mathrm{P_R}$ (the receiver) and $\mathcal{S}$; otherwise ignore it. (Note: an alternative definition could allow more than one receiving party.)
- **Phase 2 — Open phase.** Upon receiving a message (`open-ask`, *ctx*, f) from $\mathrm{P_S}$: if for some pair (ℓ, m) a tuple (`commit`, *ctx*, ℓ, m) has been stored, if $\ell \in poly(\kappa)$ and if a tuple (`ignore`, *ctx*) has *not* been stored, then store the tuple (`ignore`, *ctx*) and send the message (`open-send`, *ctx*, f, $F_{\ell,f}(m)$) to $\mathrm{P_R}$ and $\mathcal{S}$; otherwise ignore it. (Note: an alternative definition could allow further open requests, by not storing the tuple (`ignore`, *ctx*).)
- **Abort.** Upon receiving a message (`abort`, *ctx*) from $\mathrm{P_S}$: if a tuple (`ignore`, *ctx*) has not been stored, then store the tuple and send the message (`abort`, *ctx*) to $\mathrm{P_R}$ and $\mathcal{S}$.

Figure 2.1: **Ideal functionality — generalized multi-Coms.** The family $\mathbb{S}$ of committable domains may for example be the family of sets of bit-strings of length equal to the set index ℓ. The family may, if so intended, be composed of some empty sets, meaning that a `commit` call with any such index would result in the message being ignored, because any value m specified for commitment would not be contained in the respective (empty) set. In other words, the set of meaningfull indices can be a subset of the natural integers.

a multiplicative set defined of integers modulo some composite integer).

The functionality considered herein also takes in explicit consideration the case of abort by the sender. For each execution context the sender is given the possibility to abort before completing the open phase, but not after the open phase. In other words, this allows the sender to ask the ideal functionality to send an explicit abort message to the receiver. In practice, this makes it more natural to allow the receiver in a real protocol to abort in case of detecting malicious behavior by a sender, instead of ignoring the malicious behavior and continuing to accept attempts associated with the same execution context.

Simplifications. If all indices have associated the same non-empty committable domain, then the index can be omitted from the `commit` call and instead the single committable domain becomes an implicit parameter of the activation of the functionality. A more strict definition could require that P_S would immediately define the function f in the commit phase. If all functions from the function family F are equal, then the parameter can also be avoided. If all functions are equal to the identity function, then the ideal functionality is instead denoted as simple multi-Coms $\mathcal{F}_{\text{MCom}}$ (in contrast to generalized multi-Coms). The generalized version shall be considered only when explicitly mentioned; otherwise the simple version is considered, and the function index is omitted from the notation.

Remark 2.2 (Different nuances of generalized commitments). Other generalized commitment functionalities have been defined in the literature. For example, [CLOS02, Fig. 8] defines a "commit and prove" functionality, parametrized by a non-deterministic polynomial relation, replacing the typical open phase by a "prove phase" that allows the committer to ask for the opening of a specific value satisfying said relation in respect to the list of elements committed thus far. This could be used to request opening of a function of a committed value, e.g., by specifying the relation as being true whenever the value requested for opening is a tuple containing the position of the committed element, a specification of the function, and the f-image of the respective committed value. As a technical difference, in the commit phase the sender has to retain state of the committed value (or of something that allows determining its f-image). Complementary, [DN02, §4.2] also specifies a "prove phase" (besides a regular open phase), allowing the proof of an adaptively chosen relation about previously committed values. In practice this could also be used to open a function f of a committed value, e.g., by committing the f-image of a previously committed value, then requesting the opening of that f-image and then requesting confirmation of the respective relation between the two committed values (the pre-image and the image). The generalized commitment functionality $\mathcal{F}_{\text{GMCom}}$ herein does not consider (but could be generalized to consider) the case of an opening related to several committed values, which could be useful for reactive functionalities.

Other variations. A different functionality could require the function f to be specified in the commit phase. More generalized cases could include P_2 also specifying a function, and P_1 specifying one function in the commit phase and another in the open phase, and the opened

value be obtained as a composition of these functions applied to the committed value.

2.2.3.2 Properties

Hiding and binding. The *hiding* and *binding* properties derive directly from the defined ideal functionality. Since in the commit phase the receipt sent by $\mathcal{F}_{\text{GMCom}}$ only reveals the message domain (e.g., length of a bit-string), the receiver does not learn anything else about the message. Since in the open phase the value sent by $\mathcal{F}_{\text{GMCom}}$ is (a function of) the value that had been stored in memory (along with the function specification), the sender has no ability to change the value, i.e., it is bound to a function of the pre-image that it previously committed. In the real world these properties are ensured via concrete primitives, with at least one of the properties depending on computational assumptions. The complementary property might be guaranteed unconditionally, but it may also be computational.

Interactivity. A commitment scheme is called *interactive* if any of the two phases requires at least one message from the receiver (P_R) to the sender (P_S), and *non-interactive* if the communication in each phase consists only on sending a message from the sender to the receiver. This dissertation is focused on non-interactive commitments, as a way to reduce the number of communication rounds required by the main protocol — S2PC-with-Coms.

Extractability (Ext) and equivocability (Equiv). While the hiding and binding properties directly reflect inabilities of the parties, there are more subtle properties related to the simulation paradigm of security. A commitment scheme is called *equivocable* (Equiv) if the simulator in the role of sender is able to open any commitment into any value in the domain of allowed committed values [Bea96a]. A commitment scheme is called *extractable* (Ext) if the simulator in the role of receiver is able to extract the value that has been committed by an honest *sender* [SDCP00]. Naturally, a real party in a commitment scheme must not be able to extract or equivocate, as those capabilities would respectively contradict hiding and binding. However, in a simulation the simulator is endowed with extra power in comparison with a regular party, e.g., knowledge of a trapdoor.

Not all real commitment schemes are Ext or Equiv, but those properties are required of a simulatable scheme, i.e., one that emulates the ideal commitment functionality. For example, the properties may be needed when simulating an hybrid protocol with access to

an ideal commitment. In the simulation, $\mathcal{S}$ impersonates the ideal commitment and takes advantage of the trust that the other parties place in it. In particular: *extractability* in the commit phase derives from $\mathrm{P}_{\mathrm{S}}^{*}$ sending the message in clear to $\mathcal{F}_{\mathrm{MCom}}$ or $\mathcal{F}_{\mathrm{GMCom}}$, which means that $\mathcal{S}$ playing against a possibly malicious P_{S} is able to directly receive the committed value, i.e., *extract* it; *equivocability* in the open phase derives from the acceptance by P_{R} of any (well formed) message received from $\mathcal{F}_{\mathrm{MCom}}$ or $\mathcal{F}_{\mathrm{GMCom}}$, which means that $\mathcal{S}$ playing against a possibly malicious $\mathrm{P}_{\mathrm{R}}^{*}$ is able to lie about the committed value (i.e., equivocate it). Consequently, to emulate an ideal commitment scheme, e.g., when used as an auxiliary sub-routine in a larger protocol, a real commitment scheme must also be Ext-and-Equiv.

Remark 2.3 (Nuances of equivocability in case of generalized commitments). In practice, in the real world it is often left implicit that P_{R} makes the adequate syntactical verifications of the received value. For the generalized functionality $\mathcal{F}_{\mathrm{GMCom}}$ the notion of equivocability is by default meant in respect to the allowed set of images of the domain of committable values. More fine-grained nuances could be defined in case the membership in the f-image of the domain cannot be decided efficiently, i.e., if even a disallowed value could not be detected as false by the receiver.

Remark 2.4 (Ext-and-Equiv implies binding-and-hiding). It is interesting to notice that *equivocability* in the *open* phase implies that the *commit* phase is *hiding* (or otherwise the receiver could reject an equivocated opened value) and the opening phase is *revealing*; correspondingly, *extractability* of the *commit* phase implies that the *commitment* is binding (or otherwise the value to extract would be ill-defined). In other words, the combination of Ext-and-Equiv implies that the scheme is indeed a commitment scheme (hiding and binding). This means that the combination of extractability and equivocability can be seen as a generalization of the combination of hiding and binding.

Remark 2.5 (A language note on "equivocal" vs. "equivocable"). It is arguable whether "equivocal" or "equivocable" is the best word to describe the property at hand. Some authors prefer "equivocal" (e.g., [CLOS02, footnote 17]), possibly as a more well-estalished dictionary word, whereas others use the term "equivocable" (e.g., [Bea96c]). Assuming that "equivocal" denotes an ambiguity about possible values, this dissertation uses the term "equivocable" to denote the property that a simulator is able to act upon an underlying

equivocality, to induce the meaning that it intends. Hence, without danger of equivocation, commitments are herein called as "equivocable," whenever the property applies.

Non-malleability. Complementary to the mentioned properties, another relevant aspect of commitment schemes relates to the independence between several commitments. In the ideal world, the `commit` request requires explicit specification of the value being committed by the committer. This prevents a situation where one party would be able to commit a value somehow related to the value committed by another party, while not knowing which value it is. While this property can be derived from the ideal functionality, it is not directly implied by the hiding property of a real commitment scheme. Absent an ideal world, non-malleability could be defined based on a game where an adversary tries committing to a value somehow related to the value committed by another commitment, trying to have a noticeable advantage in comparison to a case where the initial commitment was not available.

The concept of non-malleability was studied by Dolev et al. [DDN00]. They defined the concept more broadly, associated also with zero-knowledge proofs of knowledge (ZKPoK) and public-key encryption. In fact, non-malleable ZKPoKs can play an important role in ensuring non-malleability of real commitment schemes, for example allowing a direct augmentation of a regular commitment scheme, enabling each party to prove possession of the information that enables a successful open phase. Non-malleable commitments can be achieved with both interactive and non-interactive protocols [DCIO98, DCKOS01].

Non-malleability can be considered in strong and weak nuances [FF09]. In a weak version, non-malleability *with respect to opening* implies that if during or after the commitment generated by an honest party an adversary is able to generate a commitment somehow related (i.e., with an added advantage) to the value committed by the honest party, then the second commitment cannot be opened, even during or after the opening of the first commitment. For example, this property is not satisfied by a commitment scheme where a receiver would accept from a second sender a commitment simply copied from a commitment previously sent by another first sender. In a strong version, non-malleability *with respect to commitment* directly implies that an adversary is not even able to produce a commitment to a related value (i.e., regardless of begin able or unable to open it). This applies even for the equality relation, where the commitment of one party should not allow another party to commit to the same value, i.e., before seeing the opening of the first commitment. As already mentioned, the strong notion can be obtained based on a non-malleable ZKPoK of knowledge. The weaker

version can be achieved without the ZKPoK in the commit phase, and instead augmenting the open phase — instead of revealing the randomness used to produce a commitment, P_S can reveal the committed value and then provide a non-malleable zero-knowledge proof (argument) that it was indeed the committed value. The weak notion might be preferred when it is sufficient for a particular application and if it allows a more efficient instantiation.

Interestingly, extraction and equivocation can also be achieved from a regular commitment scheme by respectively augmenting the commit and open phases with a ZKPoK of the committed value and a ZKP that the opened value is correct. Nonetheless, Ext and Equiv remain complementary to non-malleability.

Absence of properties. While a simulatable commitment scheme is simultaneously Ext, Equiv and non-malleable, real commitment schemes may in practice be useful even when having only some but not all of those properties. Correspondingly, the constructions in this dissertation make use of diverse commitment schemes, with varying properties. For example, the construction of a new Ext&Equiv-Com scheme in Chapter 4 is achieved by means of using separate ideal Ext Coms and Equiv Coms for short strings. For example, in the construction of the S2PC-with-Coms protocol in Chapter 3, the commitments outputted by the protocol are extractable, but some other commitments used in the internal protocol (e.g., to commit the input keys of P_A, the garbled circuit constructor) need not be extractable or equivocable.

Another relevant and useful property of some commitment schemes has to do with homomorphic operations (see §2.3.2). While an homomorphism is by definition a case of malleability, it is not incompatible with the intended non-malleability. In fact, non-malleability of a commitment scheme is intended in respect to commitments (or other primitives) computed by other parties and/or in different sessions. In contrast, it may be useful for the same party to be able produce and open a commitment that is related to previous commitments, in a verifiable way, without having to open the original commitment. For example, this is useful in this dissertation to enable efficient ZKPs.

2.2.4 Ideal S2PC (without Coms)

Before defining an ideal functionality of S2PC-with-Coms, it is instructive to analyze the more basic case of an ideal S2PC (without commitments), also serving as a basis to consider possible variations. An ideal functionality $\mathcal{F}_{\text{S2PC}}$ is described in Figure 2.2. Before any

computation, the functionality needs to receive initializing messages from both parties, with a consistent execution context (identifiers and parties), namely defining who should be the first receiver (P_B) and second receiver (P_A), and defining the respective 2-output function to evaluate. The ideal functionality then locally computes the output and sends it first to P_B, then waits for an acceptance message from P_B and only then sends the result to P_A. The functionality also explicitly embeds the ability to handle requests to abort the execution.

Hidden outputs. In the defined ideal S2PC the output value sent to each party is not sent to the simulator $\mathcal{S}$. Thus, $\mathcal{S}$ only learns a private output if it corrupts the respective party. This is in contrast to the case of the ideal commitment functionality $\mathcal{F}_{\text{GMCom}}$, where the value opened to P_2 is also visible to $\mathcal{S}$. It is conceivable an alternative ideal S2PC functionality that in the second output message (`out-2`) would reveal to the simulator some component of the output that would be common across the two parties (in a real protocol the first party to learn it would then reveal in the clear that part of the output, e.g., the common output bits in case the function is a circuit).

Unilateral initiative for a S2PC evaluation. The ideal functionality requires both parties to have initiating messages agreeing in the function to compute and in the remaining consistent parameters (e.g., complementary message identifiers and same sub-session identifier). This is not opposed to an application scenario where the "initiative" to perform a S2PC, and the decision of which function to compute, originates from a single party (hereafter called the initiator), and only then agreed by the second party. This can be implemented with the described functionality, as follows. The initiator (P_A or P_B) proposes the parameters of an execution, including the role of each party as first or second receiver (encoded in the message identifier — `in-1` or `in-2`, respectively). As a result, the other party receives the execution parameters from the ideal functionality and relays them to the environment (i.e., to some upper level routine that interprets the message). If the environment agrees with the execution, then it activates the second party with the needed parameters for the S2PC, which includes the same S2PC public parameters, a complementary message identifier (`in-2` or `in-1`, respectively) complementary to the identifier used by the initiator, and also the private input of the second party, which the party then relays to the ideal functionality. If within a particular context it is clear that the initiation does not need to be concurrent and is always initiated by a party in a particular role, e.g., always P_B (the first receiver of output),

Ideal functionality $\mathcal{F}_{\text{S2PC}}$

Implicit parameters. Let $\mathbb{F}$ denote a specification of a family $\{(F_f, D_f) : f \in \mathbb{N}\}$ of pairs, with each pair being of size polynomial in the size of the integer index f, being reconstructable, in polynomial time, from the family specification F and the index f, and being composed of a specification F_f of a two-output ternary function (i.e., $F_f(\cdot,\cdot,\cdot) \equiv (f_B(\cdot,\cdot,\cdot), f_A(\cdot,\cdot,\cdot))$ with three input parameters and with an output encoded as a pair) computable in polynomial time, and a specification D_f of the respective triplet $(D_{f,1}, D_{f,2}, D_{f,3})$ of domains (the first two are for the private inputs of the parties, the third is for needed randomness) with membership decidable in polynomial time. Let $\kappa \equiv 1^\kappa$ denote a computational security parameter.

Procedure. $\mathcal{F}_{\text{S2PC}}$ activated with session identifier *sid*, parametrized by κ and $\mathbb{F}$, and running with parties P_1, ..., P_n and adversary $\mathcal{S}$, proceeds as follows:

- **Receive inputs.** Upon receiving a message (`in-`i, *ctx*, f, x_p) from P_p: if $ctx \equiv (sid, cid, \text{P}_\text{A}, \text{P}_\text{B})$ has the session identifier *sid* with which $\mathcal{F}_{\text{S2PC}}$ was activated, if $\text{P}_\text{A}, \text{P}_\text{B} \in \{P_1, ..., P_n\}$, if $i \in \{1, 2\}$ and $\text{P}_p = \langle \text{P}_\text{B}, \text{P}_\text{A} \rangle_i$, if $f \in poly(\kappa)$ and $x_p \in D_{f,i}$, and if *neither* a tuple (`in-`i, *ctx*, ...) *nor* a tuple (`ignore`, *ctx*) have been stored, then store the tuple (`in-`i, *ctx*, f, x_p), and send the message (`got-`i, *ctx*, f) to $\text{P}_{\bar{p}} = \langle \text{P}_\text{A}, \text{P}_\text{B} \rangle_i$ and $\mathcal{S}$; otherwise ignore it.
- **Validate pair of inputs and send first output.** Upon storing a pair of initializing tuples ((`in-1`, *ctx*, f, x_B) and (`in-2`, *ctx*, f, x_A)) with complementary message identifiers (`in-1` and `in-2`) and same context *ctx*: if the function index f is not the same in the two messages, then store the tuple (`ignore`, *ctx*) and send (`abort`, *ctx*) to the two parties; otherwise proceed to select an eventually needed randomness r from the respective domain $D_{f,3}$, then compute the indicated pair $(y_\text{A}, y_\text{B}) = F_f(x_\text{A}, x_\text{B}, r)$ of outputs, then store the tuple (`outputs`, *ctx*, (y_A, y_B)) and then send (`out-1`, *ctx*, y_B) to P_B and (`out-1`, *ctx*) to $\mathcal{S}$.
- **Second output.** Upon receiving a message (`OK`, *ctx*) from P_B: if a tuple (`outputs`, *ctx*, ...) has been stored and if a tuple (`ignore`, *ctx*) has *not* been stored, then store the tuple (`ignore`, *ctx*) and send (`out-2`, *ctx*, y_A) to P_A and (`out-2`, *ctx*) to $\mathcal{S}$; otherwise ignore it.
- **Abort case.** Upon receiving a message (`abort`, *ctx*) from P_A or P_B: if a tuple (`in-`i, *ctx*, ...) has been stored and if a tuple (`ignore`, *ctx*) has *not* been stored, then store the tuple (`ignore`, *ctx*) and send the message (`abort`, *ctx*) to the other party (i.e., to P_B or P_A, respectively); otherwise ignore it.

Figure 2.2: **Ideal S2PC.** The choice of P_B and P_A respectively denoting the first and second party stems from a real protocol definition where Alice (P_A) was a garbled circuit constructor and Bob (P_B) was the evaluator that initially learns the output. The choice is of course arbitrary and P_A and P_B could be consistently swapped everywhere.

then the syntax of the second initiating message (from P_A) can be changed to not require repeating the specification of the function.

Family of functions. Considering the garbled circuit approach considered in this dissertation, in practice it is intended that the function family F represents a family of specifications

of Boolean circuits, and respective indices of input and output bits of both parties. The family of functions may also be interpreted as a single function with an extra input parameter (the index f) that is public instead of private.

Variations. The ideal functionality can be adjusted to define different ways of choosing the functions to evaluate. For example, it could be defined that each party could select the function whose output would be privately learned, upon agreement by the other party. As another example, an initiating message could represent a commitment of input x_{B}, indicating a subset of parties to whom to send a respective receipt of commitment, so that each recipient party P_j could then send (one time or several times) to the ideal functionality a complementary input x_j and function f_j, whose output would be sent to the initiating party $\mathrm{P_B}$.

2.2.5 Ideal S2PC-with-Coms

This subsection defines the ideal $\mathcal{F}_{\mathrm{S2PCwC}}$ functionality of S2PC-with-commitments (S2PC-with-Coms), used in an ideal world to intermediate between two parties, $\widehat{\mathrm{P}}_{\mathrm{A}}$ and $\widehat{\mathrm{P}}_{\mathrm{B}}$, the evaluation of a Boolean circuit and the commitment of each circuit input and circuit output of each party. The simple S2PC ideal functionality could in theory be used to generate random commitments, so the new definition of S2PC-with-Coms can be interpreted as a practical specialized recasting of the S2PC definition, when the function specification corresponds to the specification of a Boolean circuit and commitment scheme parameters, and the output of each party includes her respective circuit output as well as the commitments of the input and output of both parties and the randomness of her own commitments. The new definition is useful by explicitly separating the Coms and the Boolean circuit, allowing an easier modularization of concerns. For example, the parameters of the Com schemes may be defined based on a global PKI or CRS, whereas the Boolean circuits may be more inherently dependent on the functional goals of the application.

2.2.5.1 Ideal functionality

The ideal S2PC-with-Coms functionality $\mathcal{F}_{\mathrm{S2PCwC}}$ is specified in Figure 2.3. The design accommodates some flexibility of parameters in order to enable different types of setup (e.g., CRS and PKI) and instantiations (e.g., IFC and DLC) of commitment schemes. In comparison with the ideal S2PC functionality $\mathcal{F}_{\mathrm{S2PC}}$, the activation of the new ideal S2PC-with-Coms

functionality $\mathcal{F}_{\text{S2PCwC}}$ is parametrized by an additional Boolean flag that determines whether or not the Com scheme parameters must be verified for correctness based on an optional trapdoor that each party may provide for her own Com scheme parameters.

Each party directly specifies the public Com schemes of both parties, as a way to facilitate concurrent initialization by any party (but see Remark 2.7). If a valid trapdoor t_p is provided and the protocol is parametrized with the respective Boolean flag *ver* set to `true`, then $\mathcal{F}_{\text{S2PCwC}}$ locally verifies the correctness of the public Com scheme parameters $\mathscr{C}_p$ proposed by the respective party P_p (using an implicit verification procedure $\text{Ver}_{\text{Params}}$), and then informs the other party $\text{P}_{\bar{p}}$ of the verification result (`false` or `true`, via the message identifier `bad-`i or `got-`i), and of the proposed Com scheme parameters and circuit specification. If the verification fails, then $\mathcal{F}_{\text{S2PCwC}}$ further decides to ignore any subsequent messages with the respective execution context.

After storing a pair of inputs with the same context and complementary message identifiers (`in-1`, `in-2`) of initializing messages, if the messages are not consistent in the proposed pair of commitment scheme parameters or the circuits specification, then $\mathcal{F}_{\text{S2PCwC}}$ sends an `abort` message to both parties. If instead the pair of initiating messages is complementary consistent, then $\mathcal{F}_{\text{S2PCwC}}$ locally computes the circuit outputs, and respective commitments and randomnesses, and sends them first to P_B and only then, if allowed by P_B, to P_A.

If at any point during an ideal protocol execution (i.e., before privately outputting) the functionality receives an `abort` message from one party, then it sends a similar `abort` message to the other party and ignores further messages within the context of the execution.

Information available to $\mathcal{S}$. Along an execution, $\mathcal{F}_{\text{S2PCwC}}$ only explicitly sends to $\mathcal{S}$ the public component of each message, which does not include the private circuit input and trapdoor in the `in-`i messages, nor the circuit outputs and randomnesses of commitments in the `out-`i messages. However, an asymmetry is present in respect to the commitments, which are not sent to $\mathcal{S}$ in the `out-1` message, but are sent in the `out-2` message. This design choice is motivated by the actual real protocol described in this dissertation, where the commitments and randomnesses are decided via a two-party coin-flipping, and thus remain hidden (from P_A, and any eventual observer such as $\mathcal{S}$) until P_B sends the last message of the coin-flipping. For an alternative ideal functionality where the commitments would remain hidden from an eavesdropper, a corresponding real protocol would have to use encrypt them.

Ideal functionality $\mathcal{F}_{\text{S2PCwC}}$

Implicit parameters. Let $\mathbb{F}$ denote a specification of a family $\{F_C : C \in \mathbb{N}\}$ of Boolean circuits, each of size polynomial in the integer index C of the element in the family. Let $\kappa \equiv 1^\kappa$ be a computational security parameter. Let *ver* be a Boolean flag (i.e., with value `true` or `false`), denoting whether or not the ideal functionality must check the correctness of the Com scheme parameters based on a trapdoor.

Procedure. $\mathcal{F}_{\text{S2PCwC}}$ activated with session identifier *sid*, and parametrized with κ, $\mathbb{F}$ and *ver*, running with parties P_1, ..., P_n and adversary $\mathcal{S}$, proceeds as follows:

- **Receive inputs.** Upon receiving a message of the form $(\texttt{in-}i, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), (t_p, x_p))$ from P_p: if $ctx \equiv (sid, cid, \text{P}_\text{B}, \text{P}_\text{A})$ has the session identifier *sid* with which $\mathcal{F}_{\text{S2PCwC}}$ was activated, if $\text{P}_\text{B}, \text{P}_\text{A} \in \{P_1, ..., P_n\}$, if $i \in \{1, 2\}$ and $\text{P}_p = \langle \text{P}_\text{B}, \text{P}_\text{A} \rangle_i$, if $C \in poly(\kappa)$, with $F_C : \{0,1\}^{\ell_\text{A}} \times \{0,1\}^{\ell_\text{B}} \to \{0,1\}^{\ell'_\text{A}} \times \{0,1\}^{\ell'_\text{B}}$ specifying the numbers $(\ell_\text{A}, \ell_\text{B}, \ell'_\text{A}, \ell'_\text{B})$ of private input bits of each party and output bits of each party, if $x_p \in \{0,1\}^{\ell_p}$ (i.e., with $\ell_p = \langle \ell_\text{B}, \ell_\text{A} \rangle_i$) and if *neither* a tuple $(\texttt{in-}i, ctx, ...)$ *nor* a tuple $(\texttt{ignore}, ctx)$ have been stored, then proceed to the next bullet; otherwise ignore it.
- **Check isolated Com scheme parameters.** If $\big(\neg ver\big) \vee (ver \wedge \text{Ver}_{\text{Params}}(\mathscr{C}_p, t_p))$, then store the tuple $(\texttt{in-}i, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), x_p)$ and send $(\texttt{got-}i, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C))$ to $\text{P}_{\bar{p}} = \langle \text{P}_\text{A}, \text{P}_\text{B} \rangle_i$ and $\mathcal{S}$; otherwise store the tuple $(\texttt{ignore}, ctx)$ and send $(\texttt{bad-}i, ctx, (\mathscr{C}_\text{A}, \mathscr{C}_\text{B}, C))$ to $\text{P}_{\bar{p}}$.
- **Validate pair of inputs and send first output.** Upon storing a pair of initializing tuples $((\texttt{in-1}, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), x_\text{B})$ and $(\texttt{in-2}, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), x_\text{A}))$ with complementary message identifiers (`in-1`, `in-2`) and same context *ctx*: if the circuit specification C and the pair $(\mathscr{C}_\text{A}, \mathscr{C}_\text{B})$ of Com scheme public-parameters are not the same across the two tuples, then store the tuple (`ignore`, *ctx*) and send (`abort`, *ctx*) to the two parties; otherwise proceed to compute the circuit output $(y_\text{A}, y_\text{B}) = F_C(x_\text{A}, x_\text{B})$, then use the Com schemes to generate randomnesses $rands = (\underline{x}_\text{A}, \underline{x}_\text{B}, \underline{y}_\text{A}, \underline{y}_\text{B})$ and commitments $coms = (\overline{x}_\text{A}, \overline{x}_\text{B}, \overline{y}_\text{A}, \overline{y}_\text{B})$ of the respective input and outputs $((\underline{x}_p, \overline{x}_p) \leftarrow^\$ \mathscr{C}_p[x_p]$ and $(\underline{y}_p, \overline{y}_p) \leftarrow^\$ \mathscr{C}_p[y_p]$, for $p \in \{\text{A}, \text{B}\})$, and store the tuple $(\texttt{outputs}, ctx, rands, coms, (y_\text{A}, y_\text{B}))$ and send $(\texttt{out-1}, ctx, (\overline{x}_\text{A}, (\underline{x}_\text{B}, \overline{x}_\text{B}), \overline{y}_\text{A}, (y_\text{B}, \underline{y}_\text{B}, \overline{y}_\text{B})))$ to P_B and $(\texttt{out-1}, ctx)$ to $\mathcal{S}$.
- **Second output.** Upon receiving a message $(\texttt{OK}, ctx)$ from P_B: if a tuple $(\texttt{outputs}, ctx, ...)$ has been stored and if a tuple $(\texttt{ignore}, ctx)$ has *not* been stored, then store the tuple $(\texttt{ignore}, ctx)$ and send $(\texttt{out-2}, ctx, (\underline{x}_\text{A}, \overline{x}_\text{A}), \overline{x}_\text{B}, (y_\text{A}, \underline{y}_\text{A}, \overline{y}_\text{A}), \overline{y}_\text{B})$ to P_A and $(\texttt{out-2}, ctx, coms)$ to $\mathcal{S}$.
- **Abort case.** Upon receiving a message $(\texttt{abort}, ctx)$ from P_B or P_A: if a tuple $(\texttt{in-i}, ctx, ...)$, with i respectively equal to 1 or 2, has been stored and if a tuple $(\texttt{ignore}, ctx)$ has *not* been stored, then store the tuple $(\texttt{ignore}, ctx)$ and send $(\texttt{abort}, ctx)$ to the other party (i.e., P_B or P_A, respectively); otherwise ignore it.

Figure 2.3: **Ideal S2PC-with-Coms.** For simplicity, F is directly specified as a family of specifications of Boolean circuits, with each such circuit directly specifying for each input parameter (x_A, x_B) a fixed length $(\ell_\text{A}, \ell_\text{B})$ defining the respective domain as the set of bit-strings of said length. A more generalized definition (but not needed here) could use the function family specification described in the ideal S2PC functionality $\mathcal{F}_{\text{S2PC}}$. The protocol flow is described with succinct notation in Appendix (Figure B.4 in §B.2.1)

2.2.5.2 Properties and considerations

Simulatability of a S2PC-with-Coms protocol implies that $\mathcal{S}$ in a simulated execution is able to induce the party corrupted by the black-box real adversary to accept the commitments and randomnesses (proposed by the ideal functionality in the ideal world). This may be achieved

in the real world with a protocol based on generalized two-party coin-flipping, where two parties learn a random commitment but only one learns the respective randomness. This can be made specially efficient when the Com schemes have certain homomorphic properties, such as in the concrete (IFC and DLC based) cases analyzed in this dissertation. The coin-flipping requires that $\mathcal{S}$, while impersonating one party in a simulated execution, is able to extract from the other party the randomness of her outer commitments. This can be achieved via some NIZKPoKs, either of a trapdoor (that subsequently allows extracting randomness, e.g., in IFC based Coms) and/or directly via NIZKPoKs of openings of Coms (e.g., if the Com scheme is defined by a CRS and thus the trapdoor is not known, or for DLC based Coms where the trapdoor is not enough to extract randomness). Since the S2PC-with-Coms protocol does not include opening of outer Coms, the property of equivocability is not meaningful. Yet, if as part of a larger protocol it is useful that the opening of a commitment is equivocable, then the opening must not reveal the respective randomness.

Remark 2.6 (On the use of real commitment schemes). A different ideal S2PC-with-Coms could be defined by combining features from both an ideal functionality for S2PC and an ideal functionality for commitments. However, the definition herein is instead of the form of a S2PC-looking functionality directly computing real commitments. This makes explicit that commitments must be random, which may be relevant for extensions to larger protocols with many parties, where otherwise a non-random commitment could potentially be used as a side-channel to transmit information. While this allows using commitment schemes that are not fully simulatable, in practice a simulatable S2PC-with-Coms protocol in the real world (see Chapter 3) ensures the property of extractability and the property of non-malleability in respect to commitment by requiring a respective (non-malleable) ZKPoK of the opening of those outer commitments. As mentioned, equivocability is not meaningful within the protocol execution, because the protocol does not consider the opening of any commitment.

Remark 2.7 (On knowing the public parameters of the Com scheme of the other party). It is conceivable an application where the party that takes the initiative to perform a S2PC-with-Coms does not know in advance the Com scheme parameters of the other party, and so would not be able to specify it in the initial message. Thus, a conceivable variation of the ideal functionality could allow the initiating party to only specify her own Com scheme. Still, for simplicity of integrated analysis across different types of Com schemes, the ideal

functionality herein assumes that the Com scheme parameters of both parties are already known in advance. For example, this allows a real protocol with minimal number of rounds even if the protocol requires the initiating party to use the Com scheme of the other party, e.g., to sustain a 2-out-of-1 oblivious transfer, as described ahead in §2.5.1. This is applicable in a scenario where a global PKI is in place, from which each party is able to obtain the public parameters of any other party, without initial interaction between parties. Alternatively, it can be assumed that the initiating party of the S2PC-with-Coms learns the Com scheme parameters of the other party via an external protocol, e.g., with a single message where the non-initiating party informs her Com scheme parameters to the initiating party, along with a NIZKP of correctness. This matter is also trivial in case the public parameters of the Com schemes are equal for both parties, e.g., in case they are directly defined by a CRS setup.

Remark 2.8 (On coin-flipping new commitment scheme parameters). The defined functionality uses Com scheme parameters that are externally defined in its entirety. An alternative ideal functionality could decide herself some random parameters of the Com schemes (though still partially parametrized by a type of Com scheme and at least a security parameter), and then send them along with the output of each party, after which each party would also output to the environment the Com scheme parameters. Correspondingly, in the real world the protocol would incur a respective cost, to allow the parties to interactively decide the random parameters, e.g., based on a simulatable coin-flipping. In the analogous of a CRS, the parties would be able to agree on correct random parameters, but without any isolated party knowing a respective trapdoor (factorization, discrete log). In the analogous of a PKI, each party could learn the trapdoor of her random Com scheme parameters, but the other party would only learn the public parameters. This would be particularly costly for IFC based Com schemes, if requiring the parties to coin-flip a Blum integer with unknown factorization by each individual party.

2.3 Real commitment schemes

This section reviews background notions related to real commitment (Com) schemes, namely bit-commitment (BitCom) schemes (§2.3.1), starting with the basic mandatory properties of *hiding* and *biding*. It also defines the useful (optional) property of XOR homomorphism, and a weaker notion called pseudo-homomorphism (§2.3.2). Notions of *extractability* (§2.3.3) and

equivocability (§2.3.4) are discussed in separate. The focus here is on properties associated with stand-alone secure Com schemes. A more thorough discussion about simultaneously extractable (Ext) and equivocable (Equiv) Com schemes, needed for (two-side) simulatability, is given in Chapter 4. A summary of properties of four different concrete BitCom schemes (§2.3.5) is given in Table 2.3, to facilitate comparing different instantiations used in the S2PC-with-Coms protocol described later in the dissertation. While aspects of non-malleability are mostly left implicit, they can be ensured based on auxiliary non-malleable ZKPs or ZKPoKs performed in the main protocol, in relation to the respective standalone secure commitments. Throughout the dissertation, different uses of commitment schemes are described with varying notation (related to the commitment, the randomness, the committed value, the opening) — a table describing these uses is included in Section B.1.1 in Appendix.

2.3.1 Bit Commitments

Several techniques in this dissertation are based on properties of (some) BitCom schemes, reviewed hereafter. A BitCom scheme [Blu81, BCC88] is a two-party protocol for committing and revealing individual bits. In a *commit* phase, it allows a *sender* to commit to a bit value, by producing and sending a BitCom value to the *receiver*. The BitCom *binds* the *sender* to the chosen bit and, initially, *hides* the bit value from the *receiver*. Then, in an *open* phase, the *sender* discloses (possibly in an interactive manner) information that allows the *receiver* to learn the committed bit and *verify* its correctness. The term *opening* can be used either to denote the information that the sender reveals during the open phase, or the interactive process by which the open phase takes place. In a non-interactive open phase, the opening information may also be called *decommitment* or *bit-encoding*.

Unconditionally hiding. A BitCom scheme is unconditionally hiding if before the *open* phase a *receiver* with unbounded computational power cannot learn anything about the committed bit. A practical instantiation was used by Blum for *coin flipping* [Blu81]. In essence, unconditionally hiding follows from the property that any BitCom has openings for both bits, while at the same time the sender in unable to compute any pair of such openings (for the same BitCom). This infeasibility — a kind of collision resistance — is also known in the literature as a "claw-free" property [GMR84, Dam88a].

Unconditionally binding. A BitCom scheme is unconditionally binding if for every commitment there is a single bit value that a *sender* with unbounded computational power is able to make the *receiver* accept as successful result of the *open* phase. A practical instantiation can be based on public-key probabilistic encryption, e.g., the Goldwasser-Micali encryption scheme [GM84], if the encryption key is unknown to the receiver.

Remark 2.9 (*Unconditional* vs. *statistical* characterization of hiding and binding). The characterization of hiding and binding as unconditional assumes a correct setup of commitment scheme parameters. If the setup phase is prone to a negligible yet non-null statistical chance of error, then the mentioned properties may in rigor be characterized as statistical, instead of unconditional, when referring to the commitment scheme.

Trapdoors. A trapdoor is an element whose knowledge allows computations that are infeasible without the knowledge of the trapdoor. In an *unconditionally hiding commitment scheme with trapdoor* there is a trapdoor that enables opening any bit from any BitCom. If the sender learns the trapdoor then it breaks the *binding* property of the commitment scheme, becoming able to *equivocate* the opening of any BitCom. The knowledge of the trapdoor by a receiver does not break the hiding property of commitments, as it does not reveal any information about which bit the *sender* might have committed to (i.e., which opening the sender might know or be able to perform). In an *unconditionally binding commitment scheme with trapdoor* there is a trapdoor that can be used to efficiently *extract* (i.e., decrypt) the committed bit from any BitCom. If the receiver learns the trapdoor then it breaks the *hiding* property of the commitment scheme. *Equivocation* and *extraction* are possible even for some commitment schemes that are only computational hiding and binding, respectively. They are also possible as capabilities of a simulator, even if not possible to any real participant (sender or receiver). They might possibly not be based on a trapdoor but rather on some other *super power* of the simulator (e.g., capability to rewind the adversary, or access to an ideal functionality in a hybrid world).

A useful combination of commitment schemes with trapdoor. The basis of the forge-and-lose technique (originally described based on the unconditionally binding and unconditionally hiding BitCom schemes with trapdoor [Bra13]) described in this dissertation (§3.1.3) is a combination of *extractable* BitComs and *equivocable* BitComs with trapdoor,

with the *sender* in the extractable scheme being the *receiver* in the equivocable scheme, and knowing a common trapdoor for both schemes. For example, for an IFC instantiation based on Blum integers, assuming intractability (without a trapdoor) of deciding quadratic residuosity, this means using the same Blum integer in both schemes, with its factorization as trapdoor. The remaining description in the dissertation is more well focused on distinctive concepts of extractability and equivocability.

2.3.2 Homomorphisms and pseudo-homomorphisms

Homomorphisms. A commitment scheme is called additively homomorphic (for some additive group operation, e.g., modular sum, in the space of committable values), if any pair of commitments can be combined into a new commitment of the value resulting from applying the group addition to the two originally committed values. Correspondingly, the "randomness" needed to open the new commitment can be obtained by homomorphically combining the two original randomnesses.

Specifically, a BitCom scheme is called *XOR-homomorphic* if any pair of BitComs can be homomorphically combined (under some group operation) into a new BitCom that commits the XOR (i.e., the sum modulo 2) of the two initial committed bits, and if the same can be done with the respective randomnesses. For example this is the case of Blum BitComs and GM BitComs, defined ahead. For modular additive groups over the integers, with modulus larger than 2, a commitment scheme is called *additively homomorphic* if the same properties hold in respect to modular addition of the committed values. This is the case of Pedersen Commitments and ElGamal Commitments defined ahead.

For the purpose of the S2PC-with-Coms protocol (defined in Chapter 3), a XOR-homomorphism (i.e., homomorphism of sum modulo 2) is very useful in several constructions. For example, it is the basis for the connectors of input of P_A, i.e., making a connection between BitComs and wire keys; it enables efficient ZKPs and ZKPoKs related with committed bits, and efficient coin-flipping of random randomness and respective commitments (emulating an ideal functionality that would select random commitments). The property is also useful for linking several S2PC executions, via ZKPs about relations between the input bits of one execution and the input and output bits of previous executions.

Pseudo-homomorphisms. While XOR operations are very useful, some commitment schemes are additively homomorphic over an additive set of integers modulo a group order larger than two. Even though said Com schemes can be used to commit isolated bits, i.e., values 0 or 1, the respective additive homomorphism does not correspond to a XOR operation. For example, the additive-homomorphic combination of two commitments of value 1 would lead to a new commitment of value 2, which is not a bit (0 or 1). A solution for XOR homomorphic operations is nonetheless possible in a restricted yet useful context, without requiring a full-fledged XOR-homomorphism. Specifically, this is possible in applications (e.g., for connectors of input of P_A) where the party performing the operation knows the opening of one *auxiliary* BitCom. In this case, the "homomorphic" XOR can be computed using modular additive operations (addition and subtraction), based on one known value, as follows: XORing (an auxiliary bit) 1 to an unknown *original* bit is equivalent to subtracting the original bit from 1; XORing (an auxiliary bit) 0 to an unknown *original* bit is equal to summing the *original* bit to 0. This operation, i.e., when performed at the level of respective BitComs and randomnesses, is hereafter called a "pseudo XOR-homomorphism." In this way, ElGamal and Pedersen commitments become usable, including (upon proper adaptation) for bit-wise XOR operations over strings. Specifically, this is useful when the receiver party (in respect to whom the original bit and the auxiliary bit are secret at start), after receiving the BitCom resulting from the pseudo-homomorphic transformation either: (i) sees the opening of the new BitCom, but does not see the auxiliary BitCom (and thus cannot tell which homomorphic operation took place); or (ii) sees the opening of the auxiliary BitCom, but not the opening of the resulting BitCom.

A more detailed description of additive homomorphisms and pseudo-homomorphisms, including succinct notation, is given in §B.1.2 in Appendix B.

2.3.3 Extractable commitment schemes

Definition 1 (extractability). *An* extractable *commitment (Ext-Com) scheme is one whose* commit *phase in a simulated execution allows $\mathcal{S}$ in the role of* receiver*, and indistinguishable from an honest receiver in the view of a possibly malicious sender, to* extract *(i.e., learn) the committed value, with probability equal to or larger than a value negligibly-close to the maximum probability with which the* sender *is able to successfully* open *said value.*

There is a natural resemblance between the concept of an encryption scheme and that

of Ext-Com scheme. An encryption scheme is supposed to be decryptable by a receiver; an Ext-Com scheme is supposed to be extractable by the simulator in the role of receiver in a simulated execution, even if there is no key (a trapdoor) to decrypt a commitment.

It follows that any public-key encryption scheme can be used as a basis of an Ext-Com scheme, if augmented with a way for the simulator to obtain the respective key (the trapdoor) and assuming that the receiver of the commitment (who is supposed to remain blind to the committed value) does not know the trapdoor. In contrast, in settings of simulation with rewinding, extraction may happen by means of repeated challenge-response stages (without the rewindable sender noticing said repetition), even if there is no decryption key.

2.3.3.1 GM BitComs

The Goldwasser-Micali probabilistic encryption scheme [GM84] is based on the decisional quadratic-residuosity intractability (DQR) assumption in the multiplicative group of residues coprime with a Blum integer. This can be used directly to define an Ext-BitCom scheme, hereinafter denoted as the GM BitCom scheme, as follows.

- **Setup.** There is a Blum integer modulus N whose respective integer factorization (the trapdoor) is assumed infeasible to find by the *receiver* ($\mathrm{P_R}$), and for which it is assumed infeasible to decide quadratic residuosity of residues with Jacobi Symbol 1. However, by some implicit mechanism the simulator is able find the trapdoor.
- **Commit.** The commit phase corresponds to encrypting and sending the respective ciphertext. Specifically, to commit a bit 1 or 0, $\mathrm{P_S}$ selects a random group element r and sends its modular square r^2 or the modular additive inverse $-r^2$ of the square. (The additive inverse of a square is necessarily a non-quadratic residue with Jacobi Symbol 1, modulo a Blum integer, because the modular additive inverse of 1 has the same property; i.e., a pair of elements with Jacobi Symbol 1 and which are additively inverse vis-a-vis each other is a pair of commitments to different bits.)
- **Open type 1.** To open a bit 1 or 0, $\mathrm{P_S}$ reveals the bit and the used random group element r, letting $\mathrm{P_R}$ verify that the square or additive-inverse of the group element, respectively, is equal to the BitCom value.
- **Open type 2.** Alternatively, the sender may keep the randomness r hidden, and instead just reveal the bit b and send a NIZKP that the commitment is indeed a commitment to

the revealed bit, 0 or 1, i.e., that it is a quadratic or non-quadratic residue, respectively.

Analysis.

- **Extraction.** The simulator in the role of receiver in a simulated execution can decide quadratic-residuosity by using the trapdoor in the GM decryption algorithm.
- **(Non-)malleability with respect to opening.** The open phase of type 2 is particularly useful to prevent malleability with respect to opening. This is relevant for some BitComs (the *outer* BitComs) used in the S2PC-with-BitComs protocol defined in this dissertation, where it may be preferable to not reveal the randomness of the private input or output bits in the context of larger protocols. For example, a malicious party could replicate in another protocol execution a commitment of a private bit of another honest party. When the honest party would later reveal the randomness (in an open phase of type 1), the malicious party would thus become able to also open the commitment.
- **(Non-)malleability with respect to commitment.** For non-malleability with respect to commitment, a GM BitCom must not be replayable by a different party. This type of non-malleability can be achieved by adding to the setup or the commit phases a (non-malleable) ZKPoK of the ability to open the committed values. If the Blum integer is selected by the sender (P_S), then in the setup stage the sender may send the modulus to the receiver along with a ZKPoK of the trapdoor. This is also possible in a global-PKI setup where the integer and the factorization is available to the sender and no one else. The ability to provide such ZKPoK is equivalent to the compute pseudo-square-roots of any GM BitCom, i.e., to open the commitments. This allows non-malleability with respect to commitment because it prevents another party (not knowing the trapdoor) from being able to replicate a commitment, as it would not be able to produce said ZKPoK. If the Blum integer is instead provided by a trusted setup (e.g., CRS) that hides the trapdoor from the sender, then a ZKPoK of trapdoor is not possible. In this case it can be replaced by a ZKPoK of the opening, which must take place in the commit phase.

2.3.3.2 ElGamal Commitments and BitComs

Let g_0 be an arbitrary generator, denoted *global generator*, of a group, with finite order q, where the Decisional Diffie-Hellman (DDH) intractability assumption holds. Let discrete

logs, in multiplicative notation, be considered with respect to the global generator.

ElGamal encryption scheme (multiplicatively-homomorphic). The ElGamal encryption scheme [ElG85] (for encryption of group elements) is as follows:

- **Key-generation.** Select, uniformly at random, a non-negative integer x less than the group order q (i.e., $x \in \mathbb{Z}_q$), and define a new generator g_1 as being the exponentiation ${g_0}^x$ of the global generator g_0 to the power of the random element x. The pair (g_0, g_1) of generators constitutes the public key of an ElGamal encryption scheme. The secret key is the random integer x — the discrete log of the second generator base the first generator.
- **Encryption.** To encrypt a group element $\gamma \in \mathbb{G}$, produce a pair (c_1, c_2) of group elements, as follows: the first element is a random exponentiation ${g_0}^r$ of the global generator g_0, i.e., the global generator to the power of a random exponent r; the second element is the group-product of the respective exponentiation ${g_1}^r$ (i.e., with the same random exponent) of the other generator g_1 by the value γ being encrypted.
- **Decryption.** To decrypt a ciphertext (c_1, c_2), raise the first element c_1 to the power of the secret key x, and use the result ${c_1}^x$ as divisor to divide the second element c_2, thus obtaining the plaintext m.
- **Multiplicative homomorphism.** Given two ciphertexts ((c_1, c_2) and (c'_1, c'_2)), each being a pair of group elements, the group-product per component, i.e., resulting in a new pair of group elements, each being the result (c''_i) of a product of the original group elements (c_i, c'_i) in the respective position, is a new ciphertext (c''_1, c''_2) whose decryption result is the same as the product of the two original plaintexts.

ElGamal commitment scheme for integers (additively-homomorphic, but not extractable). A commitment scheme for non-negative integers less than the group order (i.e., in $\mathbb{Z}_q$) can be defined straightforwardly based on the ElGamal encryption scheme, as follows.

- **Setup.** The pair (g_0, g_1) of generators is defined in a way that guarantees that the receiver ($\mathrm{P_R}$) does not know the discrete log x of the second generator g_1. It is irrelevant whether or not the sender ($\mathrm{P_S}$) knows the discrete log, so the parameters may be selected by $\mathrm{P_S}$.
- **Commit phase.** To commit to an integer $m \in \mathbb{Z}_q$, $\mathrm{P_S}$ raises the global generator g_0 to the power of the integer m being committed, then encrypts the result ${g_0}^m$ with ElGamal encryption and sends the resulting ciphertext to $\mathrm{P_R}$.
- **Open phase (type 1).** To open a commitment, $\mathrm{P_S}$ sends to $\mathrm{P_R}$ the committed value m

and also the randomness r of the ElGamal encryption. To verify correctness, the receiver confirms that the received pair would lead to the previously received ciphertext.

- **Open phase (type 2).** Alternatively, the sender may keep the randomness r hidden, and instead send the committed value m along with a NIZKP that m was indeed the committed value. In practice, after an auxiliary exponentiation and division, the NIZKP can be computationally reduced to a NIZKP of an ElGamal commitment of 0 (§A.3.5).
- **Additive homomorphism.** Given two ElGamal commitments (c_1, c_2) and (c'_1, c'_2) of respective values m and m', computed using respective randomness r and r', the respective group-product (per component) of the two commitments results in a new commitment of the modular sum (modulo the group order q) of the two original committed values, with respective randomness equal to the modular sum of the original randomnesses.

Even though a simulator with the knowledge of the secret discrete log x is able to decrypt an ElGamal ciphertext to obtain the respective encrypted value, the above described ElGamal commitment scheme is not extractable. In fact, in the scope of the Com scheme the plaintext of the encryption scheme is the result of exponentiating the base-generator to the power of the committed value. Thus, extracting the committed value from the commitment alone (with the help of the decryption key) would still require computing a discrete logarithm, which is assumed to be infeasible. The scheme could become extractable by augmenting the commit phase with a ZKPoK of the committed value. However, if wanting to reduce the number of rounds, a respective NIZKPoK (e.g., §A.3.3) could increase the communication complexity.

ElGamal BitCom scheme (extractable and XOR pseudo-homomorphic). The above scheme can be converted into an Ext-BitCom scheme by restricting the message space to the set $\{0, 1\}$ of possible bits and assuming that the simulator has obtained the decryption trapdoor in a setup phase. This allows the simulator to decrypt the ciphertext and extract bit 0 or 1 according to whether the decrypted value is respectively the group identity element $1 (= g_0{}^0)$ or the base generator $g_0 (= g_0{}^1)$. The restriction to a BitCom scheme breaks the additive homomorphism, but preserves an additive pseudo-homomorphism — in this case a XOR pseudo-homomorphism.

If $\mathrm{P_S}$ is required (e.g., for simulatability purposes) to use a correct BitCom, the commit phase needs to be augmented with a ZKP that the committed value is indeed a 0 or a 1 (§A.3.2), i.e., that either the pair (c_1, c_2) of components of the ciphertext, OR the pair $(c_1, c_2/g_0)$ obtained upon having the second component divided by the first generator g_0, is a

pair of elements with same discrete log taken base the pair of two generators (g_0, g_1).

Even if this ZKP (that an ElGamal encryption is an ElGamal BitCom) is not a ZKPoK of the discrete log between the two base generators, it may still allow extractability of the committed bit (though not necessarily of the used randomness) if it allows the simulator to check which element is encrypted (i.e., 1 or g_0). This type of extraction may be useful, e.g., in the S2PC-with-BitComs protocol, when associated to BitComs schemes whose parameters are selected by a trusted setup that does not share the trapdoor with the sender. In other settings, the actual extraction of the trapdoor (the discrete log) can be made via a ZKPoK of a discrete log (§A.3.3), if known by P_S, or by having the second generator be defined via a trusted setup, which the simulator can influence in an actual simulation.

2.3.4 Equivocable commitment schemes

Definition 2 (equivocability). *An* equivocable *commitment (Equiv-Com) scheme is one whose* open *phase in a simulated execution allows $\mathcal{S}$ in the role of* sender*, and indistinguishable from an honest sender in the view of a possibly malicious receiver, to* equivocate *the opening to any intended value, in the domain of committable values and possibly externally decided only after the* commit *phase.*

2.3.4.1 Blum BitCom scheme

A concrete equivocable BitCom scheme is possible based on the multiplicative group of integers modulo a Blum integer with factorization unknown by the *sender*. In the original description by Blum [Blu81], bits 0 and 1 are encoded as group-elements with Jacobi Symbol 1 or −1, respectively. This encoding is a XOR homomorphism, i.e., into the additive group of integers modulo $\mathbb{Z}_2$, with the multiplication modulo the Blum integer being correspondingly mapped into sum modulo 2. The commit phase is then obtained by squaring a random encoding of the bit, and the *opening* is achieved by sending the known square-root and letting the receiver decode it based on the Jacobi Symbol. The scheme is equivocable, as the knowledge of the integer factorization allows computing all the square-roots of any square, thus enabling the opening of any intended bit. For an efficiency reason, the commitment scheme is hereafter slightly adjusted, being described with the help of an auxiliary pre-computed element z (possibly the smallest one) with Jacobi symbol −1, and its respective modular square z'.

Table 2.2: Blum-BitCom scheme and adaptations

A	B	C	D	E	F	G	H	
Commitment scheme	Public params	Commit-able domain	Commit-ment domain	Commit phase		Open phase (type 1)		1
				Randomness selection	BitCom calculation	Message	Verification	2
Original [Blu81]	N	$b \in \mathbb{Z}_2$	QR_N	$r \leftarrow^{\$}$ $\mathrm{JS}_N(1-2b)$	$y = r^2$	(b, r)	$\mathrm{JS}_N(r) =^? 2b-1$ $\wedge\ r^2 =^? y$	3
Alternative	(N, z)	$b \in \mathbb{Z}_2$	QR_N	$r \leftarrow^{\$} \mathbb{Z}_N^*$	$y = r^4 * z'^b$	(b, r)	$r^4 * z'^b =^? y$	4
Generalized	(N, z, k)	$m \in \mathbb{Z}_{2^k}$	QR_N	$r \leftarrow^{\$} \mathbb{Z}_N^*$	$y = r^{(2^{k+1})} * z'^m$	(m, r)	$r^{(2^{k+1})} * z'^m =^? y$	5

Legend: b (committed bit); k (maximum length of committable values, implicitly 1 in BitComs); m (committed bit-string); r (randomness used to commit); z (auxiliary short element with Jacobi Symbol -1); z' (modular square of z); $*$ (multiplication modulo N).

Procedure. A description with succinct notation is presented in row 4 of Table 2.2.

- **Commit.** The sender selects a random group element r, computes its fourth power and then only if the bit being committed is 1 it further multiplies it by the auxiliary square z'. (Special care is required against timing attacks, to prevent an adversarial receiver from distinguishing the time taken to compute a BitCom of 0 from a BitCom of 1, which respectively require two and three modular multiplications.)
- **Open type 1.** To open, the sender reveals the randomness r and the committed bit b, and then the receiver verifies that it leads to the previously obtained commitment.
- **Open type 2.** To open without revealing the randomness, the sender: (i) reveals the committed bit; then (ii) computes the quotient between the commitment and the auxiliary square z' raised to the power of the bit; and then (iii) sends a NIZKPoK of a respective fourth-root (or of a square-root with Jacobi symbol 1).

In this "alternative" description, the BitCom is still a square for which the sender knows a square-root of only one class. The difference, in comparison with the original description by Blum (row 3, from [Blu81]) is that both the commit and the open (type 1) procedure explicitly induce the generation and verification of the correct class of the square-root, based only on three multiplications instead of a more costly Jacobi symbol computation. This also makes more symmetric the commit and verification phases, with the latter just requiring evaluation of the commitment "function," based on the randomness r and the revealed bit b, followed by an equality comparison. This also fits better as a special case of a generalization to BitString commitments, succinctly mentioned in row 5 of Table 2.2, and explained in more detail in §B.4.1.2 in Appendix.

Remark 2.10 (On the representation of square-roots). For convenience (ahead in §2.5.1, when defining a 2-to-1 square scheme and a 2-out-of-1 oblivious transfer), in the context of Blum BitComs (but not for GM BitComs) it is here considered that the two elements in each pair of trivially correlated square-roots (i.e., each square-root and its modular additive inverse) are in fact two representations of the same proper element. Whenever useful to have a single representation, it is stipulated that it is the smaller one, i.e., the minimum between the element and its modular additive inverse. The same interpretation applies in the BitCom space, thus leading each element with Jacobi Symbol 1 to always be a square (even if it is not in terms of the regular multiplicative group interpretation), because there are always two *proper* square-roots either for the value or for its additive inverse. This interpretation is sometimes left implicit, apart for referring to *proper* elements. The importance of this consideration is for example avoiding a selective failure attack, where the sending of a non-square with Jacobi Symbol 1 could otherwise lead the receiver (knowledgeable of the trapdoor) to an error, which would signal to the sender that the element was not a square. A more costly alternative would be to require the sender to always provide a NIZKP that the BitCom is indeed a square. These considerations are not applicable to GM BitComs, where taking the additive inverse of a BitCom would represent changing the committed bit.

2.3.4.2 Pedersen Commitments and BitComs

Pedersen devised an additively-homomorphic (unconditionally hiding) commitment scheme [Ped92] for integers. Consider a cyclic group (in multiplicative notation) and a respective pair of generators for which the sender does not know the discrete log between the two generator. (It is irrelevant whether or not the receiver knows the discrete log). To *commit* a value m, the sender reveals the product of two factors, with the first being the first generator raised to the power of the committed value m and the second being the second generator raised to a random exponent r. The ordered pair (m, r) of exponents is called a *representation* of the commitment c. To *open* a value from c, the sender reveals the known representation (m, r) and then the verifier accepts its first component as the correct committed value if the received pair is indeed a correct representation. The scheme is additively homomorphic, for sum modulo the order of the group. The scheme is equivocable, as the knowledge of the secret discrete log allows computations of several representations of the same group element. Correspondingly, any pair of different representations of the same commitment

allows computation of the discrete log between the two generators.

Pedersen BitComs. The scheme can be restricted to BitComs, by restricting the set of committed values to 0 and 1. A proof of correctness can then be achieved by a NIZKP of an OR relation, combining the committed value is either 0 OR a 1. This can be reduced to a NIZKP of knowledge of a discrete log (base the second generator) of either the commitment OR of the commitment divided by the first generator (see §A.1.6 and Section A.3). The scheme is pseudo XOR-homomorphic, based on its additive homomorphism.

Remark 2.11 (BitComs as commitments of a parity). XOR-homomorphic BitComs could be defined from the bit-string Coms by letting a BitCom to a bit b be a bit-string Com of a random string with the respective parity b [BD90]. This allows a XOR-homomorphism as long as the respective additive sum of the committed elements does not overflow the (odd) group order, e.g., by ensuring that value committed by the Pedersen Com is exponentially large but also exponentially smaller than the group order. In fact, the same idea could also be used with ElGamal BitComs. However, this dissertation will use the pseudo-homomorphic version, because it is enough and easier to enable an optimization based on bit-string Coms, compacting several bits within a single Com.

2.3.5 Summary of concrete BitCom scheme instantiations.

Table 2.3 comprises in a concise view a summary of properties of four BitCom schemes that can be useful in instantiations of the S2PC-with-BitComs protocol described later.

In all exemplified BitCom schemes, the knowledge of the trapdoor (column K) by the party mentioned in column J is optional, i.e., not needed to commit or open a value. However, if in an IFC instantiation the Blum integer is selected by a non-trusted party, e.g., P_S (for GM) or P_R (for Blum), then a ZKP of correctness of the Blum integer should be given.

Remark 2.12 (On the need for a ZKP of correctness). A ZKP of correctness of a Blum BitCom (cell F2) may be avoided for certain applications where the interpretation of group elements considers only *proper* elements. For example, in an insecure application, a sender could try to use a trapdoor-knowledgeable receiver as an oracle for deciding quadratic residuosity of an element (presumed produced as a BitCom but instead possibly obtained

Table 2.3: Summary of properties of concrete BitCom Schemes

A	B	C	D	E	F	G	H	
BitCom scheme name	Base reference	BitCom set (group)	Randomness selection	BitCom sent to P_R	Need NIZKP of good BitCom?	Open message	Verify opening	1
Blum	[Blu81]	QR_N	$r \leftarrow^{\$} \mathbb{Z}_N^*$	$y = r^4 * z^{2b}$	Possibly	(b, r)	$r^4 * z^{2b} =^? y$	2
Pedersen	[Ped92]	$\mathbb{G} = \langle g_0 \rangle = \langle g_1 \rangle$	$y = r \leftarrow^{\$} \mathbb{G}$	${g_0}^b {g_1}^r$	Possibly	(b, r)	${g_0}^b {g_1}^r =^? y$	3
GM	[GM84]	$J_N(1)$	$y = r \leftarrow^{\$} \mathbb{Z}_N^*$	$(-1)^b r^2$	No	(b, r)	$(-1)^b r^2 =^? y$	4
ElGamal	[ElG85]	$\langle g_0 \rangle = \langle g_1 \rangle$	$y = r \leftarrow^{\$} \mathbb{G}$	$(c_1, c_2) \equiv ({g_0}^r, {g_0}^b {g_1}^r)$	Yes	(b, r)	${g_0}^r =^? c_1 \wedge {g_0}^b {g_1}^r =^? c_2$	5

I	J	K	L	M	N	
BitCom scheme name	Who can known trapdoor	Trapdoor	Uncond-itional property	Simulatability operation		6
Blum	P_R	$t = \mathrm{Sqrt}_N(\pm z * \mathrm{NTSqrt1}_N)$	Hiding	Equiv b	$r' = r * t^{1-2c}$	7
Pedersen	P_R	$t = \mathrm{DL}_{g_0}(g_1)$	Hiding	Equiv b	$r' = (2c-1)t^{-1} + r \pmod q$	8
GM	P_S	$t = \mathrm{NTSqrt1}_N$	Binding	Ext b	If $\mathrm{Ver}[t](y \in QR_N)$, then $b = 0$, else $b = 1$	9
ElGamal	P_S	$t = \mathrm{DL}_{g_0}(g_1)$	Binding	Ext b	If $(z \equiv c_2/{c_1}^t) =^? 1$, then $b = 0$ Else if $z =^? g_0$, then $b = 1$ Else ERROR	10

Legend: P_R (receiver); P_S (sender); b (committed bit); c (equivocated bit — used in rows 7 and 8); r (randomness used to commit); z (auxiliary short element with Jacobi Symbol -1 — used in rows 2 and 7); $*$ (integer multiplication modulo N). NTSqrt1 (non-trivially correlated square-root of 1, modulo N, i.e., a modular square-root with Jacobi symbol -1). In column F, the ZKP refers to proving that the BitCom y is well formed, whenever correctness is a requirement (see further notes in Remark 2.12). As previously mentioned, the description of the Blum BitCom was adjusted from its original, in order to allow as randomness any group element, instead of having to select elements from a pre-determined class (i.e., with a pre-determined Jacobi Symbol). Depending on the application, the BitCom set for Blum BitComs (cell C2) may be generalized to $J_N(1)$/proper.

without knowing a respective opening), if the application would require the to abort in case it could not extract a square-root. A ZKP of correctness of a Pedersen BitCom (cell F3) is not needed assuming that it is feasible to check non-interactively that the BitCom element (a group element) is indeed in the space of BitComs, namely when g_0 and g_1 are generators of the same cyclic group and simultaneously group membership decidability is feasible. A ZKP of correctness of a GM BitCom (cell F4) is not needed because group membership can be verified efficiently. A ZKP of correctness of an ElGamal BitCom (cell F5) is necessary whenever the application requires (e.g., due to desired simulatability properties) proving that indeed the value presented as commitment is a BitCom, namely that it commits to a bit instead of some other value (§2.3.3.2).

Definition 3 (Dual BitCom schemes with trapdoor). *Two BitCom schemes are said to be* dual *(with respect to one another) if one has an extraction trapdoor and the other has an equivocation trapdoor and the trapdoors are the same (or feasibly derivable from one another).* For example, the Blum (Equiv-)BitCom scheme and the GM (Ext-)BitCom schemes, when instantiated with the same Blum integer as modulus, are dual with respect to one another. As another example, the Pedersen (Equiv-)BitCom scheme and the ElGamal (Ext-)BitCom scheme, when instantiated with the same public parameters (two generators) are dual with respect to one another. (It is interesting to notice that a BitCom scheme can be *dual* of itself, if it is simultaneously extractable and equivocable and the same trapdoor allows both properties — this would only be useful as a commitment scheme if neither sender nor receiver would know the trapdoor, but the simulator could know it.) Naturally, the binding and hiding properties are assumed to hold only based on the assumption that the respective sender and receiver do not know the equivocation and extraction trapdoor, respectively.

Remark 2.13 (Types of equivocability). The stated definition of equivocability (Definition 2) is in respect to an ability of a simulator impersonating an honest sender, i.e., a simulator that was able to choose the way in which the commitment was generated. A stronger notion of equivocability could require a simulator impersonating a receiver to be able to calculate from the commitment alone the possible openings for any possible committed value. This property was considered by Brassard, Chaum and Crépeau in what they called *chameleon* commitments [BCC88]. Indeed, they also present an IFC based BitCom very similar to Blum BitComs, but not requiring Blum integers, and a DLC based BitCom precursor of Pedersen BitComs. They explain why their described IFC Coms are chameleon — the trapdoor allows computation of any opening from any BitComs — and why the DLC ones are not — even with the trapdoor it is not possible to compute discrete logs. In this dissertation, some components of the protocol do require the ability to extract an opening from a commitment, as a chameleon Com would allow, but this is actually applied to extractable BitComs, e.g., ElGamal BitCom, and achieved via an ad-hoc ZKPoK of the openings. Beaver [Bea96c] defined an even more stringent equivocability property, requiring that the openings must be extractable even by a simulator that does not corrupt sender and receiver — while such property is useful for adaptive corruption scenarios, it is not considered herein.

2.4 S2PC via cut-and-choose of garbled circuits

This section reviews background notions of S2PC solutions based on a cut-and-choose (C&C) of garbled circuits (§2.4.1), including application of a random seed checking technique (§2.4.2). It also mentions independent works on reducing the number of garbled circuits (§2.4.3.1), and briefly comments on other S2PC approaches (§2.4.3.2).

2.4.1 C&C-GCs-based S2PC

Basic garbled-circuit approach. The theoretical feasibility of S2PC, for functions efficiently representable by Boolean circuits, was initially shown by Yao [Yao86] (e.g., see [BHR12, §1] for a brief historical account of the origin of the garbled-circuit approach [GMW87a, BMR90, NPS99].) In the *semi-honest model* (where parties behave correctly during the protocol) simplified to the 1-output setting, only one of the parties (P_B) intends to learn the output of an agreed Boolean circuit that computes the desired function. The *basic GC approach* starts with the other party (P_A) building a GC — a cryptographic version of the Boolean circuit, which evaluates keys (e.g., random bit-strings) instead of clear bits. The GC is a directed acyclic graph of garbled gates, each receiving keys as input and outputting new keys. Each gate output key has a corresponding underlying bit (the result of applying the Boolean gate operation to the bits underlying the corresponding input keys), but the bit correspondence is hidden from P_B. P_A sends the GC and one circuit input key per each input wire to P_B. Then, P_B obliviously evaluates the GC, learning only one key per intermediate wire but not the respective underlying bit. Finally, each circuit output bit is revealed by a special association with the key learned for the respective circuit output wire. By letting the parties hold additional random bits as part of their private inputs, S2PC can be performed for probabilistic functionalities, i.e., with probabilistic outputs. Lindell and Pinkas [LP09] prove the security of a version of Yao's protocol (valid for a 2-output setting).

Garbling schemes can be implemented in diverse ways [BHR12]. This dissertation abstracts from specific constructions, except for making the typical assumptions that: (i) with two valid keys per circuit input wire (and possibly some additional randomness used to generate the GC), P_B can *verify the correctness* of the GC, in association with the intended Boolean circuit, and determine the bit underlying each input and output key; and (ii) with a single key per circuit input wire, P_B can *evaluate* the GC, learning the bits corresponding to the

obtained circuit output keys, but not learn additional information about the bit underlying the single key obtained for each input wire of P_A and for each intermediate wire.

Oblivious transfer (OT). An essential step of the basic GC-based protocol requires, for each circuit input wire of P_B (the GC-evaluator), that P_A (the GC-constructor) sends to P_B the key corresponding to the respective input bit of P_B, but without P_A learning what is the bit value. This is typically achieved with 1-out-of-2 OTs (e.g., see explanation in Section 2.5). Some protocols use enhanced variations, e.g., committing OT [CGT95], committed OT [KS06], cut-and-choose OT [LP11], authenticated OT [NNOB12], string-selection OT [KK12]. In practice, the computational cost of OTs is often significant in the overall complexity of protocols, though asymptotically the cost can be amortized with techniques that allow extending a few OT to a large number of them [Bea96b, IKNP03, NNOB12].

The S2PC-with-BitComs protocol presented in this dissertation uses OTs at the level of BitComs. For each circuit input bit of P_B, P_B produces a respective BitCom, as the first message of an OT. Then, the protocol proceeds depending on the type of BitCom scheme. If the BitCom scheme has been decided (e.g., in a setup phase) as an Equiv-BitCom scheme with equivocation trapdoor known by P_A and such that the trapdoor allows determining exactly two openings from any BitCom, then P_A is immediately able to determine two values from which P_A knows only one — a *2-out-of-1 OT*, where P_B chooses *one* value and leads P_A to learn *two* values. The method contrasts with a typical 1-out-of-2 OT (commonly used directly at the level of wire keys), where P_A chooses *two* keys and leads P_B to learn *one* of them.

If, instead, the BitCom scheme is an additive homomorphic commitment scheme with extraction trapdoor known by P_B (i.e., and encryption scheme), then P_A is able to homomorphically re-encrypt one of two chosen values, without knowing which one P_B will be able to decrypt — i.e., a 1-out-of-2 OT.

Cut-and-choose approach. Yao's protocol is insecure in the malicious model. For example, a malicious P_A could construct an undetectably incorrect GC, by changing the Boolean operations underlying the garbled gates, but maintaining the correct graph topology of gates and wires. To solve this, Pinkas [Pin03] proposed a C&C approach, achieving 2-output S2PC via a single-path approach where only P_B evaluates GCs. A simplified high level description follows. P_A constructs a set of GCs. P_B *cuts* the set into two complementary subsets and *chooses* one to *verify* the correctness of the respective GCs. If no problem is found, P_B

evaluates the remaining GCs to obtain, from a consistent majority, its own output bits and a masked version of the output of P_A. P_B sends to P_A a modified version of the masked output of P_A, without revealing from which GC it was obtained. Finally, P_A unmasks her final output bits. This approach has two main inherent challenges: (1) how to *ensure that input wire keys are consistent across GCs, such that equivalent input wires receive keys associated with the same input bits (in at least a majority of evaluated GCs)*; (2) how to *guarantee that the modified masked-output of P_A is correct and does not leak private information of P_B*. Progressive solutions proposed across recent years have solved subtle security issues, e.g., the selective-failure-attack [MF06, KS06], and improved the practical efficiency of C&C-GC-based methods [LP07, Woo07, KS08a, NO09, PSSW09, LP11, SS11]. As a third challenge, the number of GCs still remains a primary source of inefficiency, in these solutions that require a correct majority of GCs selected for evaluation. For example, achieving 40 bits of statistical security requires at least 123 GCs (74 of which are for *verification*). Asymptotically, the optimal C&C partition (three fifths of verification GCs) leads to about 0.322 bits of statistical security per GC [SS11].

The BitCom approach in this dissertation deals with all these challenges. First, taking advantage of XOR-homomorphic BitComs, the verification of consistency of input wire keys of both parties is embedded in the C&C, without an ad-hoc ZKP of consistency of keys across different GCs. Second, P_B can directly learn, from the GC evaluation, openings of BitComs of one-time-padded (i.e., masked) output bits of P_A, and then simply send these openings to P_A. Privacy is preserved because the openings do not vary with the GC index. Correctness is ensured because the openings are verifiable (i.e., authenticated) against the respective BitComs. Third, the BitCom approach enables the forge-and-lose technique, which reduces the correctness requirement to only having at least one correct *evaluation* GC, thus increasing the statistical security to about 1 bit per GC (see details in §3.1.4).

A 2-output S2PC protocol (e.g., with different outputs for each party) can also be achieved with each party playing once as GC evaluator of only her own intended circuit, i.e., with the circuits evaluated by each party only computing her respective output. Dual path approaches have been previously conceptualized at high level (e.g., [Kir08, §6.6] and [SS11, §1.2]). The BitCom approach herein also makes it trivial to ensure the same input (or transformations thereof) across the two executions.

Since the garbling scheme is abstracted, the protocol is also compatible with many garbling

optimizations, e.g., *point and permute* [NPS99], *XOR for free* [KS08b], *garbled row reduction* [PSSW09], *dual-key cipher* [BHR12], *two halves make a whole* [ZRE15].

Dual execution approaches for covert adversaries. Other dual-path approaches have been proposed using a single GC per party (i.e., not C&C-based), but with potential leakage of one bit of information [MF06, HKE12]. Subsequent improvements have been achieved by Kolesnikok et al., [KMRR15], allowing a range of tradeoffs between different levels of probability of leakage and reducing the amount of leakage in case of successful malicious behavior. This dissertation is focused on fully malicious security, i.e., requiring a negligible probability of malicious leakage in spite of a single evaluator of garbled circuits.

2.4.2 Selecting the cut-and-choose partition

A crucial aspect of a cut-and-choose protocol is the selection of the cut-and-choose partition challenge, and its integration with the commit and response stages. Intuitively, a malicious garbled circuit constructor (P_A) must not learn the cut-and-choose partition before it is bound to concrete garbled circuits, or otherwise it could construct good circuits for all the check instances and bad circuits for all the evaluation circuits. Conversely, a simulator (in the role of P_A in a simulated execution) needs to be able to induce any circuit output (the one decided by the ideal S2PC functionality in the ideal world), and so it needs to find which circuits to construct correctly (for the check operation) and which ones incorrectly (for the evaluation operation). There are several ways by which to enable the simulator to find or induce the cut-and-choose challenge it needs. If interactivity is not a problem, then the challenge could be decided in its own separate stage, via a simulatable coin-flipping protocol, i.e., where a simulator is able to induce the final outcome. However, for a reduced interaction, namely not requiring any new communication step, better methods are possible based on commitment schemes, as described in the next paragraphs. The next paragraphs describe two methods — one based on an equivocable commitment, the other on extractable commitments.

The equivocable method, combined with a NPRO. One approach to enable simulatability is to have P_A use an equivocable commitment scheme to commit to the garbled circuits and related elements, then have P_B decide the cut-and-choose challenge, and only then have P_A open the garbled circuits and respective related elements (for check or evaluation).

In this method the simulator would be able to equivocate the garbled circuits. This technique can be traced back to a similar use by Damgård in the context of Σ protocols (3-round protocols of the form commit-challenge-response [Dam00]) for zero-knowledge proofs. After seeing a Σ-challenge decided by the verifier, the simulator (in the role of prover in a simulated execution) is able to produce a random pair of Σ-commit and Σ-response messages that are consistent with the Σ-challenge, still in time to convince the verifier that the Σ-commit message is the one that had been committed. Adapting the technique to a S2PC based on cut-and-choose, in the response stage the simulator is able to respond correct garbled circuits for the check instances and fake garbled circuits for the evaluation instances, the later designed in a way that their circuit output is always the one decided by the ideal functionality in the ideal world. In practice, the Equiv-Com only has to be applied to a short collision-resistant hash of the sequence of elements being committed.

In order to reduce interaction, it may instead be P_A to decide the challenge, as a (non-programmable) random oracle (NPRO) image of the Equiv-Com [Lin15], where the NPRO outputs a pseudo-random string whenever the input is used for the first time. As a new attack a malicious P_A becomes able to brute-force many trial-and-error attempts of different NPRO pre-images, trying to obtain an NPRO output that encodes a cut-and-choose partition whose all check indexes correspond to good instances and all evaluation instances correspond to bad instances. To prevent this, the statistical security parameter needs to be transformed into a new (larger) computational security parameter, to account for the computational power of P_A. Specifically, the new parameter must be such that the respective cut-and-choose parameters are so large that P_A does not have enough computational power to make enough trials, in feasible time, till finding one pre-image whose output challenge is consistent with the bad instances produced by P_A. Several techniques may mitigate the increase in the security parameter, enabling consideration of a short-term computational security parameter specific for the use of the NPRO. For example, if the protocol is designed to have a short duration deadline, e.g., P_B would accept a reply from P_A only if it is received after less than one minute of time has passed since the S2PC execution started, then it must only be guaranteed that P_A is not able to brute-force the NPRO output in said duration. The techniques may be based on the analogous of a "salt" and "iteration count" used to mitigate brute-force attacks in password-based key derivation schemes [TBBC10]. For example, the protocol may be designed to require that the NPRO pre-image includes the execution context (e.g., including the session and subsession identifiers) and some unpredictable nonce proposed by P_B for

each execution, thus limiting the capability of trial-and-error attempts by P_A, before the beginning of the protocol execution (i.e., before knowing the "salt"). Additionally, as a design decision the NPRO (or some function that in practice is replacing it) may be designed to require at least a certain amount of time to compute, not enough to significantly affect the overall protocol execution time, but enough to reduce by several orders of magnitude the number of trial-and-error attempts by P_A.

The extractable method, combined with OTs. Another approach is for P_B to initially commit the cut-and-choose challenge, using an Ext-Com, even before P_A builds the garbled circuits. This allows the simulator (impersonating P_A interacting with a black-box possibly malicious P_B) to extract the challenge partition and know exactly which instances will be selected for check and which ones for evaluation. A straightforward technique would require an extra communication round where P_B would open the committed challenge after P_A would have committed to the circuits. However, this communication round can be avoided, as P_A does not really need to know the partition, as long as it is able to provide correct replies to each instance. This is exactly the kind of role that an oblivious transfer may provide, i.e., letting P_A send two types of answers but with P_B only receiving one kind. Without attempting a reduction of communication steps, this type of technique was used in a prior S2PC protocol by Lindell and Pinkas [LP11], letting P_B receive two keys per wire for check circuits and one key per wire for evaluation circuits, but without disclosing to P_A which instances were of each type (check or evaluation). More recently, the technique was used directly to achieve non-interactive S2PC (i.e., one message to each side) [AMPR14]. The advantage in comparison with the equivocable method is that it does not require equating the statistical parameter up to the computational parameter, because P_A will not get to learn the cut-and-choose partition. This extractable method was not considered in the original forge-and-lose paper, but it can be incorporated into the protocol as an option to remove the interaction between the commit and the response stages of the cut-and-choose (as done in [AMPR14]). Combined with the forge-and-lose technique, which also allows non-interactive recovery of the input of P_A, the statistical security is thus equal to the interactive case.

The random seed checking (RSC) technique.

As pointed out by Goyal et al. [GMS08], the communication complexity associated with the transmission of garbled circuits within a C&C approach can be significantly improved by

avoiding communication of the check circuits. The technique can be derived from two simple observations. First, the sending of garbled circuits in the COMMIT stage can be deferred to the RESPOND stage, as long as P_A commits to them during the COMMIT stage. Second, since in a typical cut-and-choose the VERIFY stage only requires P_B to know elements that do not depend on the private inputs of P_A and P_B, those elements can be pseudo-randomly generated from a short random seed and other public information. Based on these observations, the technique is implemented as follows. In the COMMIT stage, P_A pseudo-randomly generates, for each challenge index (of unknown type), the necessary elements, based only on a small random seed, and then computes a respective short (compressive) commitment (at least computationally binding) of the generated elements. Henceforth, these are denoted as RSC commitments, as differentiation from other commitments used within the protocol. Then, in the RESPOND stage: for each *check* instance, P_A only sends the respective small random seed, thus allowing P_B to make all respective verifications; for each *evaluation* challenge, P_A simply sends the generated elements (but not the random seed) and the respective responses for evaluation. The optimization in communication is clear: ignoring the size of the short RSC commitments and random seeds, the communication associated with verification challenges is eliminated, while the one associated with evaluation challenges remains the same.

Remark 2.14 (Hiding the CR-hashes of the RSC technique). While [GMS08] proposed to use a CR-Hash to bind P_A to the circuits, in rigor a simple CR-Hash of a circuit does not guarantee hiding a CR-Hash does . Thus, in general a hiding commitment must be used, if the protocol is interactive with respect to letting P_B observe the hash before selecting the cut-and-choose partition, and if simulation is considered in a setting without rewinding. Instead of relying on a random oracle to argue about the hiding properties of the CR-Hash, the hash can be committed by P_A, using "randomness" also generated by the random seed, and then have the commitment be sent instead of the hash.

Transmitting the RSC commitments and seeds. In the cut-and-choose selection method based on Equiv-Coms and a NPRO the RSC commitments of CR-hashes can be replaced by a single RSC Equiv-Com, with independent randomness. Then, P_A only has to open the seeds of the check instances and the elements (garbled circuits and other auxiliary element) of evaluation instances, and the randomness of the equivocable commitment, thus letting P_B verify that they are consistent with the Equiv RSC commitment.

In the cut-and-choose selection method based on Ext-Coms and OTs the use of the RSC technique is more difficult because P_A does not get to learn the cut-and-choose partition and thus does not know for which instances to send seeds and for which to send the actual circuits and other elements. A work-around suggested in [AMPR14] involves an erasure code, applicable if the cut-and-choose partition uses a fixed number e of evaluation challenges and number v of check challenges. Since P_A does not learn the cut-and-choose partition, it instead sends e erasure-code shares in clear, and then sends the remaining v seeds via OT, from which P_B can reconstruct v circuits using a PRG, and use them as additional shares, to reconstruct enough good circuit by erasure decoding.

2.4.3 Other related work on S2PC

(Note: some work subsequent to the original forge-and-lose paper is compared in §3.6.2.)

2.4.3.1 Two other optimal C&C-GCs

Two recently proposed C&C-GCs-based protocols [Lin13, HKE13] also minimize the number of GCs, requiring only that at least one evaluation GC is correct.

Lindell [Lin13] enhances a typical C&C-GCs-based protocol by introducing a second C&C-GCs, dubbed *secure-evaluation-of-cheating* (SEOC), where P_B recovers the input of P_A in case P_B can provide two different garbled output values from the first C&C-GCs. The concept of input-recovery resembles the forge-and-lose technique, but the methods are quite different. For example, the SEOC phase requires interaction between the parties after the first GC evaluation phase, whereas in the forge-and-lose the input-recovery occurs offline.

Huang, Katz and Evans [HKE13] propose a method that combines the C&C-GCs approach with a verifiable secret sharing scheme (VSSS). The parties play different roles in two symmetric C&C-GCs, and then securely compare their outputs. This requires the double of GCs, but in parallel across the two parties. By requiring a predetermined number of verification challenges, the necessary number of GCs is only logarithmically higher than the optimal that is achieved with an independent selection of challenges. In their method, the deterrent against optimal malicious GCs construction does not involve the GC constructor party having her input revealed to the GC evaluator.

In the SEOC and VSSS descriptions, the method of ensuring input consistency across

different GCs is supported on DLC assumptions. The descriptions do not consider general linkage of S2PC executions related with output bits, but their input bits are also committed using XOR-homomorphic BitComs. In contrast, the original S2PC-with-BitComs with forge-and-lose was based on IFC supported on Blum integers and required a lower number of exponentiations, though with each exponentiation being more expensive due to the larger size of group elements and group order, for the same cryptographic security parameter. The current dissertation generalized the previous instantiations to also allow DLC-based BitCom schemes.

2.4.3.2 Other approaches

Jarecki and Shmatikov [JS07] described a S2PC protocol with committed inputs, using a single verifiably-correct GC, but with the required number of exponentiations being linear in the number of gates. In comparison, this dissertation is focused on garbling schemes based on symmetric primitives (e.g., block-ciphers, whose greater efficiency over-compensates the cost of multiple GCs in the C&C), and the required number of exponentiations to be linear in the number of circuit input and output bits and in the statistical parameter. Even a size-efficient garbled circuit can be proven correct via a size-efficient ("succinct") zero knowledge proof (i.e., "argument") [Gro10], but with known methods this would significantly hinder the efficiency of communication, again requiring at least one expensive group operation (e.g., an exponentiation) per gate, much more expensively than repeating the generation and evaluation of several garbled circuits within a cut-and-choose approach.

Nielsen and Orlandi proposed LEGO [NO09], and more recently Frederiksen et al. proposed Mini-Lego [FJN+13], a fault-tolerant circuit design that computes correctly even if some garbled gates are incorrect. Their protocol, which uses a cut-and-choose at the garbled-gate level (instead of at the GC level) to ensure that most garbled gates used for evaluation are correct, requires a single GC but of larger dimension. It would be interesting to explore how to integrate a forge-and-lose technique into their cut-and-chose at the gate level.

Kolesnikov and Kumaresan [KK12] described a S2PC slice-evaluation protocol, based on information theoretic GCs, allowing the input of one GC to directly use the output of a previous GC. Their improvements are valid if the linked GCs are shallow, and if one party is semi-honest and the other is covert. In contrast, the S2PC-with-BitComs protocol in this dissertation allows any circuit depth and any party being malicious.

Another approach for S2PC is to interactively evaluate each multiplicative gate, based

on oblivious transfers [GMW87a]. This approach would inherently require a number of rounds linear in the multiplicative depth of the circuit being obliviously evaluated, but it may nonetheless be instantiated with competing computational complexity, e.g., as developed by Nielsen et al. [NNOB12]. However, the approach may become unsuitable when network delay is an issue and simultaneously the circuit depth is very large — conversely, the approach in this dissertation is focused on constant-round protocols.

Mohassel and Riva [MR13] devised a protocol that does not require exponentiations (in the online phase), and instead only requires symmetric-key operations in number linear in the product of the circuit size and the statistical parameter, namely avoiding exponentiations. (In the offline phase it still requires exponentiations in number linear in the statistical parameter, to bootstrap the OT extension technique.) They achieve this by using garbled circuits to check the consistency of inputs and outputs of other garbled circuits. In contrast, the protocol in this dissertation uses public-key operations that are useful to provide a connection with public-key commitments that are externally meaningful (e.g., in a multi-party PKI setting).

In the garbled circuit approach, the size of the circuit may grow very large in comparison with the time that it would take to (non-securely) evaluate a program with access to random access memory. Since the memory access pattern may be unknown before the computation (i.e., depend on the private input), the circuit circuit requires a linear number of gates for each isolated memory access. The use of oblivious RAM [GO96, GGH+13] may mitigate this problem, by enabling a polylogarithmic communication-complexity for each memory access, asymptotically more efficient than the linear complexity required by a garbled circuit. Still, actual implementations may be impractical for reasonable sized parameters and it applicability is not useful for all kinds of functions. In a complementary perspective, S2PC based on garbled circuits may also be useful as a building block for constructions of general oblivious RAM.

One technical limitation of the garbled circuit approach it its communication complexity. The approach requires communicating a (garbled version of a) completely unrolled circuit that computes, in a straight-line manner, the function whose oblivious evaluation is intended. Conversely, S2PC is possible with communication sub-linear in the circuit size, a.k.a. succinct S2PC, based on stronger primitives, such as fully homomorphic encryption [Gen09], and reusable garbled circuits [GKP+13] (based on functional encryption). A very recent work by Boyle et. al also breaks the barrier of the circuit size, using DDH assumptions to allow communication sub-linear in the size of the unrolled circuit [BGI16]. Even though the

mentioned techniques allow asymptotically better communication complexity, the state-of-the-art constructions are not (yet) practical from a computational complexity perspective.

2.5 BitCom-based oblivious transfer

Oblivious transfer (OT) is an important cryptographic primitive for S2PC. An early version was devised by Rabin [Rab81], with P_A "sending" one chosen value, and then P_B "receiving" it only with probability 1/2. A different functionality was thereafter formalized as 1-out-of-2 OT [EGL85], with P_A choosing and "sending" two values, and then P_B only learning one in the position of its choice. Both OT flavors are equivalent [Cré88], in the sense that one can be achieved from the other. Yet, 1-out-of-2 OT is directly more useful in garbled-circuit based protocols, allowing the evaluator party (P_B) to learn only one key out of two keys chosen by the constructor party (P_A) for each input wire of P_B in each garbled circuit. It is possible to efficiently implement 1-out-of-2 OT [NP01], including with a single-round protocol resilient to malicious parties [PVW08]. In the setting of this dissertation it is useful to consider OT functionalities that also provide to P_A a BitCom of the bit position chosen by P_B.

More generally, by abstracting which party initially "chooses" a value (i.e., apart from the private bit of P_B), and focusing only on the output, a $(2,1)$-OT can be defined as a functionality that allows P_A to output two values (x_0, x_1) and P_B to output one position bit b and one value x_b from the respective position in the pair of values known by P_A, without P_A knowing which value is learned by P_B, and without P_A knowing the other value.

In contrast to a 1-out-of-2 OT (1/2 OT), a 2-out-of-1 OT (2/1 OT) — denomination from [Bra13] — is a (2,1)-OT where P_B "chooses" one value and then lets P_A learn two values, one of which being the one chosen by P_B. Upon a setup assumption defining a public key for P_A, such an OT can be performed with a single message from P_B to P_A, as many times as desired.

From an output perspective (and assuming that the private input values are also part of the output), 1-out-of-2 OT may be considered equivalent to 2-out-of-1 OT, in the sense that one can be transformed in the other at the expense of a single extra message. Yet, intuitively, in a S2PC protocol they may be used with different application perspectives: 1-out-of-2 OT is intuitively associated with exchange of input keys of garbled circuits; 2-out-of-1 OT is used in the S2PC-with-BitComs in this dissertation for exchange of openings of BitComs (independently of the number of garbled circuits). Also, in a non-interactive 2-out-of-1 OT

there must be a computational relation between the two values learned by P_A (e.g., two non-trivially correlated square-roots of the same square), whereas in a 1-out-of-2 the two values can typically be an arbitrary pair of values of the same set. Notwithstanding, the transcript of a 1-out-of-2 protocol in the view of P_B might also be non-unconditionally hiding (i.e., only computationally hiding) of a transformation between the two values.

An OT construction is described below for each of the two approaches (1-out-of-2 and 2-out-of-1). Their distinctive concrete application is described only later (§3.3.2), when explaining the connectors for the input of P_B in the S2PC-with-BitComs protocol.

2.5.1 2-out-of-1 OT based on Blum BitComs

Before specifying a concrete 2-out-of-1 OT construction, it is useful to define a supporting structure, inspired by the Blum BitCom scheme.

Definition 4 (2-to-1 square scheme). *For a suitably-defined efficient function f of the randomness and the committed bit, a BitCom scheme is called a* 2-to-1 square scheme under *f if it satisfies the following three useful properties:*

1. ***(Proper openings.)*** *All BitCom openings (i.e., the values obtained during an open phase) of a bit from a specific BitCom have the same f-image, different across the two possible bits. In other words, each BitCom has exactly two possible f-images of openings.*
2. ***(From trapdoor to openings.)*** *There is a trapdoor which allows extracting the pair of f-images of openings of any BitCom (a* non-trivially correlated pair *of square-roots).*
3. ***(From openings to trapdoor.)*** *Any non-trivially correlated pair of f-images of openings is a trapdoor. (thus, a non-trivially correlated pair of openings is also a trapdoor)*

Under an appropriate representation of (proper) group elements (see Remark 2.10), e.g., if any group element is considered equivalent to its modular additive inverse, the Blum BitCom scheme (both the original and the alternative descriptions) satisfies the enumerated properties:

1. In the original description (row 3 in Table 2.2), a BitCom is the square of a randomness of class equal to the committed bit. Each BitCom has four square-roots, two per bit (i.e., per class), but it is possible to define a single proper square-root per bit. Specifically, the scheme is a 2-to-1 square scheme under the reduction (the f function) to proper elements (i.e., from any randomness output the mininum between itself and its additive inverse).

In the alternative description (row 4 in Table 2.2), adopted herein, a Blum BitCom is the fourth power of the randomness r, further multiplied by the square of an auxiliary element z of class 1 only if the committed bit is 1. The scheme is a 2-to-1- square scheme under the function defined by taking the square of the randomness and multiplying by the auxiliary element z of class 1 if and only if the committed bit is 1. The f-image of any opening is thus a square-root of class equal to the committed bit. Furthermore, the f-image of an opening of 0 is a principal square-root, and the f-image of an opening of 1 is a principal square-root multiplied by the auxiliary element.

2. The needed trapdoor is the factorization of the Blum integer, i.e., the knowledge of the factorization allows feasible computation of the two proper square-roots of any BitCom. A modular square-root of a BitCom can be computed modulo each prime and then the square-root modulo the Blum integer follows from application of the Chinese Remainder Theorem. In the original description (row 4), a non-trivial correlated square-root can then be obtained by simply multiplying the non-trivial square-root of 1. The expression *non-trivially correlated pair* means that the two elements (proper square-roots) are related but cannot be simultaneously found (except with the help of a trapdoor). In the alternative description (row 3), the pair of proper square-roots can be obtained by calculating a principal square-root of the BitCom and then dividing the auxiliary element z. In this case the two proper square-roots are trivially correlated (because the auxiliary element z is public), but nonetheless the actual openings of a BitCom are non-trivially correlated.
3. The knowledge of any two (non-trivially correlated) proper-square roots allows an efficient computation of the prime factors of the Blum integer. In the original description, it is enough to compute the greatest common divisor between the Blum integer and the sum of the two proper square-roots. In the alternative description, it is the greatest common divisor between the proper square-root corresponding to 0 and the result of multiplying the auxiliary element z by the proper square-root corresponding to 1.

A *2-out-of-1 OT* from the second property. Let there be a 2-to-1 square scheme with public parameters known by P_A and P_B, and with trapdoor known only by P_A. Let P_B be initialized with a private input bit b. P_B samples randomness suitable for a BitCom of its input bit, and then sends the respective BitCom to P_A. P_A then uses its trapdoor to extract the respective two proper (f-image) square-roots.

***Forge-and-lose* from the third property.** The third property (also shared by the Pedersen BitCom scheme) is useful for the forge-and-lose technique, as the discovery (by P_B) of a pair of openings (i.e., a trapdoor of P_A), in case P_A acted maliciously, is the condition that allows P_B to decrypt the input bits of P_A. In Blum BitComs, the integer factorization can be found from any pair of proper square-roots of the same square. In Pedersen commitments, any two distinct representations of the same value can be used to find the discrete log between the two generators.

Remark 2.15 (A difference between Blum and Pedersen BitComs). Both Blum and Pedersen BitCom schemes are unconditionally hiding, XOR pseudo-homomorphic and satisfy the first and third properties of a 2-to-1 square scheme. However, the Pedersen scheme does not satisfy the second property, because the trapdoor does not allow computing an opening if one is not already known. While it is conceivable a 1-out-of-2 OT where P_B outputs one representation and P_A outputs two representations of the same Pedersen BitCom, it is not clear how this could be achieved with a sngle message (from P_B to P_A).

Remark 2.16 (ZKPoK to avoid selective failure attack). A ZKPoK of a valid opening of a BitCom (also serving as ZKP of correctness) may be required to avoid selective failure attacks. For example, while Blum BitComs are supposed to be squares in a given group, the decisional quadratic-residuosity (DQR) is also assumed to be intractable to the sender of the BitCom. If the actions of a receiver, knowledgeable of the trapdoor, is distinguishable between being able vs. being unable to extract a square-root (i.e., between the BitCom being valid or invalid), then a malicious sender could potentially use the receiver as an oracle to solve the DQR problem. For example, if the receiver would abort if and only if detecting a non-square, then the malicious sender would learn that the sent element was indeed a non-square. In some circumstances the ZKPoK can be avoided, e.g., if making a reduction to using only proper elements, i.e., making sure that the receiver will treat squares and non-squares (with Jacobi symbol 1) in the same way, namely extracting pseudo-square-roots and not disclosing information about them.

2.5.2 1-out-of-2 OT based on ElGamal BitComs

In 1-out-of-2 OT, P_A "chooses" a pair of values and P_B chooses the position of the value that it wants to learn. As output, P_B learns only the value in the selected position and P_A does not learn anything about said position. 1-out-of-2 OT can also be achieved non-interactively after a public key setup [BM90]. It can also be achieved somewhat efficiently, securely against malicious adversaries, with a single-round (i.e., 2 messages) of communication [PVW08]. Considering the application of the OT in the context of the cut-and-choose structure of the S2PC-with-Coms protocol in this dissertation, the protocol only needs to be simulatable against a malicious P_B, and so a simpler protocol is possible. More specifically, the cut-and-choose structure already thwarts the case of a malicious sender, because the sender reveals the randomness used for check instances, and because the OT is performed (for both check and evaluation instances) before the sender knows the cut-and-choose partition of challenges.

The simpler OT, i.e., designed for an honest-sender, is as follows. As first message, the receiver (P_B) sends to the sender (P_A) an additively homomorphic commitment of a 0 or a 1, and a NIZKP that it is indeed a BitCom (i.e., that it commits to a bit). If need-be, it also sends a ZK proof that enables a simulator to extract the committed bits. A ZKPoK of the committed bit may be avoided if a ZKPoK of the trapdoor has already taken place (e.g., in a PKI setting). In a CRS setting, the ZKPoK can be reduced to a NIZKP of same committed bits in respect to other BitComs (outer-Coms in the S2PC-with-Coms protocol) whose knowledge of opening would anyway have to be proven with a (NI)ZKPoK. As second and final message, the sender homomorphically encrypts a linear combination of two exponents (and also randomizes the ciphertext to ensure semantic hiding of the exponents), and sends it to the receiver. The decryption by P_B then yields the exponent that in the linear combination was the coefficient with respective degree (0 or 1). In summary, this involves each party sending one ElGamal encryption to the other, and the receiver (P_B) also sending a NIZKP that the first encryption is an ElGamal BitCom.

The application of this 1-out-of-2 OT in the S2PC-with-Coms protocol is described at high level in §3.3.2.2. For each input bit of P_B, the first message is required only once, and the second message is repeated once for each evaluation instance. The protocol is specified in more detail in §B.4.2.1 in Appendix B.

Chapter 3

S2PC-with-Coms and the forge-and-lose technique

This chapter describes and analyzes the protocol for S2PC-with-commitments. It covers results published in the paper [Bra13] that introduced the *forge-and-lose* technique for S2PC, as well as subsequent improvements. It contains a revised protocol description and analysis, now with a better focus on reducing the interaction between parties, and suitable to a wider set of cryptographic instantiations (IFC and DLC), trusted setups (GPKI, GCRS) and type of BitCom schemes (Ext and Equiv), including bit-string commitments (additively pseudo-homomorphic). It also contains an updated communication-complexity benchmark and a revised proof of security, based on simulation without rewinding.

In regard to S2PC, the protocol uses a cut-and-choose approach over garbled circuits, with significant advantages in comparison with traditional protocols that require a majority of the *evaluated* garbled circuits to be correct. By augmenting the cut-and-choose with a *forge-and-lose* technique [Bra13], the protocol only requires that at least one *evaluated* garbled circuit is correct. This reduces the number of garbled circuits to approximately one third, for the same statistical security goal. The technique is accomplished based on BitComs with trapdoor. The output of the protocol also includes reusable XOR-homomorphic (or pseudo-homomorphic) BitComs of all input and output circuit bits (but not of bits in the internal gates), thereby enabling efficient linkage of several S2PCs in a reactive manner. The analysis allows the parameters of the commitment schemes to be derived from different types of setup, e.g., a global public-key infrastructure (PKI) or a global common-reference string (CRS) (the later also possibly defined as a fixed protocol parameter). The protocol may also use a local-CRS for simulatability of NIZKs, NIZKPoKs and some internal commitments; the

setup is avoidable if allowing more interactivity and rewinding in the simulation.

The outer BitComs are *connected* to the input and output wire keys of the garbled circuits via structures that at a high level of abstraction are here called *connectors*. Different instantiations are possible, e.g., based on integer-factorization cryptography (IFC) (with a decisional quadratic-residuosity intractability assumption) or discrete-log cryptography (DLC) (with a decisional Diffie-Hellman intractability assumption). The outer bits can instead be committed with bit-string commitments with additive properties, more compactly than with one BitCom per bit, thus reducing the size of the final output state of each party. The outer commitments are randomized with the help of permutations decided via a simulatable coin-flipping, itself based on an extractable-and-equivocable commitment scheme.

The coin-flipping is asymmetric, as for each type of wire one party only learns the permutations, whereas the other party also learns the randomness used to produce them. Simulatability of this coin-flipping involves one party also committing to a non-interactive zero knowledge proof of knowledge of the randomness. Oblivious transfers (OTs) for the input bits of the circuit evaluator are described at the level of BitComs, instead of at the level of input wire keys of garbled circuits, and with implementations possible based on 2-out-of-1 OTs and 1-out-of-2 OTs. Based on a random oracle (which may be non-programmable), the protocol can be implemented in three communication steps, or two for simple S2PC (i.e., without commitments in the output) with only one party learning an output.

Organization. Section 1.3 has provided an introduction and Chapter 2 reviewed some background and related work. This Chapter includes remaining material. Section 3.1 identifies the protocol stages associated with the cut-and-choose structure and with the BitCom approach, introduces the idea of connectors, explains the forge-and-lose technique and analyzes the improvement in statistical security and/or reduction of number of garbled circuits. Section 3.2 gives a high level description of the new S2PC-with-Coms protocol, describing each logical stage. Section 3.3 explains the *connectors* that sustain the BitCom approach, based on XOR-homomorphic properties, showing how BitComs or BitStringComs can be *connected* to circuit input and output wire keys, to ensure the consistency of the keys across different garbled circuits. Section 3.4 analyzes the complexity of the protocol under several instantiations of primitives and cut-and-choose configurations. Section 3.5 initiates the security analysis of the protocol and mentions the possibility of linking several S2PC executions. Section forgelose:sec:more-related-work comments on developments to

the forge-and-lose technique, comparing the original technique vs. the improvements in this dissertation and other related work. Appendix A specifies the needed NIZKPs and NIZKPoKs. Appendix B describes the S2PC-with-BitComs protocol in low level, including some optimizations, and complements the security analysis of the protocol.

3.1 Overview of the protocol

This section overviews the main elements of the S2PC-with-Coms protocol. First, it describes a logical modularization of the protocol stages (§3.1.1), based on a cut-and-choose approach, and integrating BitComs and a coin-flipping stage. Then it introduces the notion of connectors, combining a BitCom setting (where there is a BitCom for each circuit input and output bit) and the cut-and-choose structure (where there are several garbled circuits, each with two keys for each input and output wire) (§3.1.2). The forge-and-lose technique is then described, using properties of different BitCom schemes to enable a new criterion of successful evaluation (§3.1.3). This motivates a comparison of error probabilities (Table 3.1) between the cut-and-choose with forge-and-lose vs. that of a cut-and-choose requiring a majority of correct evaluation garbled circuits, and a calculation of number of garbled circuits (Table 3.2) needed to achieve certain levels of statistical security (§3.1.4).

3.1.1 Protocol stages

The S2PC-with-BitComs protocol described in this dissertation is built on top of a C&C approach with a Commit-Challenge-Respond-Verify-Evaluate logical structure. Given the goal of outputting commitments of the input and output, the logical structure also includes: stages (Produce initial BitComs of P_A and Produce initial BitComs of P_B) where the parties produce commitments of their circuit input bits and prepare other commitments (e.g., of bit-masks and bit-offsets) that will help with the commitment of the output bits. These initial commitments happen before the stages associated with the cut-and-choose structure. In the end, in a Permute Outer Coms stage the initial (outer) commitments are permuted into final random outer commitments. The actual random permutations are decided in a two-party Coin-flip Permutations stage, with three communication steps that can be interleaved and merged across the other stages. Enclosing the protocol, there is an initial Setup stage, where the needed BitCom scheme parameters are defined, and

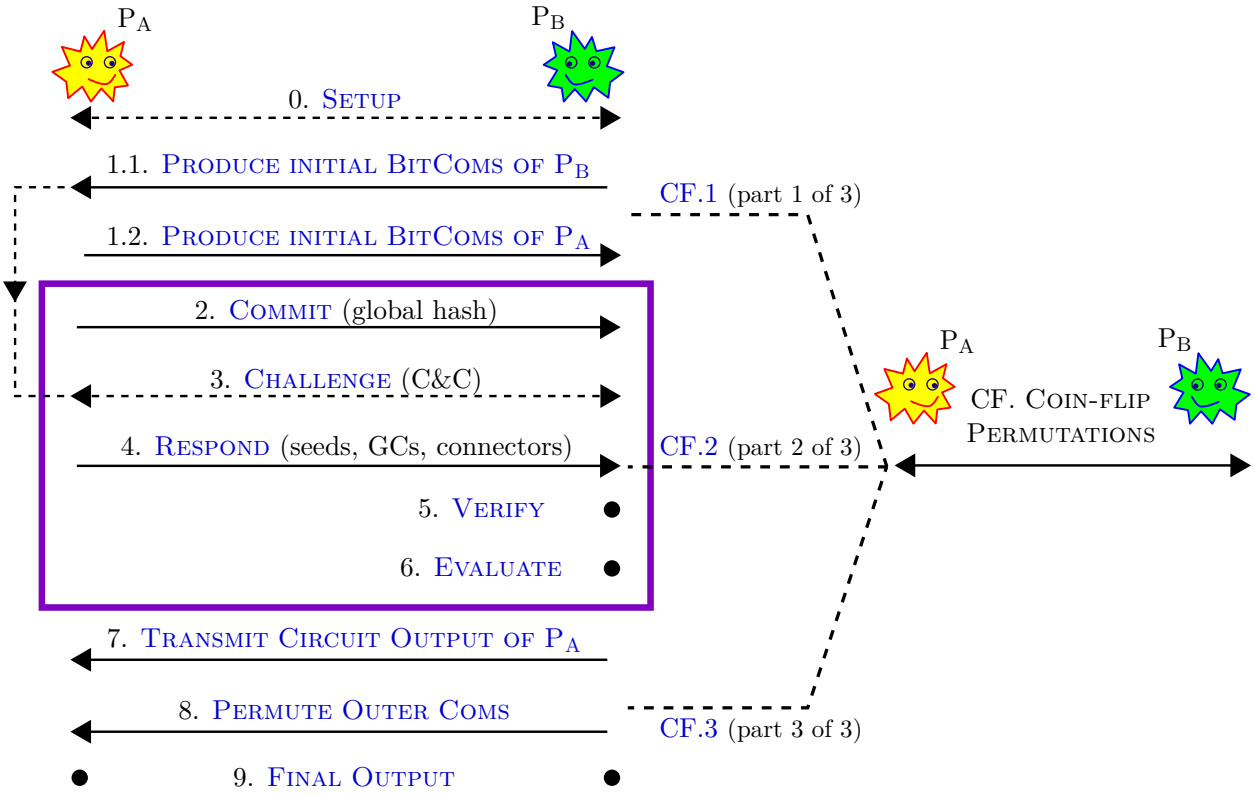


Illustration 3.1: Logical Stages of the S2PC-with-BitComs protocol. Legend: → (message from P_A to P_B); ← (message from P_B to P_A); ↔ (interaction between P_A and P_B, possibly with just one message in a single direction, possibly with messages in both directions); • (local activity at the respective party); ⇢ (explicit interleaving of the coin-flip stage across different stages of the cut-and-choose structure); ⇣ (possible definition of the lower stage based on information exchanged in the upper stage); C&C (cut-and-choose partition); GCs (garbled circuits). The Coin-flip Permutations stage (CF), is interleaved between stage 1.1 (Produce initial BitComs of P_B) and stage 8 (Permute Outer Coms). The actual message exchange may merge several stages, e.g., to reduce the number of communication rounds (see Remark 3.1).

a Final Output stage where each party structures the respective final private output. Illustration 3.1 depicts the sequence of logical stages, and then follows a high level description.

Stages related to the cut-and-choose structure.

In the Commit stage, P_A commits to several instances (some of which will later be selected for *check* and the remaining for *evaluation*). Each instance is composed of one garbled circuit and complementary elements (dubbed *connectors*) related with BitComs and with the circuit input and output wire keys of garbled circuits. Based on a random seed checking (RSC) technique, all "randomness" needed for each instance is actually pseudo-randomly generated from a short random seed. After computing the elements for all instances, P_A aggregates all

elements and computes a respective collision-resistant hash, denoted *global hash*, and sends to P_B a respective short Equiv-Com of the hash (denoted RSC commitment).

The Challenge stage determines a partition of the set of instances into two subsets: *check* and *evaluation*. Possibly, the subsets may be conditioned to a predefined restriction about their sizes (e.g., a fixed proportion of *check* vs. *evaluation* instances, or not letting the number of *evaluation* instances exceed some proportion. The partition must not be known to P_A before the end of the Commit stage. Apart from that, the decision can be performed in several distinctive structural ways (see §2.4.2). It may be decided after the Commit stage, (i) arbitrarily by P_B and sent to P_A; or (ii) interactively via a coin-flipping between P_B and P_A; or (iii) by P_A based on a non-programmable random oracle (assuming an appropriate adjustment to statistical security) and sent along with the communication of stages 2 (Commit) and 4 (Respond). It may also be decided sooner, committed by P_B in stage 1.1 (Produce initial BitComs of P_B), and never revealed to P_A, but nonetheless allowing P_B to provide (via oblivious transfer) a respective response in stage 4 (Respond).

In the subsequent Respond stage, P_A sends to P_B the elements that allow P_B to *fully verify* the correctness of the *check* garbled circuits and connectors, and to *partially verify* the *evaluation* connectors, and to *evaluate* the *evaluation* garbled circuits (and respective connectors). Specifically: for each *check* instance, P_A sends only the respective RSC seeds; for each *evaluation* instance, P_A does not send the seed, but sends several of the respective derivable elements; also, P_B opens the global hash from the RSC commitment.

In the Verify stage, P_B uses the received elements to generate a candidate pre-image of the global hash and verify that it is consistent with the hash value opened from the RSC commitment. P_B also verifies that some additional elements received for evaluation instances satisfy some homomorphic properties. If any verification step fails, then P_B aborts the protocol execution; otherwise, P_B establishes that there is an overwhelming probability that at least one evaluation garbled circuit (and respective connectors) is correct.

Finally, P_B proceeds to an Evaluate stage, evaluating the *evaluation* GCs and respective connectors, and using their results to determine the final circuit output bits and respective openings of output BitComs. It is worth noticing that between the Verify and Evaluate stages there is no further *response* stage that could let P_A misbehave, i.e., the two stages are locally and consecutively executed by P_B, without further interaction with P_A. (This is a fundamental difference in comparison with [Lin13], where the "secure evaluation of cheating"

stage requires an interaction after the initial evaluation stage.)

Stages related to the BitCom approach.

Complementary to the C&C structure, the S2PC-with-Coms protocol also considers the integration of commitments and BitComs as part of the final output. While in theory the Boolean Circuit could directly compute also include BitComs, in practice that would introduce an unacceptable overhead in the protocol, so the commitments are instead computed in a different layer. In a SETUP stage, the needed BitCom schemes are defined, some of which may have an associated bit-string Com scheme. The parameters of some schemes may be established as part of a trusted setup (e.g., public-key infrastructure or common reference string) and others may be selected by each party and sent and proved correct to the other party. Some NIZKPs are used to ensure correctness and some NIZKPoKs more generally to promote simulatability. In initial stages (PRODUCE INITIAL BITCOMS OF P_A and PRODUCE INITIAL BITCOMS OF P_B) the parties commit their own input bits, and prepare BitComs of offset bits for the output. Some Coms are denoted as *outer*, as they are directly related to the output of the S2PC-with-Coms protocol. Other Coms and BitComs are denoted as *intermediate*, as they are used to support the *connectors* that create a verifiable connection with the garbled circuit wire keys, but are not part of the final output of a protocol execution. The *outer* Coms produced in this stage are denoted as *initial*, because they will later need to be randomized, into *final* outer Coms, so that the protocol emulates an ideal S2PC-with-Coms functionality $\mathcal{F}_{\text{S2PCwC}}$ that would produce random BitComs.

The randomization of the outer Coms is achieved by homomorphically permuting the initial outer-Coms into final outer-Coms. The random permutations for wires of P_A are decided directly by P_B, and committed (using an Ext-and-Equiv Com scheme) even before P_A produces her outer-Coms. The random permutations for wires of P_B are instead defined via a (two-side-)simulatable two-party coin-flipping protocol, because they can only be decided after P_B has already defined the initial outer-Coms. For simulatability reasons, the permutations are decided in a way that, for each type of wire only the "owner" of the wire learns the outer-randomness permutation, whereas the other party only learns the respective outer-Com permutation. Also for simulatability reasons, the contribution of outer-Com permutations proposed by each party needs to be accompanied by a respective NIZKPoK of opening. For the wires of P_A this is merged into the already needed NIZKPoK of outer-Coms, but for the wires of P_B the NIZKPoK transcript needs to be initially committed along with the

contribution of P_B. Actually, depending on the instantiation (e.g., for GM BitComs, but not for ElGamal BitComs), the NIZKPoK may be avoided if extraction is already enabled via an extracted trapdoor obtained after a NIZKPoK of trapdoor. If the NIZKPoK of openings is explicitly required, then P_B commits to it along with his permutations contribution, within the coin-flipping protocol. P_B also sends a NIZKP that his contributions to the outer-com permutations correspond indeed to commitments to all-zeros bit-strings. In the end of the protocol, in a PERMUTE OUTER COMS stage, the parties use the permutations to homomorphically calculate the final outer Coms and respective randomnesses.

Assuming that the coin-flipping is implemented with the traditional template structure, then the first step (P_B committing to a contribution) can be performed early in the protocol, when P_B produces his initial BitComs; the second step (P_A sending a random contribution) can be performed along with the RESPOND stage, where P_A sends the evaluation garbled circuits to P_B. In the final message of the coin-flipping, P_B opens his contribution so that P_A also learns the coin-flipping outcome. The outer Coms and respective coin-flipping of permutations may be ignored if considering only a simple S2PC protocol that does not intend to produce commitments as part of the output.

Remark 3.1 (On merging and interleaving logical stages). As mentioned in the legend of Illustration 3.1, several logical stages may be merged and/or interleaved. Any interleaving of the coin-flipping needs to ensure that each party is not able to affect a final honest output, namely the final random value of outer Coms. The communication needed to define the parameters of the BitCom and Com schemes of P_B in stage 0 (SETUP) may be performed along with stage 1.1 (PRODUCE INITIAL BITCOMS OF P_B), when committing the circuit input bits of P_B and preparing the oblivious transfer technique. The communication of stage 1.2 (PRODUCE INITIAL BITCOMS OF P_A), associated with committing the circuit input bits of P_A and producing other BitComs that support the forge-and-lose technique, may be combined with the communication of stage 2 (COMMIT). Stage 3 (CHALLENGE) can be performed in different ways: either after stage 2 and revealed to P_A, or in stage 1.1 (PRODUCE INITIAL BITCOMS OF P_B) but not revealed to P_A, at least till it finishes stage 4 (RESPOND). The stages CF (COIN-FLIP PERMUTATIONS) and 8 (PERMUTE OUTER COMS) can be ignored in a simple S2PC that does not contain Coms as part of the output.

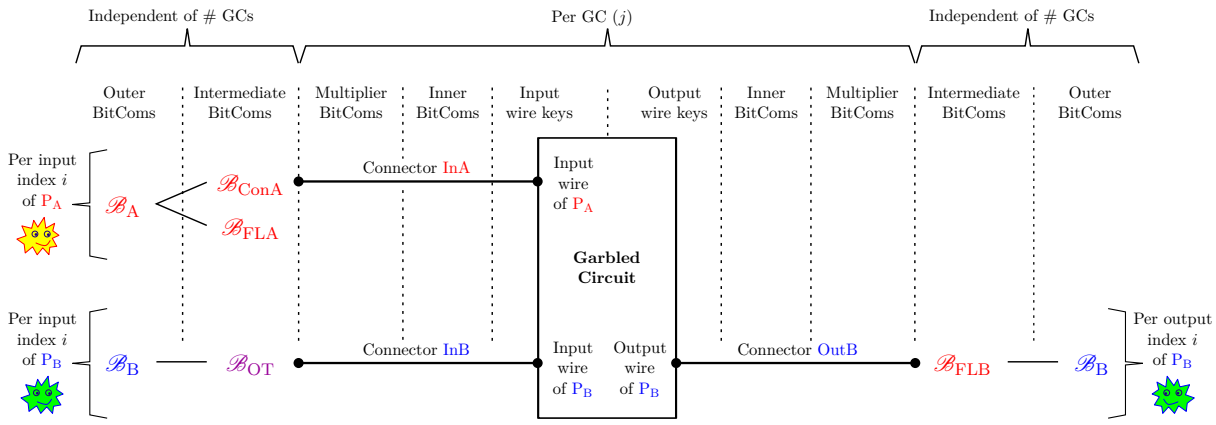


Illustration 3.2: Interface between outer BitComs, connectors and wire keys. The outer BitCom schemes ($\mathscr{B}_A$, $\mathscr{B}_B$) are used to produce the BitComs that are part of the final output of the S2PC-with-Coms. The intermediate BitCom scheme $\mathscr{B}_{OT}$ related to input bits of P_B (used for oblivious transfers) is depicted with purple color because its trapdoor might be known either by P_A or by P_B, depending on the instantiation. The intermediate BitCom scheme $\mathscr{B}_{FLB}$ related to output bits of P_B (used to sustain the forge-and-lose technique) is depicted in red because its trapdoor is selected by P_A. Out of the two intermediate BitCom schemes related to input bits of P_A, one ($\mathscr{B}_{ConA}$) is related to the connectors, whereas the other ($\mathscr{B}_{FLA}$, dual of $\mathscr{B}_{FLB}$) is used to sustain the forge-and-lose technique.

3.1.2 Connectors

In the new protocol, the integration between *outer* Coms and the C&C structure is sustained by structures called *connectors*. First, for each type of connector, the pre-condition is that an *intermediate* BitCom has already been produced for the respective (input or output) wire index (this is done in the Produce initial BitComs of P_A or Produce initial BitComs of P_B stage), independently of the number of GCs. The *intermediate* BitCom is the interface between the *outer* BitCom (related to the final BitComs that each party will output as part of S2PC-with-BitComs) and the *internal* part of the connector. All connectors associated with the same wire index (i.e., across all garbled circuits) have the same intermediate BitCom, but the internal part of each connector is connected to a wire of a different garbled circuit. Illustration 3.2 depicts this relation at high level, also showing that each connector is also composed of internal BitComs (called multipliers and inner BitComs) within the interface with the keys of the respective wire of the respective garbled circuit.

Connectors provide a (statistically verifiable) connection between the pair of keys of the

respective wire and the openings of the respective *intermediate* BitComs. In this approach, based on XOR-homomorphism of BitComs, the consistency of input and output wire keys across different GCs is *statistically* ensured within the C&C, rather than using a ZKP of consistency. (The overall protocol still includes other (efficient) ZKPs and ZKPoKs related with BitComs, but they are not about the consistency of wire keys across different GCs.)

Connectors are used in a way somewhat similar to a commitment scheme, i.e., with *commit* and *open* phases, but different by taking advantage of the C&C substrate when committing to several elements that will be opened in different ways. Each connector (and also the GC) is committed in the C&C COMMIT stage, hiding the respective two wire keys, but binding P_A to them and to their relation with the (also hidden) encodings of the respective outer BitCom. Then, after the C&C partition is determined, all garbled circuits are revealed, and all connectors are *partially revealed* during the C&C RESPOND stage, in one of two possible complementary modes: a *reveal for check*, related with *check* instances; or a *reveal for evaluation*, related with *evaluation* instances. All verifications associated with these two reveal modes are performed in the C&C VERIFY stage, when P_B can still, immune to selective failure attacks, complain and abort in case it finds something wrong. P_A never executes simultaneously the two reveal modes for the same challenge index, because such action would reveal the input bits of P_A, as well as (as explained ahead) two encodings of intermediate BitComs of the output of P_B (constituting the trapdoor of P_A in the forge-and-lose technique).

Even though each connector is only partially revealed, the commitment to the *connector* still binds P_A to the answers that it can give in each type of reveal phase. Thus, an incorrect connector can pass undetectably at most through one type of reveal mode. This means that, within the C&C approach, there is a negligible probability that P_A builds *bad* connectors for all evaluation instances and goes by undetected. The internal functioning of connectors varies with the type of wire they refer to (input of P_A, input of P_B, output of P_B), as illustrated still at high level in Illustration 3.3.

Types of elements associated with connectors and the C&C structure. For conceptual simplicity, the involved elements are distinguished in three types:

- **Type-C ("commit").** Those that commit to the function being securely computed (i.e., the GC) and to connector elements.
- **Type-RC ("reveal for check").** Those that allow verification of the Type-C elements,

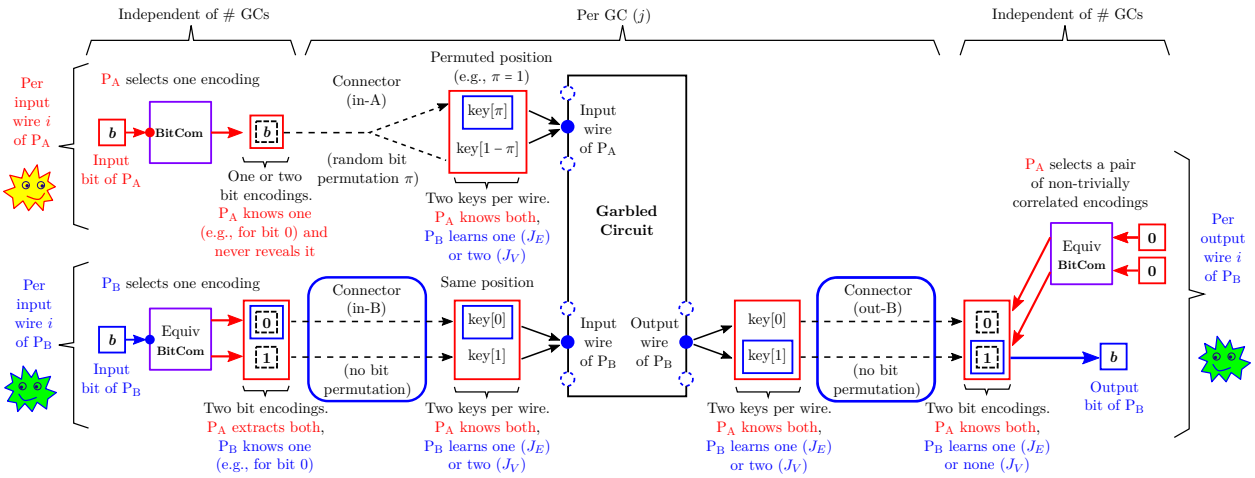


Illustration 3.3: Connectors. Legend: P_A (GC *constructor*); P_B (GC *evaluator*); Equiv (equivocable); in-A, in-B, out-B (input of P_A, input of P_B, output of P_B). $\boxed{c}$ (group-element encoding bit c); key$[c]$ (wire key with underlying bit c). The example here given for the connector of input of P_B is based on a 2-out-of-1 OT (as in the original paper), but this chapter also discusses later a construction based on a 1-out-of-2 OT. For simplicity, all BitCom schemes may be assumed to be XOR-homomorphic and with trapdoor (even though some specialized variations may exist).

but do not allow determination of the final output of P_B. All of these can (for each challenge index) be derived from a random short seed.

- **Type-RE ("reveal for evaluation").** Those (not derivable from a random seed) that, in complement to the Type-C elements, finalize a verifiable connection between the BitCom openings and the wire keys, allowing P_B to obtain one key per input wire and one bit and BitCom opening per output wire.

Without a RSC technique, the disclosure of Type-C, Type-RC and Type-RE elements would be respectively done in the COMMIT stage, the *reveal-for-check* part of the RESPOND stage, and the *reveal-for-check* part of the RESPOND stage. Instead, the use of RSC eliminates the need to communicate Type-RE elements, and reduces the number of Type-C elements to just those related to *evaluation* instances. In the COMMIT stage, P_B only receives a short RSC commitment, instead of the actual Type-C elements. Later, in the RESPOND stage, the Type-C elements are provided to P_B either by P_A sending the respective short RSC seed (for *check* indices), or by P_A directly revealing the elements (for *evaluation* indices). The Type-RC elements are also pseudo-randomly derivable from the short seed obtained in the *reveal for check* — in fact, they are intermediate values required to generate the Type-C elements. The Type-RE elements are directly revealed (along with Type-C elements) in the

reveal for evaluation, only for evaluation instances.

3.1.3 The forge-and-lose technique

The new protocol brings a statistical improvement over the typical C&C-GCs-based approach, by using the BitCom approach to provide a new path for successful computation of final circuit output. The essence is a new technique called "forge-and-lose" — a name designed to rhyme with "cut-and-choose" and suggestively indicating that an attempt by a malicious P_A (the garbled circuit constructor) to *forge* elements in the cut-and-choose will lead P_A to *lose* her privacy. More precisely, if in the EVALUATE stage there is at least one garbled circuit and respective connectors leading to a correct output (i.e., valid openings of the BitComs, for the *correct* circuit output bits), and if a malicious P_A^* successfully *forges* some output associated with another garbled circuit and connectors (i.e., valid openings of the BitComs, for *incorrect* circuit output bits), then P_A^* *loses* the privacy of her input bits to P_B, allowing P_B to directly use a Boolean circuit to compute the intended output. This loss of privacy is not a violation of security, but rather a disincentive against malicious behavior by P_A^*.

The forge-and-lose technique is illustrated at high level in Illustration 1.1. It can be logically considered in two parts: in regard to input bits, it involves an Ext-Com of the input of P_A, which can be extracted with an appropriate trapdoor kept secret from P_B; in regard to output bits, it involves an evaluation path that, in case of P_A forging some elements selected for evaluation, allows P_B to discover the secret trapdoor. The next two paragraphs give a brief description, still at high level.

Encrypt input of P_A. As part of the BitCom approach, P_A encrypts/commits her own input bits. This is done with an Ext-Commit scheme with key/trapdoor equal to the trapdoor (known by P_A) of the Equiv-BitCom scheme used (by P_A) to prepare BitComs of the output bits of P_B. The Equiv-Com scheme is such that the knowledge of a pair of openings (i.e., for two different bits) of any BitCom is equivalent to the knowledge of the trapdoor. For example, in a proposed instantiation based on Blum integers, the trapdoor is the integer factorization of a Blum integer, each input bit of P_A is committed with a GM BitCom (Ext), and each output bit of P_B is committed with a Blum BitCom (Equiv). For a DLC instantiation, the schemes can be based on ElGamal BitComs and Pedersen BitComs, respectively.

Forge-and-lose evaluation. In the EVALUATE stage, if a *connector* leads an output wire key to an invalid opening, then P_B ignores the respective instance. If in all evaluation instances for which all connectors lead to valid BitCom openings those openings are consistent across all such instances, i.e., if for each output wire index the same valid BitCom opening is obtained across all evaluated garbled circuits, then P_B accepts them as correct. However, if P_A acted maliciously, there may exist a forged garbled circuit and connector leading to a valid BitCom opening that is different from the BitCom opening obtained from another correct garbled circuit and connector, for the same output wire index. If P_B obtains any such pair of openings, i.e., a non-trivially correlated pair of openings, then P_B gets the trapdoor with which P_A encrypted her input. Then, P_B uses the trapdoor to decrypt the input bits of P_A and uses them directly to evaluate the intended Boolean circuit.

Remark 3.2 (Similarity with double spending). It is interesting to notice a conceptual similarity between the forge-and-lose technique with respect to S2PC and the *double-spending* prevention mechanism devised by Chaum, Fiat and Naor with respect to electronic cash [CFN90]. In the later, the technique provides a deterrent against P_A illegitimately reusing cash — the reuse of the same electronic coin in two different payments implies that her identity can be found from the combined information of the two respective transactions (eventually received by a central authority — a bank), where otherwise the identity would remain private. In the forge-and-lose application the goal is not to enable identification of the malicious party across different executions of a protocol, but rather to enable a successful computation in case of attempted malicious behavior. Actually, a respectively identified malicious behavior remains non-complainable by the honest party P_B, lest it would break the privacy of his input in case of a selective failure attack by P_A.

Remark 3.3 (Alternatives without trapdoor commitments). The forge-and-lose was defined (and is used hereafter) based on an explicit trapdoor that allows extraction of any previously committed bits. There are conceivable alternatives to the building blocks of the forge-and-lose approach, namely without using trapdoor commitments, and still retaining the non-interactive flavor (i.e., not requiring an additional phase after evaluation of garbled circuits). Indeed, the main idea is that any pair of valid-in-isolation but pairwise-inconsistent outputs, i.e., from the same wire index across two different garbled circuits, allows P_B to recover the input of P_B. For example, the combination of correct output wire keys for two

different bits, associated with the same output wire index, could conceivably lead P_B to directly learn the randomness used to initially commit the bits of P_A, even though without any trapdoor existing for the BitCom schemes. An actual approach without trapdoor commitments, and involving an augmentation of the underlying Boolean circuit computed by the S2PC, was devised by Frederiksen et al. [FJN14a] (see further comment in §3.6.2). In this dissertation, the use of Equiv-BitComs is not only compact in terms of communication size (at least in an ECC instantiation), but also related with the broader goal of achieving S2PC-with-Coms, instead of simple S2PC.

3.1.4 Better statistical security and/or fewer garbled circuits

This subsection considers the soundness error probability and the number of garbled circuits for diverse statistical security goals and diverse cut-and-choose configurations. The calculations assume that a malicious P_A uses the optimal adversarial strategy to lead P_B to accept an incorrect output. The calculated probabilities are valid for C&C protocols (such as the one in this dissertation, and many others in ZK protocols) where a *bad* index (i.e., one in which P_A has cheated in the COMMIT stage) is detected if selected for *check*. Thus, for soundness to be broken, all the v instances selected for *check* must be *good*. Also, within the remaining e *evaluation* instances, the number b of *bad* instances must be enough to lead P_B to an incorrect result. This number depends on the C&C partitioning method (e.g., fixed vs. variable number of check instances) and the soundness requirement in terms of number of *good* evaluation indices (e.g., a majority in traditional protocols, vs. at least one in the forge-and-lose paradigm).

The error probability is considered in Table 3.1. The formula for error probability follows a simple counting argument. The total number of possible C&C choices is equal to the number of *check* subsets possible from within the set of all indices. Of these, the choices that lead to error are those for which all the *bad* elements are not selected for check and for which there are enough bad indices to induce error. The number of these choices is equal to the number of possible *check* subsets that can be selected from a set of *good* indices. The error probability is given by the quotient of the later quantity over the former quantity. Table 3.1 shows the error probabilities associated with different C&C methods, including (see column B) fixed and variable number of *check* and *evaluation* instances.

Table 3.1: Soundness error probability

A	B	C	D	E	F	G	
Correctness requirement	C&C partition method		b (# bad indices)	Error probability (Pr)		σ : Bits of statistical security ($-\log_2$Pr) (approximate)	1
	Mode	Restriction		Exact	Stirling's approximation		2
A majority of evaluation indices is good	Fixed	$\langle v, e\rangle$	$\lceil e/2\rceil$ (fixed)	$\frac{\text{Bin}(s-b,v)}{\text{Bin}(s,v)} = \frac{(s-b)!e!}{(e-b)!s!}$	...	...	3
		$v \approx s/2$		$\approx \frac{(3s/4)!(s/2)!}{(s/4)!s!}$	$2^{-(1+3s)/2} \cdot 3^{(2+3s)/4}$	$0.311s - 0.292$	4
		$v \approx 3s/5$		$\approx \frac{(4s/5)!(2s/5)!}{(s/5)!s!}$	$2^{3/2} \cdot 5^{-1/2} \cdot (5/4)^{-s}$	$0.322s - 0.339$	5
At least one evaluation index is good (e.g., with forge-and-lose)	Fixed	$\langle v, e\rangle \approx \langle \alpha s, \beta s\rangle$	e (fixed)	$e!v!/s!$	$\sqrt{2\pi\alpha\beta s}(\alpha^\alpha\beta^\beta)^s$	$c_1 s - (\log_2 s)/2 - c_0$	6
		$v = \lceil s/2\rceil$		$\lfloor s/2\rfloor!\lceil s/2\rceil!/s!$	$2^{-s}\sqrt{\pi \cdot s/2}$	$s - (\log_2 s)/2 - 0.326$	7
	Variable	$0 \le v < s$	e (variable)	$1/(2^s - 1) \approx 2^{-s}$		$\approx s$	8
		$0 < e \le v < s$		$\le 1/(2^{s-1} - 1) \approx 2^{-(s-1)}$		$\gtrapprox s - 1$	9
		$e_{\min} \le e \le e_{\max}$		$1/\left(\sum_{i=e_{\min},\ldots,e_{\max}}(i!(s-i!)/s!)\right)$		$\log_2\left(\sum_{i=e_{\min}}^{e_{\max}} \frac{s!}{i!(s-i)!}\right)$	10

Legend: # (number of), v (# *check* indices), e (# *evaluation* indices), s ($= v + e$, # challenge indices), b (# bad indices); γ (auxiliary parameter defining a limitation on the number of evaluation instances in relation to the number of bits of statistical security). By definition, $s = v + e$ and $\alpha + \beta = 1$, with $\alpha, \beta > 0$. $\text{Bin}(\cdot,\cdot)$ denotes the binomial coefficient, with $\text{Bin}(n, m) = n!/(m!(n-m)!)$. Stirling's first order approximation establishes $n! \approx \sqrt{2\pi n}(n/E)^n$, as n approaches infinite, where $E \approx 2.7828$ is Euler's constant (the basis of the natural logarithm), and $\pi \approx 3.1416$ is the quotient between the circumference and the diameter of a circle. $c_1 = \log_2(\alpha^{-\alpha}\beta^{-\beta})$; $c_0 = (\log_2(2\pi\alpha\beta))/2$.
Remark: The error probability in cell E7 would remain the same if in cell C7 the number of check instances would change from $v = \lfloor s/2\rfloor$ to $v = \lceil s/2\rceil$. Conversely, when s is odd the error probability in cell E4 between the cases $v = \lceil s/2\rceil$ and $v = \lfloor s/2\rfloor$ (cell C4), depending on the value $s \pmod 4$. For $s \pmod 4 = 3$, $v = \lceil s/2\rceil$ is better by 1 bit of statistical security, i.e., it yields half of the error probability. For $s \pmod 4 = 1$, $v = \lfloor s/2\rfloor$ can be better up to about 0.5 bits of statistical security.

Traditional cut-and-choose. In traditional cut-and-choose methods, where a majority of evaluation instances is required to be correct, an error may occur when the number b of *bad* circuits is at least half of the evaluation instances (cell D3–5). If the number of types of challenges is fixed in advance, then the probability of error is defined as a quotient that considers the number of bad challenges to be exactly the smallest integer greater than half of the number of evaluation instances, and measures the probability that all of them are indeed selected for evaluation (cell E3–5). If fixing the number of challenges to be half of each type, then the cut-and-choose achieves about 0.31 bits of statistical security (cell G4). However, the maximum statistical security per GC (about 0.32 bits [SS11]) is achieved with proportions of about three fifths of challenged selected for check and two fifths for evaluation (row 5). For 40 bits of statistical security, the concrete minimal number of garbled circuits is 123 [Bra13], with 74 for check and 49 for evaluation (see row 4 in Table 3.2), whereas previously proposed configurations suggested 125 GCs [SS11].

Table 3.2: Number of garbled circuits to achieve statistical security

A	B	C	D	E	F	G	H	
Correctness requirement	C&C partition method		Security goal (σ) / Configuration	40	80	96	128	1
	Mode	Restriction						
Majority of evaluation indices is good	Fixed	$v = \lceil s/2 \rceil$ (half-half)	(s, v, e)	(129, 64, 65)	(257, 128, 129)	(309, 154, 155)	(410, 205, 205)	2
			Bits security	40.542	80.390	96.577	128.123	3
		Minimize s, then minimize e ($e \approx 2s/5$)	(s, v, e)	(**123**, 74, **49**)	(247, 150, 97)	(297, 178, 119)	(396, 239, 157)	4
			Bits security	40.257	80.174	96.271	128.144	5
At least one evaluation index is good (e.g., with forge-and-lose)	Variable	$0 < e < s$ (independent)	(s, v, e)	(**40**, $s-e$, **40**)	(80, $s-e$, 80)	(96, $s-e$, 96)	(128, $s-e$,128)	6
			Bits security	40.000	80.000	96.000	128.000	7
		$0 < e \leq s/2$ (more v than e)	$(s, v_{\min}, e_{\max})$	(**41**, 21, **20**)	(81, 41, 40)	(97, 49, 48)	(129, 65, 64)	8
			Bits security	40.000	80.000	96.000	128.000	9
	Fixed	$e = \lfloor s/2 \rfloor$ (minimize s, then maximize σ)	(s, v, e)	(44, 22, 22)	(84, 42, 42)	(100, 50, 50)	(132, 66, 66)	10
			Bits security	40.936	80.474	96.349	128.149	11
		$e = \lfloor s/2 \rfloor$ (minimize s, then minimize e)	(s, v, e)	(44, 25, 19)	(84, 45, 39)	(100, 53, 47)	(132, 68, 64)	12
			Bits security	40.358	80.168	96.091	128.063	13
	Variable	$0 < e \leq e_{\max} \wedge e_{\max} \leq \sigma/4$	$(s, v_{\min}, e_{\max})$	(76, 66, 10)	(142, 122, 20)	(168, 144, 24)	(220, 188, 32)	14
			Bits security	40.024	80.145	96.147	128.186	15
	Fixed	$e = \lfloor \sigma/4 \rfloor$	(s, v, e)	(77, 67, 10)	(143, 123, 20)	(169, 145, 24)	(221, 189, 32)	16
			Bits security	39.997	80.110	96.110	128.146	17
	Variable	$0 < e \leq e_{\max} \wedge e_{\max} \leq \sigma/5$	$(s, v_{\min}, e_{\max})$	(**123**, 115, **8**)	(225, 209, 16)	(272, 253, 19)	(365, 340, 25)	18
			Bits security	40.008	80.096	96.090	128.009	19
		$0 < e \leq e_{\max} \wedge e_{\max} \leq \sigma/6$	$(s, v_{\min}, e_{\max})$	(306, 300, 6)	(409, 396, 13)	(442, 426, 16)	(603, 582, 21)	20
			Bits security	40.010	80.021	96.015	128.031	21
		$0 < e \leq e_{\max} \wedge e_{\max} \leq \sigma/8$	$(s, v_{\min}, e_{\max})$	(668, 663, 5)	(1163, 1153, 10)	(1359, 1347, 12)	(1748, 1732, 16)	22
			Bits security	40.001	80.002	96.007	128.008	23

Legend: # (number of), $s = e + v$ (# garbled circuits, i.e., size of cut-and-choose set), v (# *check* instances), e (# *evaluation* instances), The parameters in the even rows correspond to configurations that minimize s, conditioned to satisfying the statistical security goal (row 1) and the indicated restriction on e (column C).

Cut-and-choose with forge-and-lose. The forge-and-lose path reduces the probabilistic gap available for malicious behavior by P_A that might lead P_B to accept an incorrect output. The technique provides up to 1 bit of statistical security per GC, which is an improvement factor of about 3.1 (either in reduction of number of GCs or in increase of number of bits of statistical security) in comparison with C&C-GCs that require a majority of correct *evaluation* GCs. If the number of challenges of each type is fixed in advance (rows 6 and 7), then the error probability is now defined by a simpler quotient, equal to the inverse of the total number of partitions with said numbers of challenge types (cell E6). For example, if each challenge type is fixed to be about half of the number s of instances, then the number of bits

of statistical security (σ) is already almost equal to the number of garbled circuits, except for a logarithmic offset (cell G9).

In the forge-and-lose setting, the optimal C&C partition in terms of reducing the overall number of instances corresponds to an independent selection of check and evaluation challenges, which allows achieving 1 bit of statistical security for each garbled circuit cell G8. (An independent selection of challenges was also used in the contemporary "secure evaluation of cheating" method of Lindell [Lin13].) Still, for some efficiency tradeoffs it may be preferable to impose some restrictions (rows 9–10). For example, by requiring that there are fewer *evaluation* instances than *check* instances (cell B9), while still allowing them to be variable, only one extra GC is needed in comparison with the case of a uniform random selection. The added benefit is that when applying a RSC technique the communication associated with GCs is thus limited to half of the overall number of GCs, for the same error probability and statistical security. A side-by-side comparison of the number of garbled circuits needed to achieve diverse goals of statistical security is given in Table 3.2 — e.g., compare row 6 vs. row 8.

More generally, the probability of error (and respective statistical security) can be calculated for any allowed variation of the number of evaluation circuits (row 10). For example, if useful to further reduce the number of evaluation instances (e.g., when using the RSC technique), even though at the cost of increasing the overall number of garbled circuits, it is possible to decide a restriction as a maximum proportion of number of evaluation instances in respect to the number of bits of statistical security. Table 3.2 exemplifies concrete configurations for a progressive reduction of the number of evaluation instances, namely one-fourth (rows 14–17), one-fifth (rows 18–19), one-sixth (rows 20–21) and one-eight (rows 22–23) of the number of bits of statistical security. Using a fixed number of evaluation instances may require a larger overall number of garbled circuits, but only by a very small amount, e.g., 3, or 1, respectively for the half-half and one-fourth configurations (compare row 8 vs. row 10, and row 14 vs. row 16).

A more refined cost model of the cut-and-choose approach is considered in [ZHKS16], considering equilibrium strategies that take into account the relative costs of each type of challenge (check vs. evaluation), including its dependence with the implementation configuration and the circuit being evaluated.

3.2 High-level description of protocol

This subsection gives a high level description of the new protocol for S2PC-with-Coms, based on a cut-and-choose of garbled circuits and using a forge-and-lose technique. The description follows the sequence of logical stages already depicted in Illustration 3.1 and mentioned in §3.1.1. (Instantiations of ZKPs and ZKPoKs related to BitComs and BitCom schemes are specified in Appendix A. The actual low level protocol specification is given in Appendix B.)

Stage 0. SETUP**.** The parties agree on a Boolean circuit specification, identifying the sets of input and output bits of each party, including a differentiation between the set of private output wires of each party and the set of common output wires. (Some of these sets might be empty.) The circuit is implicitly adapted so that the output bits bits of P_A are XOR-masked with mask-bits that are provided by P_A as additional random input bits.

The parties agree on: the security parameters; the cut-and-choose partitioning mode and parameters; the needed cryptographic primitives (e.g., PRG, CR-Hash, garbling scheme). The parties also agree on XOR pseudo-homomorphic BitCom schemes (defined via a global trusted setup) for the outer BitComs and Coms of the bits of each party, and also for the intermediate BitComs that sustain the connectors of the respective types of wires, as follows:

- BitCom schemes ($\mathscr{B}_A$, $\mathscr{B}_B$) for the outer BitComs of the two parties (P_A, P_B), and a respective generalization for BitStringCom schemes ($\mathscr{C}_A$, $\mathscr{C}_B$).
- A BitCom scheme ($\mathscr{B}_{ConA}$) to sustain the connectors of the input of P_A, (including for committing a bit-mask for each output bit of P_A).
- A BitCom scheme ($\mathscr{B}_{OT}$) to sustain the oblivious transfer needed for the connectors of the input of P_B. Depending on the type of OT, the BitCom scheme may either have an equivocation trapdoor explicitly known by P_A (in the case of 2-out-of-1 OT), or an extraction trapdoor explicitly known by P_B (in the case of 1-out-of-2 OT).
- A pair of *dual* BitCom schemes ($\mathscr{B}_{FLA}$, $\mathscr{B}_{FLB}$) to sustain the BitCom-based forge-and-lose technique, both of which with the same trapdoor explicitly known by P_B. Specifically, an Ext-BitCom scheme ($\mathscr{B}_{FLA}$) is used by P_A to commit the input bits of P_A (and the bits that will mask the output bits of P_A), and an Equiv-BitCom scheme ($\mathscr{B}_{FLB}$) is used also by P_A to initially prepare intermediate BitComs for output bits of P_B and for masked output bits of P_A. The trapdoor of the BitCom scheme ($\mathscr{B}_{FLA}$) that sustains the forge-and-lose technique in respect to the input bits of P_A is derivable from any pair of openings of

Table 3.3: Examples of BitCom scheme instantiations

A	B	C	D	E	F	
BitCom schemes \ Trusted Setup		GPKI		GCRS		1
		IFC	DLC	IFC	DLC	2
Outer	$\mathscr{B}_{\mathrm{A}}$ (Ext)	GM_{A}	$\mathrm{ElG}_{\mathrm{A}}$	GM_0	ElG_0	3
Intermediate	$\mathscr{B}_{\mathrm{ConA}}$ (Ext or Equiv)					4
	$\mathscr{B}_{\mathrm{FLA}}$ (Ext)			GM_{A}	$\mathrm{ElG}_{\mathrm{A}}$	5
Outer	$\mathscr{B}_{\mathrm{B}}$ (Ext)	GM_{B}	$\mathrm{ElG}_{\mathrm{B}}$	GM_0	ElG_0	6
Intermediate	$\mathscr{B}_{\mathrm{OT}}$ (Ext or Equiv)	$\mathrm{Blum}_{\mathrm{A}}$		$\mathrm{Blum}_{\mathrm{A}}$	$\mathrm{ElG}_{\mathrm{B}}$	7
	$\mathscr{B}_{\mathrm{FLB}}$ (Equiv)		$\mathrm{Ped}_{\mathrm{A}}$		$\mathrm{Ped}_{\mathrm{A}}$	8

Legend: GPKI (global public-key infrastructure); GCRS (global common-reference string); IFC (integer-factorization cryptography); DLC (discrete-log cryptography); GM, ElG (Goldwasser-Micali and ElGamal extractable BitCom schemes); Blum, Ped (Blum and Pedersen equivocable BitCom schemes). A, B, 0 (indices denoting who knows the respective trapdoor, respectively: P_{A}, P_{B}, no one). $\mathscr{B}_{\mathrm{FLA}}$ (row 5) and $\mathscr{B}_{\mathrm{FLB}}$ (row 8) are dual, i.e., have the same trapdoor and are respectively Ext and Equiv. Different instantiations are possible, e.g., $\mathscr{B}_{\mathrm{ConA}}$ in row 4 could be different from both $\mathscr{B}_{\mathrm{A}}$ and $\mathscr{B}_{\mathrm{FLA}}$ (e.g., see an optimization in §B.4.1.3 where $\mathscr{B}_{\mathrm{ConA}}$ must be unconditionally hiding). As mentioned in §2.2.2, each global trusted setup used to determine the parameters of the outer BitCom schemes is accompanied with a local trusted setup to enable proper simulations.

different bits associated with a same BitCom produced with the BitCom scheme ($\mathscr{B}_{\mathrm{FLB}}$) that sustains the forge-and-lose technique in respect to the output bits of P_{B}.

Some of the mentioned BitCom schemes may be the same and/or have the same public parameters, depending on the type of trusted setup, and on some implementation choices and possible optimizations. Example instantiations of BitCom schemes are presented in Table 3.3, across types of Trusted Setup and type of intractability assumption. For example, if assuming a global-PKI trusted setup, then the parties are automatically endowed with BitCom schemes with explicit private trapdoors, and all the mentioned BitCom schemes can be based thereon. If instead there is, for example, a global-CRS trusted-assumption, then both parties can have the same outer BitComs scheme, with unknown trapdoor, and in that case the parties need to bootstrap some of the other BitCom schemes with explicitly known trapdoor, e.g., for the forge-and-lose and the oblivious transfer techniques.

In case of a GPKI setup, each party gives a ZKPoK of the respective trapdoor, in order to allow the simulator in the role of the other party in a simulated execution to extract the

trapdoor so that it can subsequently, more easily, extract the committed bits and (in case of IFC) the respective randomness. This ZPKoK does not apply in the case of a GCRS setup because then there is no explicitly known trapdoor.

Remark 3.4 (On the need of ZKPs and ZKPoKs). The protocol needs some ZKPs and ZKPoKs in regard to the chosen BitCom schemes and/or the committed bits and/or the randomness used in commitments. Some ZKPs may be needed to ensure correctness, for example to guarantee that newly selected BitCom schemes have correct parameters (e.g., correct Blum integers) and/or to guarantee that BitComs are correct (actual elements from the space of possible BitComs) and/or to guarantee that BitComs supposed to be committing to the same bits are indeed committing to the same bits. The need for these ZKPs may change with the type of BitCom scheme. For example: a ZKP of correctness of BitComs may be avoidable for BitComs for which it is possible without the trapdoor to verify correctness in isolation (e.g., GM BitComs); a ZKP of correctness of BitCom schemes may also be avoidable if the schemes are verifiable without the trapdoor (e.g., in case of some ElGamal BitCom scheme parameters, that an element is a group generator). Avoiding ZKPoKs requires a more subtle argument. In spite of correctness, some ZKPoKs may still be needed to ensure simulatability, e.g., to give the simulator the ability to extract a trapdoor and therefrom be able to extract committed bits and/or private openings used by the corrupted party in the simulation. (See also Remark 2.16.)

The need for some ZKPs or ZKPoK sub-protocols may change with the type of trusted setup, the chosen BitCom schemes and their use in the protocol (e.g., the correctness of intermediate output BitComs is verified via the cut-and-choose approach and homomorphic properties), the simulatability restrictions (e.g., with rewinding vs. without rewinding) and/or the type of acceptable interactivity (e.g., interactive vs. non-interactive proofs). Given the significant variation with the concrete parameters that may be decided at implementation time, the needed ZKP and ZKPoK requirements are sometimes left somewhat implicit in the remainder of this section. The set of needed ZKPs and ZKPoKs is identified in Table 3.6 in §3.4.1, and is more explicitly considered in the low-level protocol-description in Section B.2.

Stage 1.1 — Produce initial BitComs of P_B.

- **1.1.1.** Outer BitComs of input bits of P_B. Using the outer BitCom scheme ($\mathscr{B}_B$) of P_B, P_B produces an *initial* (outer) BitCom of each of his own input bits, and sends the

BitCom to P_A. If need be (e.g., for ElGamal BitComs, but not for GM BitComs), P_B also sends a NIZKP of correctness of these BitComs to P_A.

- **1.1.2.** Outer BitComs of output offsets of P_B. P_B uses his outer BitCom scheme ($\mathscr{B}_B$) to produce a BitCom of a random offset bit for each (future) private output bit of P_B. These BitComs will later be permuted to new BitComs that commit to the actual private circuit output bits of P_B to be computed by the S2PC. If need be (e.g., for ElGamal BitComs, but not for GM BitComs), P_B sends a NIZKP of correctness of these BitComs.
- **1.1.3.** Intermediate OT BitComs of input of P_B. If the BitCom scheme ($\mathscr{B}_{OT}$) used to support oblivious transfers is different (e.g., for 2-out-of-1 OT) from the outer BitCom scheme of P_B, then P_B builds respective additional BitComs of his input bits to support the needed OT, and sends them to P_A, along with a NIZKP of same committed bits across the two BitCom schemes.
- **1.1.4.** Parse Outer BitComs of P_B. Locally, both P_A and P_B parse the outer BitComs of each type of wire (input and output) of P_B into respective BitStringBitComs. If a compact BitStringCom extension is not directly available or considered, then the parsing may be a simple vectorization of the BitComs (e.g., a vector of GM BitComs, in an IFC instantiation). However, if the underlying BitCom scheme ($\mathscr{B}_B$) can be naturally extended to a BitStringCom scheme ($\mathscr{C}_B$) where the commitment of a string is as large as a single BitCom (e.g., if using ElGamal Coms), then this yields a significant reduction in the final output size of outer Coms, i.e., of the state to retain as final output in the end of the execution). This also reduces the size of the later coin-flipping needed to permute the randomness of the outer Coms. If the trapdoor of the outer Ext-Com scheme is not available (in GCRS), or if it is insufficient (e.g., for ElGamal, in GPKI) for extraction of the "randomness" of the respective commitment, then P_B provides a ZKPoK of said randomness (e.g., §A.3.3 for a NIZKPoK of a discrete log, in a DLC instantiation).

Stage CF.1 — Coin-flip Permutations (Start). For input and output wire sets of P_A, P_B sends an Ext-and-Equiv Com of random outer-randomness permutations. For input and output wire sets of P_B, P_B initiates a generalized coin-flipping (type 1) of outer-Com permutations. (P_B will receive outer-randomnesses, but P_A will only learn the respective outer-Coms). In a hybrid model with access to an ideal commitment $\mathcal{F}_{MCom}$ and a generalized coin-flipping (type-1) $\mathcal{F}_{GMCF\text{-}1}$, P_B simply sends respective commit and coin-flipping requests

to the ideal functionalities, and then the functionalities sends to P_B a receipt that something of a given length has been committed and than a coin-flipping request has been sent.

Stage 1.2 — Produce initial BitComs of P_A.

- **1.2.1.** Outer BitComs of input bits of P_A. P_A uses her outer BitCom scheme ($\mathscr{B}_A$) to produce an *initial* (outer) BitCom of each of her circuit input bits, and sends the BitComs to P_B. If need be (e.g., for ElGamal BitComs, but not for GM BitComs), P_A also sends a NIZKP of correctness of these BitComs to P_B.
- **1.2.2.** Outer BitComs of output offsets of P_A. Out of all circuit output bits to be learned by P_A, it matters to distinguish between common output bits (i.e., to be learned by both parties) and private output bits of P_A (i.e., which the S2PC is supposed to hide from P_B). If the (intermediate) BitCom scheme ($\mathscr{B}_{ConA}$) used for connectors of input of P_A is different from the outer BitCom scheme ($\mathscr{B}_A$), then P_A uses the intermediate BitCom scheme to commit again to all her input bits and to all random offset bits corresponding to private output wires. P_A then sends a NIZKP that the committed bits are the same across the two BitCom schemes.
- **1.2.3.** Intermediate connector BitComs of input of P_A. If the BitCom scheme ($\mathscr{B}_{ConA}$) for connectors of input of P_A is different from the outer Com scheme of P_A, then P_A builds respective additional BitComs of her input bits to support the connectors, and sends a respective ZKP of same committed bits across the two BitCom schemes. This applies both to the input bits of P_A and to the private offset output bits of P_A.
- **1.2.4.** Parse Outer BitComs of P_A. Similarly to the parsing of BitComs of initial outer bits of P_B, both parties locally combine the initial outer BitComs of P_A (of private input bits and of private offset-output bits) into BitStringComs. P_A also computes the respective combination of randomness. Furthermore, for the set of common output bits of P_A, P_A uses her outer Com scheme ($\mathscr{C}_A$) to commit to a string of all zeros, and sends to P_B a respective NIZKP that the committed string is all zeros. Locally, each party may also homomorphically combine this Com of common output bits with the Com of private output bits of P_A.
- **1.2.5. F&L-related Coms of input bits of P_A.** If the Ext-BitCom scheme ($\mathscr{B}_{FLA}$) used to support the forge-and-lose (F&L) technique in respect to wires of P_A is different from any of the previous two BitCom schemes ($\mathscr{B}_A$, $\mathscr{B}_{ConA}$) associated with bits of P_A, then P_A produces a respective additional commitment of all her input and output bits

(possibly directly using a BitStringCom scheme). In this case P_A also gives a ZKP of same committed bits, i.e., in respect to the bits committed by the outer Com scheme.

- **1.2.6. F&L-related BitComs for output bits.** For each (future) output bit of P_A and P_B, P_A uses the Equiv-BitCom scheme ($\mathscr{B}_{FLB}$) that supports the forge-and-lose technique in respect to output connectors, to prepare one Equiv-BitCom, knowing two openings ("randomnesses"), one for bit 0 and another for bit 1. P_A sends the BitComs to P_B, but not the respective pairs of openings. Instead, the openings will be encoded within the respective connectors, so that later P_B learns only one opening per output bit.

Stage 2 — Commit. For each index of the cut-and-choose (C&C) (i.e., for each instance to be selected for *check* or *evaluation*), P_A uniformly samples a short random string (denoted RSC seed) and uses it as a seed to generate a pseudo-random garbled circuit, and respective *connectors* for each input and output wire of the garbled circuit (as later described in Section 3.3). P_A computes a collision-resistant hash (denoted *global hash*) of all the garbled circuits and of all commitments associated with the respective connectors, and sends to P_B only a respective equivocable commitment (denoted RSC commitment).

Remark 3.5 (On using Ext-Coms and Equiv-Coms). A different technique that may improve the complexity of simulations is to send to P_B an Ext-Com of the seed and an Equiv-Com of the hash, both using "randomness" pseudo-randomly generated from the seed. However, since the technique is not required here to enable simulatability, it is omitted to simplify the overall description. The exact same technique is explicitly described in §C.1.4 in Appendix, within the scope of a non-interactive transformation of an Ext-and-Equiv Com.

Stage 3 — Challenge. If the cut-and-choose selection style is interactive, then P_B decides and sends to P_A a random C&C partition, conditioned to the agreed parameters (e.g., the number of *evaluation* challenges being limited within an interval). If the style is instead non-interactive with decision by P_A, then P_A calculates the partition via a non-programmable oracle, using as input the execution context and the RSC Equiv Com (and in this case the statistical security parameter must equate the computational security parameter). (The protocol could be easily adapted to also allow the alternative of non-interactive decision by P_B, in an earlier stage, but this possibility is left implicit (see §2.4.2)).

Stage CF.2 — Coin-flip Permutations (Continue). P_B informs the ideal generalized coin-flipping (type-1) $\mathcal{F}_{\text{GMCF-1}}$ that it may proceed with the coin-flipping of outer-Com permutations for wire sets of P_B (started in stage CF.1). In the real world this is instantiated by having P_A send a coin-flipping contribution to P_B, thus allowing P_B to calculate the final outer-randomness permutations. For wire sets of P_A there is no need for an explicit coin-flipping contribution from P_A because said contribution is already implicitly embedded in the random initial outer-Coms of P_A.

If in the previous Challenge stage the cut-and-choose partition is decided by P_A (based on a NPRO), then a single message from P_A to P_B is sufficient to take care of the communication between the previous Produce initial BitComs of P_A stage (stage 1.2) and the following Respond stage (stage 4).

Stage 4 — Respond. P_A opens to P_B the global hash value from the RSC commitment. Then, for each C&C challenge index, of type *check* or *evaluation*, P_A sends to P_B the elements that constitute either the *reveal for check* or the *reveal for evaluation*, respectively, as needed for the subsequent stages (and as further specified in Section 3.3, including Tables 3.4 and Table 3.5). Specifically: for each *check* index, P_A simply reveals the respective RSC seed, which allows P_B to reconstruct the Type-C and Type-RC elements; for each *evaluation* index, P_A reveals the Type-C and Type-RE elements, which include one wire key for each input wire of P_A, and a multiplier for input and output wire of P_B. This allows P_B to partially verify the correctness of the connectors, and to evaluate the GCs and respective connectors.

Stage 5 — Verify. For each *check* index, P_B uses the RSC seed to regenerate the Type-C elements, namely the GC, the inner BitComs of all connectors and the pseudo-randomly permuted pairs of commitments of input wire keys of P_A. (This involves intermediately regenerating the Type-RC elements, which are otherwise not needed.) For each *evaluation* index, P_B verifies that the key opening for input wires of P_A are correct, and (using XOR-homomorphic properties of the BitComs) that the permutation encoding for input wires of P_A and the multipliers for wires of P_B are correct. Using the regenerated and received elements of connectors and GCs, for both types of challenges, P_B checks that the hash of their concatenation is equal to the global hash that was opened by P_A in the previous step. If some verification fails, then P_B aborts the execution and privately outputs abort.

Stage 6 — EVALUATE**.** For each *evaluation* index, P_B determines one key per input wire of P_A and P_B and evaluates the GC, obtaining one key per output wire of P_B and then using the respective part of the revealed connector (specifically one of the two received multipliers) to obtain an opening (bit encoding) of the respective *intermediate* output BitCom associated with the forge-and-lose technique. For adequate C&C configurations, there is an overwhelming probability that there is at least one *evaluation* GC whose connectors lead to valid openings in all output wires, conditioned to the event that all *check* instances were verified as correct. If all obtained valid openings are consistent across all evaluation instances, then P_B accepts them as correct. Otherwise, P_B proceeds into the forge-and-lose path as follows: if it finds two different openings of the same output BitCom (i.e., a non-trivially correlated pair of openings), for some wire index across two different garbled circuits, then it uses them as a trapdoor of the Ext-BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) of the forge-and-lose technique (which was used to commit the input bits of P_A and the bit-masks for the private output bits of P_A); P_B uses this trapdoor to extract the Ext-committed bits of P_A, and then uses the input bits of both parties to directly evaluate the Boolean circuit and thus obtain the final circuit output; finally, from within the output BitCom openings already obtained in the initial part of this stage, P_B finds those that are consistent with the circuit output bits, and accepts them as the correct ones.

Stage 7 — TRANSMIT CIRCUIT OUTPUT OF P_A**.**

- For private output wires of P_A, P_B sends to P_A the circuit output bits of P_A, which are masked, and then proves that these are indeed the bits that it obtained in the garbled circuit evaluation. Depending on the type of instantiation, the proof might be more or less simple. In general it can be a NIZKPoK of the known openings, but in specific circumstances (including the PKI instantiations considered herein) it does not need to be ZK and thus P_B may simply send an encryption (decryptable by P_A) of the CR-Hash of the known openings. Since P_A already knows all openings, it can recompute the hash of the openings corresponding to the informed masked bits, to verify correctness, and then decide the circuit output as the respective unmasking.
- For common output wires, P_B sends to P_A an encryption of the obtained bits, such that P_A can decrypt them, and then sends a NIZKP that those are indeed the obtained bits. In practice the encryption can be based on a public-key Ext-BitCom scheme, with private-key (trapdoor) explicitly known by P_A (e.g., the one used with the forge-and-lose technique).

The encryption is required to handle the case of a passive adversary (eavesdropper) that (per the defined ideal functionality) is not supposed to learn the circuit output.

Naturally, this stage can be avoided all-together in the case of S2PC of a Boolean circuit without output wires for P_A.

Stage CF.3 — Coin-flip Permutations (Finish)**.** The coin-flipping sub-protocol is finalized with P_B opening his previously committed contribution of permutations of outer-randomness for wire sets of P_A, and letting the ideal generalized coin-flipping functionality $\mathcal{F}_{\text{GMCF-1}}$ sending to P_A the outer-Com permutations for wire sets of P_B in practice this stage may involve an additional NIZKP of correctness of the respective outer-Com permutations.

Stage 8 — Permute Outer Coms**.**

- **Adjust initial outer BitComs of output bits of P_B.** P_B sends to P_A the masked-output bits that are needed to correct the initial random committed bits (associated with outer encodings of output bits), unknown to P_A, into the actually obtained circuit output bits of P_B, also unknown to P_A. Then, P_B also sends a NIZKP that these masked-bits are correct, i.e., that their XOR with the initial random bits is equal to the bits for which P_B learned a valid opening as a result of the S2PC. This can be reduced to a NIZKP of same committed bits across two BitCom schemes ($\mathscr{B}_B$, $\mathscr{B}_{\text{FLB}}$). Based on the disclosed offset bits and the homomorphic properties of the outer BitCom scheme ($\mathscr{B}_B$), each party locally adjusts the outer BitComs, and P_B correspondingly adjusts the respective outer openings. The adjusted BitComs and openings are still denoted as initial.
- **Adjust initial outer BitComs of output bits of P_A.** P_B sends to P_A the learned masked-bits of output of P_A, along with a proof of knowledge of the respective openings of the BitCom scheme $\mathscr{B}_{\text{FLB}}$ of connectors of output. The resulting BitComs and openings, upon the described adjustment, are denoted as initial.
- **Apply random BitCom permutations.** Each party applies the previously decided (coin-flipped) permutations to the *initial* outer randomnesses (eventually already adjusted in case of output bits) associated with the respective circuit input and output bits, and applies the BitComs associated with the coin-flipped randomnesses as permutations to the respective BitComs of the circuit input and output bits of both parties. Given the XOR-homomorphism of the outer BitCom schemes, the initial and the final BitComs commit to the same bits. Since the simulator is able to extract the openings from the

initial Coms (e.g., upon the ZKPoKs of trapdoor and/or of opening of respective BitComs), it is also able to homomorphically compute the opening of the respective final outer Coms.

Remark 3.6 (On the outer BitCom scheme used for output bits of $\mathbf{P_B}$). In the original forge-and-lose paper [Bra13], the final outer BitComs of the output of P_B were based on an unconditionally-hiding BitCom scheme with trapdoor explicitly known by P_A. For that reason, there the intermediate BitCom scheme ($\mathscr{B}_{\text{FLB}}$) supporting the forge-and-lose technique could be the same. There P_B did not need to commit to random offset bits as a preparation for adjusting the final BitComs of the output of P_B. Yet, since a coin-flipping was still required to randomize the final BitComs, the minimal possible number of rounds was the same (three). The stylistic variation in this dissertation, with the outer BitComs of P_B being instead based on a BitCom scheme ($\mathscr{B}_B$) whose trapdoor is not known by P_A (and which might or might not be known by P_B, e.g., respectively for PKI or CRS trusted setups), extends its suitability to the case of outer BitCom schemes being defined by a CRS setup. It also allows, in a PKI setting, that the BitComs associated with each party are always associated with the public parameters of said party, which may be useful in larger multi-party protocols where BitComs may be transferred across several parties.

Stage 9 — Final Output. Each party privately outputs her circuit input and output bits and the respective final encodings, and also outputs the (commonly known) final outer BitComs of the circuit input and output bits of both parties.

Remark 3.7 (The use of forge-and-lose path should not be disclosed). Even if there are no output circuit wires for P_A, care is needed to prevent P_B from disclosing whether the output was learned via the regular path or the forge-and-lose path. In particular, to prevent a timing attack that could be used as a successful selection failure attack, the time that takes P_B to disclose to P_A the finish of the S2PC computation should not vary depending on whether or not the forge-and-lose path was used.

3.3 Connectors

This section explains how the "randomness" (i.e., the "encoding" needed for opening) of outer BitComs (of input and output bits) are *connected* to the respective input and output wire

keys of GCs. The construction varies with the type of wire: input of P_A (§3.3.1), input of P_B (§3.3.2) and output of P_B (§3.3.3). A final note is also given on allowing any wire keys needed by the GC generating mechanism (§3.3.4).

Connectors are used within the C&C approach, statistically ensuring the consistency of the input bits used and the output bits obtained across garbled circuits, but without ever revealing the actual private bits. Since connectors are only needed for input and output wires, their overall cost does not depend on the number of internal circuit wires and gates in the circuit. Specifically, the overall construction requires communication of group elements (multipliers and inner encodings) in number proportional to the number of input and output wires, and also proportional to the number of evaluation challenges, but independent of the number of internal wires in the circuit. Some optimizations are described in Appendix B.4.

Connectors are used in a context somewhat similar to a commitment scheme that has *commit* and *open* phases. The starting point of the connector construction assumes that one *initial* XOR-homomorphic *intermediate* BitCom has been produced for each respective wire, independently of the number of GCs. Then, for each input and output wire in each GC, a connector is built to provide a (statistically verifiable) connection between the pair of keys of the respective wire and the openings of the respective outer BitComs.

A high-level depiction of connectors has already been given in Illustration 3.2 and Illustration 3.3. This section describes the connectors in detail. In the Commit stage of the overall protocol, P_A commits to the *connector* elements, thus becoming bound to a connection between the BitCom openings and the respective wire keys. Then, in the Respond stage, P_A provides responses to either a *reveal for check* or a *reveal for evaluation* mode. P_B then proceeds to the Verify stage to verify all connectors, for the respectively selected *reveal* modes (*check* and *evaluation*). Only if all verifications are successful does P_B continue to the Evaluate stage, where it uses the connector elements received in the *reveal for evaluation* mode (for *evaluation* instances) to obtain one BitCom opening for each output BitCom.

It is worth noticing that connectors are logically separated from the initial *outer* BitComs — the ones that need to be randomly permuted in the Permute Outer Coms stage. Nonetheless, for some instantiations the BitCom scheme (e.g., $\mathscr{B}_A$) used for certain outer BitComs (e.g., for the input bits of P_A), can be the same as the BitCom scheme (e.g., $\mathscr{B}_{ConA}$) used for the respective connectors. Several such examples can be deduced from Table 3.3. When the outer BitCom scheme is different from the intermediate BitCom scheme, then the

Table 3.4: Notation related to connectors

Denomination	Symbols	Description
Outer bit	b	Private bit of an input or output wire of the circuit, as committed by the respective party.
Challenge indices	j, (J_V, J_E)	Index of instance in the cut-and-choose set; subsets of *check* and *evaluation* indices of instances, respectively.
Wire indices	i, (I_A, I_B, O_B)	Index of input or output wire; sets of wire indices of input of P_A, input of P_B, output of P_B, respectively.
Intermediate encoding	μ	Encoding of the respective outer bit, being an opening of the respective BitCom.
Inner encoding	ν	Known opening of a BitCom used inside a connector, used to help with the *connection*.
Encoded bit	$.^{(c)}$	(indication that a certain group element $\cdot$ is in class c, i.e., is an encoding of bit c.
Multiplier	α, β	Encodings that connect an outer encoding into an inner encoding (or vice-versa): α is for wires of P_A; β is for wires of P_B.
BitComs	α', β', μ', ν'	BitComs associated with the respective encodings α, β, μ, ν, i.e., BitComs of the respective underlying bits.
Wire key	k or ξ	Key of input or output wire (k for keys computed by P_A; ξ for (tentative) keys transmitted to or obtained by P_B).
Value, position and permutation	$.^{[b]}, \pi, .^{<c>}$	Bit value underlying the key $\cdot$; permutation bit; position of key $\cdot$ within a permuted pair of keys ($\langle c \rangle \equiv [c \oplus \pi]$).
Key commitment and its randomness (Only for input of P_A)	$\overline{k}$, $\underline{k}$	Bit-string commitment of key k; randomness needed to generate and/or open k from $\overline{k}$.

consistency across respective BitComs must be guaranteed, e.g., by means of zero-knowledge proofs (still independent of the number of garbled circuits).

At high level, the notion of "connector" intends to encapsulate the complexity involved in making a connection between BitComs and circuit wire keys, as well as the specificity associated with each type of wire. Nonetheless, the nomenclature used to describe their internal elements (e.g., "multipliers" and "inner encodings" and respective BitComs) is common across the types of connectors. Table 3.4 shows notation related to connectors.

The connectors are also integrated into a RSC technique. In the Commit stage, P_B only receives a short RSC Equiv-Com $\overline{\Lambda}$ of a CR-Hash (*global hash*) of all Type-C elements (GCs and commitments of the elements that compose the connectors). Later, P_A opens the global hash in the Respond stage. For each *check* index $j \in J_V$, P_A then sends, in the Respond stage, only the respective seed λ_j, allowing P_B to use an appropriate PRG to first recompute intermediate Type-RC (*reveal for check*) elements and then the Type-C elements.

Table 3.5: Types of communicated elements related to connectors

A	B	C	D	E	F	G	H	I	J	
Wire index	Type of element		Type-RC		Type-C			Type-RE	Related description	1
$i \in I_A$	Key openings		$\left(\left(k_{j,i}^{[0]}, \underline{k}_{j,i}^{[0]}\right), \left(k_{j,i}^{[1]}, \underline{k}_{j,i}^{[1]}\right)\right)$	—	—	—	—	$\left(k_{j,i}^{[b_i]}, \underline{k}_{j,i}^{[b_i]}\right)$	Illust. 3.4 (§3.3.1)	2
	Key commitments		—	—	$\left(\overline{k}_{j,i}^{\langle 0\rangle}, \overline{k}_{j,i}^{\langle 1\rangle}\right)$	—	$\overline{k}_{j,i}^{[1\oplus b_i]}$	—		3
	Group elements or scalars		$\alpha_{j,i}{}^{(\pi_{j,i})}$	—	$\nu'_{j,i}$	—	—	$\nu_{j,i}^{(b_i \oplus \pi_{j,i})}$		4
$i \in I_B$	Group elements or scalars	if 2/1 OT	$\left(\nu_{j,i,0}^{(0)}, \nu_{j,i,1}^{(1)}\right)$	—	$\left(\nu'_{j,i,0}, \nu'_{j,i,1}\right)$	—	—	$\left(\beta_{j,i,0}^{(0)}, \beta_{j,i,1}^{(0)}\right)$	Illust. 3.5a (§3.3.2.1)	5
		if 1/2 OT	$(\beta_{j,i,0}, \beta_{j,i,1})$	—	—	$\nu'_{j,i}$	—	—	Illust. 3.5b (§3.3.2.2)	6
$i \in O_B$	Group elements or scalars		$\left(\nu_{j,i,0}^{(0)}, \nu_{j,i,1}^{(1)}\right)$	—	$\left(\nu'_{j,i,0}, \nu'_{j,i,1}\right)$	—	—	$\left(\beta_{j,i,0}^{(0)}, \beta_{j,i,1}^{(0)}\right)$	Illust. 3.6 (§3.3.3)	7
—	Others		—	λ_j	—	GC_j	—	—	—	8
Sent for which instances		if using RSC	—	$j \in J_V$	—	$j \in J_E$			—	9
		if not using RSC	$j \in J_V$		$j \in [s]$		—	J_E	—	10

Legend: notation from Table 3.4 applies; GC (garbled circuit); λ_j (PRG seed used to generate GC_j and the connector elements of instance j); 1/2 OT (1-out-of-2 OT); 2/1 OT (2-out-of-1 OT); RSC (random seed checking technique).

Type of OT. Rows 5 and 6 are alternative, i.e., either the 1/2 OT or the 2/1 OT is used. The element in cell G6 is directly decipherable by P_B, contrarily to the commitments in column F.

RSC vs. non-RSC technique. If using RSC (row 9), the RSC seed λ_j (in cell E8) is enough to generate all respective Type-C elements in columns F–G, and all respective Type-RC elements. If not using RSC (row 10), λ_j is simply the randomness needed to prove correctness of the GCs of *check* instances. The key commitment in cell H3 is redundant with one of the two commitments in cell F3, but only one of the cells matters for communication (H3 if using RSC; F3 otherwise).

P_B is thus able to verify correctness, by checking that the Type-C elements of the check instances contribute to a correct pre-image of the committed global hash. For *evaluation* indices $j \in J_E$, and still in the RESPOND stage, P_A does not send the RSC seed λ_j, but sends the Type-RE (*reveal for evaluation*) elements that allow a verifiably correct connection between input wire keys (different across evaluation GCs) and input BitComs (independent of the number of GCs), and between output wire keys and output BitComs. It also sends some complementary Type-C elements necessary for circuit evaluation (e.g., the GCs), for verification of the global hash (e.g., complementary key commitments, for the input keys of P_A) and possibly to determine input keys of P_B(if using the 1-out-of-2 OT method).

In order to better differentiate types of elements and to promote a comparative benchmarking, the Table also considers the case of non-use of RSC (row 10).

The next subsections describe in detail how the connector for each type of wire (input of P_A, input of P_B, output of P_B) is integrated within the cut-and-choose and the RSC technique. Table 3.5 summarizes types of elements (Type-C, Type-RC, Type-RE), per type of wire (input of P_A, input of P_B, output of P_B).

3.3.1 Connectors for input of P_A

For each *evaluation* instance $j \in J_E$, the main challenge for each input wire $i \in I_A$ of P_A is to simultaneously ensure: (i) the hiding of the bit b_i underlying the single wire key $k_{j,i}^{[b_i]}$ opened by P_A (cell I2 in Table 3.5); and (ii) the consistency between the underlying bit b_i and the bit committed by the respective intermediate BitCom μ'_i (i.e., that they are the same).

The hiding of the bit b_i underlying the opened key $k_{j,i}^{[b_i]}$ is achieved by associating it with the position $\langle\cdot\rangle$ of the opened key $k_{j,i}^{\langle b_i \oplus \pi_{j,i}\rangle}$ in respect to a respective pair $\left(\overline{k}_{j,i}^{\langle 0\rangle}, \overline{k}_{j,i}^{\langle 1\rangle}\right)$ of randomly permuted key commitments (cell F3). The applied permutation bit $\pi_{j,i}$ is not revealed, but the permuted position bit $b_i \oplus \pi_{j,i}$ is thus learned (where $[c] \equiv \langle c \oplus \pi_{j,i}\rangle$).

The consistency between the committed input bit and the bit underlying the opened key is statistically "proven" with the help of *check* instances ($j \in J_V$). After the RSC seed is revealed, P_B is able to learn the permuted pair of keys $\left(k_{j,i}^{\langle 0\rangle}, k_{j,i}^{\langle 1\rangle}\right)$ (cell D2), along with the permutation bit $\pi_{j,i}$ (but not the permuted bit $b_i \oplus \pi_{j,i}$), thus being able to check that the committed permutation is correct. This verification is possible because the permutation bit is itself committed with a XOR-homomorphic BitCom $\alpha'_{j,i}$, with its opening being a *multiplier* $\alpha_{j,i} \equiv \alpha_{j,i}^{(\pi_{j,i})}$ that can be verified as the randomness of the permutation bit in respect to the respective inner BitCom $\alpha'_{j,i}$. Some elements of the *connector* construction are depicted in Illustration 3.4 and mentioned in Table 3.5, as a complement to the following description.

The BitComs for intermediate, multipler and inner BitComs can be based on XOR-homomorphic Goldwasser-Micali encryptions (Ext), or Blum BitComs (Equiv), or on pseudo XOR-homomorphic Pedersen BitComs (Equiv) or Elgamal BitComs (Ext). A significant optimization based on bit-string Coms is explained in Appendix B.4.

3.3.1.1 Construction based on a XOR-homomorphism

Pre-condition. In the Produce initial BitComs of P_A stage, P_A uses a XOR homomorphic BitCom scheme ($\mathscr{B}_{\text{ConA}}$) to select one random encoding $\mu_i^{(b_i)}$ of her input bit b_i and then sends the respective BitCom μ'_i to P_B. The randomness and the BitCom are qualified as *intermediate*, as they stand between the outer BitComs (from $\mathscr{B}_A$) and the *connectors*. This is contrast with *inner* components that are part of the connectors and have some relation with the input keys $k_{j,i}^{[c]}$ of garbled circuits. For simplicity of description it may be assumed that the intermediate BitComs and respective randomnesses are the exact same as the outer

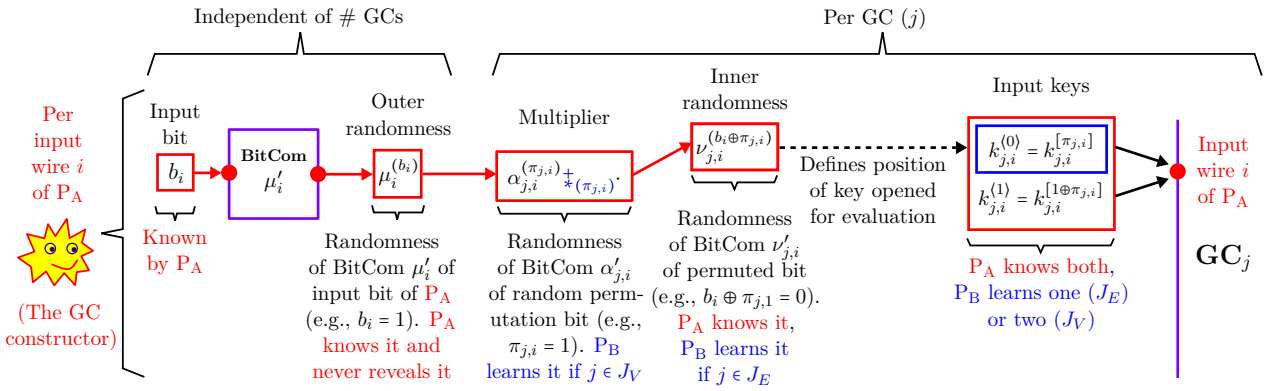


Illustration 3.4: Connection for input wires of $\mathrm{P_A}$. Notation from Table 3.4 applies. The multiplier α is the randomness used to commit the permutation bit π, whereas ν is the randomness used to commit the permuted bit $(b \oplus \pi)$. If the BitCom scheme is unconditionally hiding with trapdoor, then $\mathrm{P_B}$ might (if it knows the trapdoor) know two openings of each BitCom but it will not know which one $\mathrm{P_A}$ knows. In the construction, $\mathrm{P_A}$ will reveal the inner randomness (for *evaluation* challenges) or the multiplier (for *check* challenges), but never both at the same time, and never the outer randomness $\mu_i^{(b_i)}$. In case of a regular XOR-homomorphism, the group operation $\dot{+}_{(\pi_{j,i})}$ at the level of randomnesses can be (in multiplicative notation) a simple group multiplication, independent of $\pi_{j,i}$; in case of a pseudo XOR-homomorphism, the operation $\dot{+}_{(\pi_{j,i})}$ is a group addition or subtraction (in additive notation) (by the second argument $\mu_i^{(b_i)}$), when $\pi_{j,i}$ is respectively 0 or 1. See §B.4.1 for a generalization to bit-string Coms.

ones. However, some optimizations are possible when using different schemes and adding a NIZK that the BitComs across different schemes committ to the same bits.

Commit. In the COMMIT stage, $\mathrm{P_A}$ selects, for each GC index $j \in [s]$, a random seed λ_j (just one across all wire types, including for wires of $\mathrm{P_B}$), to be used in the subsequent generation of elements. For each pair of GC index $j \in [s]$ and input wire index $i \in I_\mathrm{A}$, $\mathrm{P_A}$ generates a pseudo-random permutation bit $\pi_{j,i}$ and a respective pseudo-random "randomness" $\alpha_{j,i} \equiv \alpha_{j,i}^{(\pi_{j,i})}$ — a group-element dubbed *multiplier* — using the same BitCom scheme used to commit the respective input bit b_i of $\mathrm{P_A}$. Then, $\mathrm{P_A}$ uses the XOR-homomorphic group operation to obtain a new randomness $\nu_{j,i} \equiv \nu_{j,i}^{b_i \oplus \pi_{j,i}}$, dubbed *inner randomness*, for committing the permuted version $b_i \oplus \pi_{j,i}$ of her input bit. In other words, the inner randomness is obtained by an appropriate group product between the outer randomness and the multiplier. Then, using some bit-string commitment scheme $\mathscr{C}_{\mathrm{InKey}}$, $\mathrm{P_A}$ builds one commitment $\overline{k}_{j,i}^{[c]}$ of each of the two input wire keys $k^{[c]}$ (one for bit 0 and the other for bit 1, i.e., for $c \in \{0,1\}$), and then prepares them into a respectively permuted pair $(k_{j,i}^{\langle 0 \rangle}, k_{j,i}^{\langle 1 \rangle})$. Instead of directly sending the inner BitCom $\nu'_{j,i}$ and the pair of committed wire keys to $\mathrm{P_B}$, $\mathrm{P_A}$ adds them to

the tuple of elements whose CR-Hash will be computed into a *global hash*, in order to then be committed into a single short RSC Equiv-Com.

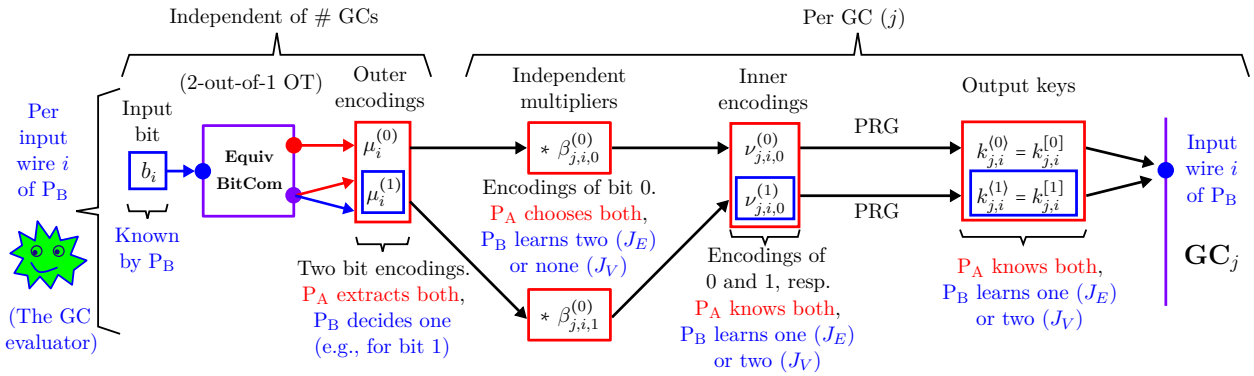
Reveal for *check*. In the RESPOND stage, P_A opens to P_B the RSC seed λ_j (one across all wires of P_A and P_B) associated with each *check* index $j \in J_V$. This allows P_B, using the adequate PRG procedures, to recompute, for each check instance j and each input wire $i \in I_A$ of P_A, the pair $(k_{j,i}^{[0]}, k_{j,i}^{[1]})$ of wire input keys and the respective *permutation bit* $\pi_{j,i}$ and verify that they are correct, i.e., that the respective key commitments and permutation randomness contribute to a correct pre-image of the global hash opened by P_A.

Reveal for *evaluation*. Also in the RESPOND stage, P_A opens to P_B, for each pair of *evaluation* index $j \in J_E$ and input wire index $i \in I_A$ of P_A, the input key $k_{j,i}^{[b_i]}$ (corresponding to her input bit b_i), and the *permuted input* bit $b_i \oplus \pi_{j,i}$ (by revealing the respective inner randomness $\nu_{j,i}^{(\pi_{j,i})}$). This allows P_B to verify that the opened key $k_{j,i}^{[b_i]} = k_{j,i}^{\langle b_i+\pi_{j,i}\rangle}$ is in a position consistent with the permuted bit. Since this position is independent of the real input bit b_i, nothing is revealed about the bit underlying the opened key. If P_A would instead reveal the other key, P_B would detect the inconsistency and be able to complain.

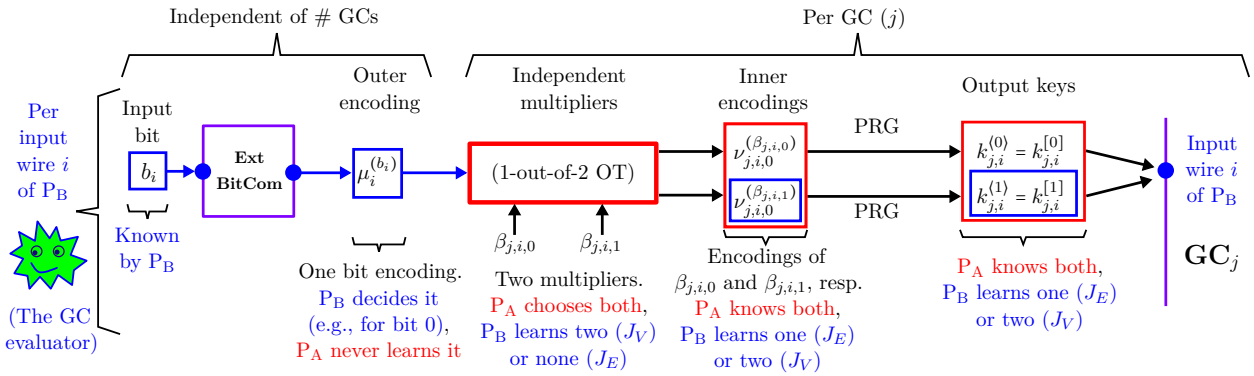
The connectors maintain all their properties if the XOR-homomorphic BitCom scheme is replaced by an additively-homomorphic commitment scheme (e.g., ElGamal or Pedersen), replacing the homomorphic operation with a pseudo-XOR homomorphic operation §2.3.2. The Appendix (§B.4.1.1) describes a further optimization based on bit-string commitments.

3.3.2 Connectors for input of P_B

For each pair (j, i) of *evaluation* instance $j \in J_E$ and input wire index $i \in I_B$ of P_B, the challenging aspect is that P_B must only learn one wire input key $k_{j,i}^{[b_i]}$, corresponding to his respective input bit b_i, and P_A must simultaneously not know which one was learned. Also, P_A must not be able to perform a selective failure attack (e.g., produce elements for which the evaluation by P_B is successful if a certain bit of P_B is 0, but unsuccessful if the bit is 1). Furthermore, this should be integrated with the BitCom approach, such that P_B learns the keys corresponding to the committed input bits. The bit b_i underlying each input key $k_{j,i}^{[b_i]}$ learned by P_B does not need to be hidden from P_B, because P_B knows his own private input bits. Thus, the random bit permutation $\pi_{j,i}$ previously considered for input wires of P_A does



(a) Based on 2-out-of-1 OT. Both multipliers (β) are encodings of 0, XOR-homomorphic under $*$, ensuring that each outer encoding (μ) (of which P_B only knows one) encodes the same bit as the respective inner encoding (ν).



(b) Based on 1-out-of-2 OT

Illustration 3.5: Connection for input wires of P_B. Notation from Table 3.4 applies.

not need to be considered for the input wires of P_B.

Two *connector* constructions are described for the input bits of P_B: one based on a 2-out-of-1 OT at the level of BitComs (i.e., one OT for each input bit of P_B, independently of the number of garbled circuits) (§3.3.2.1), suited to an IFC instantiation based on Blum BitComs; another based on 1-out-of-2 OT at the level of wire keys of the garbled circuits (§3.3.2.2), suited to a DLC instantiation based on ElGamal BitComs.

3.3.2.1 Construction based on a 2-out-of-1 OT

The connection is depicted in Illustration 3.5a.

Pre-condition. In the Produce initial BitComs of P_B stage, P_B uses a XOR-homomorphic 2-to-1 square scheme (which in particular is an Equiv BitCom scheme) to select one

random encoding $\mu_i^{(b_i)}$ for each of his input bits b_i and then sends the respective BitCom μ'_i to $\mathrm{P_A}$. Using the trapdoor of the 2-to-1 square scheme, $\mathrm{P_A}$ executes a non-interactive 2-out-of-1 oblivious transfer, i.e., it extracts the two possible encodings $(\mu_i^{(0)}, \mu_i^{(1)})$, dubbed *outer* encodings, from the BitCom value. Since the two possible outer encodings (i.e., two non-trivially correlated proper square-roots of the BitCom) computed by $\mathrm{P_A}$ constitute a trapdoor of $\mathrm{P_A}$, they cannot be used directly as input wire keys, or otherwise $\mathrm{P_B}$ would learn the trapdoor once learning all the keys of *check* GCs. Instead, the idea described below is to deal with two independent inner encodings $(\nu_{j,i,0}^{(0)}, \nu_{j,i,1}^{(1)})$ that $\mathrm{P_B}$ may learn for check instances. Given the XOR-homomorphic properties of BitComs, the consistency between inner and outer encodings will be verifiable (even for *evaluation* instances) based only on the respective BitComs.

Commit. In the COMMIT stage, $\mathrm{P_A}$ uses, for each GC index $j \in [s]$, the respective random seed λ_j to generate, for each input wire $i \in I_{\mathrm{B}}$ of $\mathrm{P_B}$, a pair of independent inner randomnesses $(\nu_{j,i,0}^{(0)}, \nu_{j,i,1}^{(1)})$, one for each bit (cell D5 in Table 3.5). $\mathrm{P_A}$ computes the respective inner BitComs $(\nu'_{j,i,0}, \nu'_{j,i,1})$ (cell F5) and then uses an adequate PRG procedure ($\mathrm{PRGen_{InKey}}$) to generate from each inner randomness $\nu_{j,i,c}^{(c)}$ a respective pseudo-random input key $k_{j,i}^{[c]}$ for the garbled circuit. The BitComs (along with the GCs and other elements across different wire types) are then used to compute the *global hash*, whose (RSC) Equiv-Com is sent to $\mathrm{P_B}$.

Reveal for *check*. In the RESPOND stage, $\mathrm{P_A}$ opens to $\mathrm{P_B}$ the RSC seed λ_j (the same across all wires of $\mathrm{P_A}$ and $\mathrm{P_B}$) for each *check* index $j \in J_V$. This will allow $\mathrm{P_B}$ to recompute the inner BitComs, along with the GCs and the output wires of $\mathrm{P_B}$, to recompute the global hash, and thus verify the correctness of the response of $\mathrm{P_A}$.

Reveal for *evaluation*. Also in the RESPOND stage, $\mathrm{P_A}$ reveals, for each *evaluation* index $j \in J_E$, the two multipliers $(\beta_{j,i,0}^{(0)}, \beta_{j,i,1}^{(0)})$ (cell I5 in Table 3.5) — associated to bit 0, leading the two possible outer encodings $(\mu_i^{(c)})$ into the respective inner encodings $\nu_{j,i,c}^{(c)}$. Even though $\mathrm{P_B}$ only knows one outer encoding $\mu_i^{(b_i)}$, it can use the homomorphic property to verify that both multipliers are correct, namely that the respective multiplier BitComs $\beta'_{j,i,c}$ lead the outer BitCom μ'_i into the respective inner BitComs $\nu'_{j,i,c}$. Since are associated with bit 0, it follows from the XOR-homomorphism that each inner encoding $\nu_{j,i,c}^{(c)}$ encodes the same bit c as the respective outer encoding $\mu_i^{(c)}$. Thus, $\mathrm{P_B}$ can safely use the multiplier $\beta_{j,i,b_i}^{(0)}$ corresponding to

the known outer encoding $\mu_i^{(b_i)}$ to determine a single inner encoding $\nu_{j,i,b_i}^{(b_i)}$, assured that it corresponds to the intended bit b_i, and then use the PRG procedure to generate the respective input key $k_{j,i}^{[b_i]}$. This procedure is resilient to selective failure attack because both multipliers are verified for correctness, and because the generated input key is statistically correct, i.e., it would be detected as incorrect if this instance had instead been selected for *check*.

3.3.2.2 Construction based on a 1-out-of-2 OT

The connection is depicted in Illustration 3.5b. The 1-out-of-2 OT is described in low-level in Appendix B (§B.4.2.1).

Pre-condition. In the PRODUCE INITIAL BITCOMS OF P_B stage, P_A commits to each of his input bits using an XOR pseudo-homomorphic BitCom scheme with extraction trapdoor known by P_B, supported on a multiplicative homomorphic encryption scheme, i.e., P_B encrypts ($\mathscr{E}$) each of his input bits.

- **Select parameters.** For example, using an ElGamal BitCom scheme (see §2.3.3.2) the procedure is as follows. P_B selects the parameters of the commitment scheme: a first public generator g_0 may have been previously fixed; a second public generator g_B is computed by raising the first generator to the power of a random secret exponent x.
- **Produce BitComs.** Then, P_B produces and sends to P_A an ElGamal BitCom μ_i' for each input bit $i \in I_B$ of P_B. P_B also sends a corresponding NIZKP that all pairs are indeed ElGamal BitComs (i.e., commitments to 0 or 1) (see §A.3.2).
- **Allow extraction of committed bits.** In settings where the simulator is able to known the trapdoor, e.g., in a PKI setting if a ZKPoK of trapdoor has already been provided, then no additional ZKPoK is necessary. However, in a global CRS setting, where the simulator is not able to learn the trapdoor, a ZKPoK of the committed bit is necessary. In simulations do not allow rewinding, the ZKPoK can be achieved by producing new BitComs with parameters for which the simulator knows a trapdoor, and then sending a ZKP of same committed bits. If rewinding is allowed (and thus not in a UC setting) a simple Schnorr protocol (see §A.3.1) would be enough.

Commit. In the COMMIT stage, for each pair (j, i) of challenge index $j \in [s]$ and input wire index $i \in I_B$ of P_B, P_A uses the BitCom μ_i' of each input bit b_i of P_B to compute the

final message of a two-move honest-sender 1-out-of-2 oblivious transfer, which essentially is a randomized *linear homomorphic transformation* (LHT) of the initial commitment μ'_i of $\mathrm{P_B}$, as follows. Initially, $\mathrm{P_A}$ selects a pair $(\beta_{j,i,0}, \beta_{j,i,1})$ of random exponents (called *multipliers*) (cell D6 in Table 3.5), and uses them to produce a LHT of the original commitment μ'_i, i.e., to homomorphically compute the ElGamal commitment $\nu'_{j,i}$ (not BitCom) of the multiplier β_{j,i,b_i} with position index b_i equal to the bit committed by $\mathrm{P_B}$, but without learning said bit. Then, to ensure semantic hiding from $\mathrm{P_A}$, $\mathrm{P_B}$ also selects a new random exponent and uses it to randomize the ElGamal Com into a new ElGamal Com $\nu'_{j,i}$ that semantically hides the previous two exponents used by $\mathrm{P_B}$. This last step corresponds to an homomorphic sum of 0, i.e., combining a new encryption of 0. The resulting element is an ElGamal ciphertext, which can be decrypted by $\mathrm{P_B}$ to yield the result of raising the global generator g_0 to the mentioned multiplier value β_{j,i,b_i}.

The new commitment $\nu'_{j,i}$ produced by $\mathrm{P_A}$ is denoted as an *inner commitment* (or inner ciphertext), in contrast to the outer BitCom μ' of the input bit of $\mathrm{P_B}$. It is used as a Type-C element (cell G6), part of the pre-image of the global hash Λ of the RSC technique.

Reveal for *check*. In the RESPOND stage, $\mathrm{P_A}$ opens to $\mathrm{P_B}$ the RSC seed λ_j (the same across all wires of PA and PB) for each *check* index $j \in J_V$. $\mathrm{P_B}$ uses the seed to recompute both pseudo-random multipliers $(\beta_{j,i,0}, \beta_{j,i,1})$, and uses them to homomorphically recompute the commitment value $\nu'_{j,i}$ and verify that it contributes to a valid pre-image of the global hash.

Reveal for *evaluation*. Also in the RESPOND stage, $\mathrm{P_B}$ reveals, for each *evaluation* index $j \in J_E$, the ciphertext value $\nu'_{j,i}$ (cell I6), allowing $\mathrm{P_B}$ to decrypt ($\mathscr{E}^{-1}$) a respective inner plaintext ν_{j,i,b_i}, (the result of raising the global generator g_0 to the power of a multiplier β_{j,i,b_i}), and use it as a seed to pseudo-randomly generate the respective input wire key $k_{j,i}^{[b_i]}$. $\mathrm{P_B}$ also checks that the evaluation ciphertexts contribute to a valid pre-image of the global hash.

3.3.3 Connectors for output of $\mathrm{P_B}$

The connection between output wire keys and openings of output BitComs is somewhat similar to a reversed version of the construction for input wires of $\mathrm{P_B}$ in the method of 2-out-of-1 OT. However, the underlying Equiv BitCom with trapdoor does not need to be a full-fledged 2-to-1 square scheme, because the construction does not involve OTs and it is

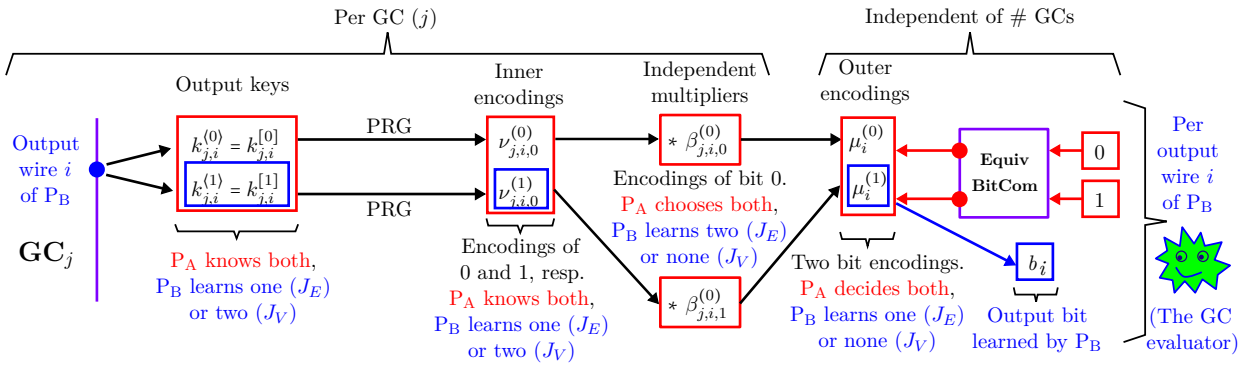


Illustration 3.6: Connection for output wires of P_B. Notation from Table 3.4 applies.

P_A who initially decides the BitComs. In other words, for connectors of output of P_B, P_A is able to compute pairs of openings at the same time as it decides the respective BitCom. In contrast, in the 2-out-of-1 OT for a connector of an input bit of P_B the two openings must be computed by P_A from BitComs decided by P_B. Still, in order to enable the forge-and-lose technique, the BitCom scheme must satisfy the property that two openings (for two different bits) associated with the same BitCom are equivalent to a trapdoor that allows opening the BitComs of the input bits of P_A. This is satisfied not only by Blum BitComs (IFC) but also by Pedersen BitComs (DLC), where any two representations of the same BitCom enable the feasible computation of the discrete log between two generators, which can for example be the trapdoor of the ElGamal BitComs used to commit the input of P_A.

At high level, the connector construction is as follows: for each output bit of P_B, P_A sends to P_B an Equiv-BitCom for which P_A knows two possible openings; then, P_A *helps* P_B connect the output keys of evaluated garbled circuits to the respective openings of the Equiv BitComs; an important requirement is avoiding selective failure attack, i.e., prevent P_A from inducing a situation where the success vs. failure of the evaluation depends on some particular output bit of P_B. For simplicity, it is assumed that the output keys of the garbled circuits exceptionally disclose the respective underlying bit, e.g., by making the respective GC generation procedure $\mathrm{PRGen}_{\mathrm{GC}}$ generate output keys whose least significant bit ($\epsilon(\cdot)$) is the respective underlying bit of the key.

Pre-condition. In the PRODUCE F&L-RELATED COMS stage (stage 1.2.6), P_A prepares Equiv BitComs μ_i' for the output bits of P_A, even without any party yet knowing the respective circuit output bit. P_A, knowing a trapdoor of the BitCom scheme, is able to determine two respective openings $\mu_i^{(c)}$ of each such BitCom, one for each possible bit value c. P_B does not

initially know any opening — it will learn only one ($\mu_i^{(b_i)}$) during the EVALUATE stage, or two if P_A misbehaves (and in that case it will be able to activate the forge-and-lose path).

Commit. In the COMMIT stage, P_A uses, for each GC index $j \in [s]$, the respective random seed λ_j to pseudo-randomly generate a garbled circuit, also obtaining two respective keys $k_{j,i}^{[c]}$ for each output wire $i \in O_B$. P_A uses a PRG procedure $\text{PRGen}_{\$\text{ForCom}}$ to compute from each such key a respective inner encoding $\nu_{j,i,c}^{(c)}$ for a respective BitCom $\nu'_{j,i,c}$ of the output bit of P_B. (In order to improve the collision resistance of the inner encoding, i.e., in respect to a database of external pre-computed pRG expansions, the PRG method actually also uses as complementary auxiliary seed a pseudo-randomly generated bit-string $\lambda_j^{(auxi)}$ of 256 bits, equal across all wires but equal within a circuit index.) The inner BitComs (cell F7 in Table 3.5), along with all GCs and other elements from other wires, contribute to the pre-image of the global hash, used to create the RSC commitment sent to P_B. Since the VERIFY stage will allow P_B to learn the two keys per output wire, the respective inner encodings, must not link directly to the outer encodings (i.e., to the two randomnesses of the outer BitCom) — otherwise P_B would be able to open any bit from the output BitComs, and also any pair of such encodings constitutes a trapdoor that allows P_B to extract the private input bits of P_A.

Reveal for *check*. In the RESPOND stage, the RSC seed λ_j opened for each check index $j \in J_E$ allows P_B to reconstruct the GCs, and respective inner BitComs, which can be used to construct the pre-image of the global hash also opened by P_A and verify its correctness.

Reveal for *evaluation*. Also in the RESPOND stage, P_A reveals the two multipliers $(\beta_{j,i,0}^{(0)}, \beta_{j,i,1}^{(0)})$ (both being randomness for committing bit 0) that lead the respective inner (independent) encodings $\nu_{j,i,c}^{(c)}$ into the two possible outer (non-trivially correlated) encodings $\mu_i^{(c)}$. Since P_B only learns one key ($k_{j,i}^{[b_i]}$) per output wire, which enables computation of the respective inner encoding ($\nu_{j,i,b_i}^{(b_i)}$), it can only use one multiplier ($\beta_{j,i,b_i}^{(0)}$) to obtain a respective outer encoding ($\mu_i^{(b_i)}$). Nonetheless, P_B is able to use the XOR-homomorphic properties of the BitCom scheme to verify the correctness of both multipliers, namely by checking that they are both encodings of 0 and that their respective BitComs lead the respective inner BitComs into the outer BitCom (μ'_i).

3.3.4 A remark on key selection

For simplicity, the garbled circuit generation mechanism ($\mathrm{PRGen_{GC}}$) was assumed to accept as input any list of pairs $(k_{j,i}^{[0]}, k_{j,i}^{[1]})$ of input wire keys that the garbled circuit should have, namely a list of pairs of pseudo-random keys. This allows that, for the connectors of input of $\mathrm{P_B}$, keys can be pseudo-randomly generated from group elements $\nu_{j,i}^{(c)}$. However, for certain optimized techniques (e.g., "free-XOR" [KS08b] and "two-halves make a whole" [ZRE15]) the input keys may have to be somehow correlated, e.g., such that the two keys of each input wire differ by a global offset common across all wires of a garbled circuit. For the sake of generality, it is here noted that any such correlation between input keys can be incorporated within the garbled circuit generation mechanism ($\mathrm{PRGen_{GC}}$) at the expense of at most only one extra unary garbled-gate (for Boolean function EQUAL) per input wire. (In the original forge-and-lose paper [Bra13], this mechanism had been explicitly described outside of the gabled-circuit generation, using elements called "widgtets"). Essentially, each input wire in each GC can start with an EQUAL garbled gate that converts each random outer input-key (e.g., the PRG expansion of a group element) to a new inner input-key correlated as needed (e.g., satisfying a global offset with the complementary inner input-key). Such garbled gate can be simply composed of only two cipher-texts, each of them being an enciphering of the needed inner key value, using as key the outer input-key. For this reason, when benchmarking the communication complexity associated with garbled circuits the size explicitly accounts for two extra cipher-texts per input wire of $\mathrm{P_B}$. This is not needed for connectors of input wires of $\mathrm{P_A}$ because there the connectors allow $\mathrm{P_A}$ to commit to any pair of input keys (in a randomly permuted order). For the output wires of $\mathrm{P_B}$ the EQUAL gate trick could also be used, in order to ensure that the circuit output keys must exceptionally reveal their underlying bit. However, since most garbling mechanisms already trivially allow ensuring this property (e.g., by including in each internal wire key an *indicator bit* that informs how to use thekey in subsequent gates), the protocol description directly assumes that there is a predicate ϵ that extracts the underlying bit from each output wire, and the communication benchmark does not consider an overhead for output wires of $\mathrm{P_B}$.

3.4 Complexity analysis

This section analyzes the complexity of the S2PC-with-Coms protocol, under different types of cryptographic instantiations and parametrizations. §3.4.1 describes IFC and DLC instantiations for BitCom schemes and NIZK sub-protocols. §3.4.2 analyzes the concrete communication complexity of different components of the protocol, for S2PC-with-Coms of an AES-128 circuit and also for a SHA-256 circuit. §3.4.3 analyzes the computational complexity, with a focus on group operations (multiplications and exponentiations).

3.4.1 Instantiations of BitCom Schemes, NIZKPs and NIZKPoKs

Table 3.6 exemplifies instantiations of outer BitCom schemes ($\mathscr{B}_{\mathrm{A}}$, $\mathscr{B}_{\mathrm{B}}$), intermediate BitCom schemes ($\mathscr{B}_{\mathrm{ConA}}$, $\mathscr{B}_{\mathrm{FLA}}$, $\mathscr{B}_{\mathrm{FLB}}$, $\mathscr{B}_{\mathrm{OT}}$) and respective associated NIZKPs and NIZKPoKs. The schemes are considered across two main types of instantiation: integer-factorization cryptography (IFC) and discrete-log cryptography (DLC) (row 1). They correspond to respective different types of cryptographic-intractability assumption: decisional quadratic-residuosity (DQR), including integer factorization; and decisional Diffie-Hellman (DDH), including discrete-log computation. Associated with each intractability assumption there is a respective type of group. The IFC instantiation is based on multiplicative groups over the integers modulo a Blum integer, and with the trapdoor being the Blum-integer factorization. The DLC instantiation, itself sub-instantiable by regular multiplicative groups modulo a large prime number, a.k.a. finite-field cryptography (FFC), and by elliptic curves over finite fields, a.k.a. elliptic-curve cryptography (ECC), has as trapdoor the discrete log between two base generators. The acronyms IFC, FFC and ECC were suggested in [Bar16].

The IFC instantiation considered in the original paper [Bra13] is here reviewed with the new protocol structure (e.g., Ext-Coms in the output) and updated constructions (e.g., more efficient Equiv-Com); the DLC case takes advantage of the more generalized description of connectors that has been considered in this dissertation. The side-by-side comparison of IFC and DLC instantiations facilitates the highlighting of tradeoffs between communication and computation, while also illustrating different concrete constructions of connectors.

The exemplified instantiations are based on four BitCom schemes: Blum and Pedersen as unconditionally hiding and equivocable, respectively for IFC and DLC; GM and ElGamal as unconditionally binding and extractable BitCom schemes, respectively for IFC and DLC.

Table 3.6: ZK sub-protocols per (IFC and DLC) example instantiations of BitComs

A	B	C	D	E	
Type of instantiation / assumption / group elements			IFC / DQR / Blum integers	DLC / DDH / FFC or ECC	1
For $p \in \{A,B\}$: BitCom scheme ($\mathscr{B}_p$) for **outer** BitComs of P_p (P_p as sender; $P_{\bar{p}}$ as receiver)	Type of BitComs (not opened)		**GM** (Ext)	**ElGamal** (Ext)	2
	NIZKP of good BitComs		Not needed	$\text{NIZKP}_{\text{GoodElGBitComs}}(\mathscr{B}_p)$	3
	Parsing of outer BitComs into a single BitString Com **per wire type**		Vector of GM BitComs ($\mathscr{B}_p$)	Homomorphic combination into a single ElGamal Com ($\mathscr{C}_p$)	4
	NIZKPoK of opening (bits and respective randomness) of outer BitStringComs	If GCRS	$\text{NIZKPoK}_{\text{PseudoSqrts}}(\mathscr{B}_p)$	$\text{NIZKPoK}_{\text{ElGOpening}}(\mathscr{C}_p)$	5
		If GPKI	$\text{NIZKPoK}_{\text{BI-trapdoor}}(\mathscr{B}_p)$ (a single time)		6
For outer-Com permutations	NIZKP of Coms to 0	Any setup	$\text{NIZKP}_{\text{GM-All-0s}}(\mathscr{B}_p)$	$\text{NIZKP}_{\text{ElG-All-0s}}(\mathscr{C}_p)$	7
	NIZKPoK of randomness	If GCRS	$\text{NIZKPoK}_{\text{PseudoSqrts}}(\mathscr{B}_p)$	$\text{NIZKPoK}_{\text{ElGOpening}}(\mathscr{C}_p)$	8
		If GPKI	Not needed (cell D6 is enough)		9
BitCom scheme ($\mathscr{B}_{\text{FLA}}$) for **intermediate** F&L BitComs of input of P_A ($i \in I_A$)	Type of BitComs		**GM** (Ext)	**ElGamal** (Ext)	10
	Knowledge of trapdoor (t_{FL})		Known to P_A		11
	Ensure good pair of dual BitCom schemes ($\mathscr{B}_{\text{FLA}}$, $\mathscr{B}_{\text{FLB}}$)	Any setup	(It is assumed that the public parameters of $\mathscr{B}_{\text{FLB}}$ can be derived from those of $\mathscr{B}_{\text{FLA}}$, and vice-versa; thus, either a NIZKP of good $\mathscr{B}_{\text{FLA}}$ or a NIZKP of good $\mathscr{B}_{\text{FLB}}$ is enough, if needed)		12
		If GCRS	($\mathscr{B}_{\text{FLA}} \neq \mathscr{B}_A$:) $\text{NIZKP}_{\text{GoodBlumInt}}(\mathscr{B}_{\text{FLA}})$	Not needed (assuming that DLC parameters do not require ZKP of correctness)	13
		If GPKI	($\mathscr{B}_{\text{FLA}} = \mathscr{B}_A$:) Not needed		14
	ZKP of consistency across BitComs	If GCRS	($\mathscr{B}_{\text{FLA}} \neq \mathscr{B}_A$:) $\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_A,\mathscr{B}_{\text{FLA}}]$		15
		If GPKI	($\mathscr{B}_{\text{FLA}} = \mathscr{B}_A$:) Not needed		16
BitCom scheme ($\mathscr{B}_{\text{ConA}}$) for connectors of input of P_A ($i \in I_A$)	If $\mathscr{B}_{\text{ConA}} \equiv \mathscr{B}_A$		Nothing else required (row 3 is enough)		17
	If ShortBitCom optimization, with $\mathscr{B}_{\text{ConA}} \neq \mathscr{B}_A$ (§B.4.1.3)	Type of BitComs	**Blum**	**Pedersen**	18
		Prove good scheme	$\text{NIZKP}_{\text{GoodBlumInt}}(\mathscr{B}_{\text{ConA}})$	Not needed (argument as in D13–14)	19
		Prove consistent BitComs	$\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_A,\mathscr{B}_{\text{ConA}}]$	$\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_A,\mathscr{B}_{\text{ConA}}]$	20
If $\mathscr{B}_A \neq \mathscr{B}_{\text{ConA}} \wedge \mathscr{B}_A \neq \mathscr{B}_{\text{FLA}} \wedge \mathscr{B}_{\text{ConA}} \neq \mathscr{B}_{\text{FLA}}$			$\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_A,\mathscr{B}_{\text{ConA}},\mathscr{B}_{\text{FLA}}]$ instead of the ZKPs in rows 20 and 15		21
BitCom scheme ($\mathscr{B}_{\text{OT}}$) for **intermediate** BitComs of input of P_B ($i \in I_B$), to sustain OT	Type of oblivious transfer		2-out-of-1 OT (§3.3.2.1)	1-out-of-2 OT (§3.3.2.2)	22
	Type of BitComs		**Blum** (**Equiv**)	**ElGamal** (**Ext**)	23
	Knowledge of trapdoor t_{OT}		**Known** to P_A	**Known** to P_B	24
	Prove good BitCom scheme	If GCRS	($\mathscr{B}_{\text{OT}}$=$\mathscr{B}_{\text{FLB}}$ is dual of $\mathscr{B}_{\text{FLA}} \neq \mathscr{B}_A$:) Cell D13 is enough (assuming D12)	($\mathscr{B}_{\text{OT}} \notin \{\mathscr{B}_B,\mathscr{B}_{\text{FLB}}\}$:) Not needed (argument as in E13)	25
		If GPKI	($\mathscr{B}_{\text{OT}}$=$\mathscr{B}_{\text{FLB}}$ is dual of $\mathscr{B}_A$=$\mathscr{B}_{\text{FLA}}$:) Not needed (implied by trusted setup, assuming D12)	($\mathscr{B}_{\text{OT}}$=$\mathscr{B}_B$:) Not needed (argument as in E14	26
	Prove consistent BitComs ($\mathscr{B}_{\text{OT}}$ vs. $\mathscr{B}_B$)	If GCRS	($\mathscr{B}_{\text{OT}} \neq \mathscr{B}_B$:) $\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_{\text{OT}},\mathscr{B}_B]$	($\mathscr{B}_{\text{OT}} \neq \mathscr{B}_B$:) $\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_{\text{OT}},\mathscr{B}_B]$	27
		If GPKI		($\mathscr{B}_{\text{OT}} = \mathscr{B}_B$:) Not needed	28
BitCom scheme ($\mathscr{B}_{\text{FLB}}$) for **intermediate** BitComs of output of P_B ($i \in O_B$), to sustain F&L	Type of BitComs		**Blum** (**Equiv**)	**Pedersen** (**Equiv**)	29
	Relation to other BitCom schemes		Same as $\mathscr{B}_{\text{OT}}$, dual of $\mathscr{B}_{\text{FLA}}$	Different from $\mathscr{B}_{\text{OT}}$, dual of $\mathscr{B}_{\text{FLA}}$	30
	Prove correct and consistent BitComs ($\mathscr{B}_{\text{FLB}}$ vs. $\mathscr{B}_B$)	By P_A:	Not needed (verified via cut-and-choose and homomorphic properties of multipliers)		31
		By P_B:	$\text{NIZKP}_{\text{SameComBits}}[\mathscr{B}_B,\mathscr{B}_{\text{FLB}}]$ (see (597))		32

Note: given the correctness of outer BitComs (row 3), the several $\text{NIZKP}_{\text{SameComBits}}$ are assumed to also prove correctness of the involved intermediate BitComs (i.e., those that would otherwise require such a proof) — this is relevant in cells E15, E20, E21. The transcript produced by P_B for any of the NIZKPoKs in rows 8 and 9 is committed (using an Ext-and-Equiv Com) as part of the coin-flipping of outer-Com permutations.

The trusted setup (§2.2.2) can have one of two possible types: global-PKI with local-CRS, and global-CRS with local-CRS, respectively labeled simply as GPKI and GCRS. The global setup defines the outer commitment scheme parameters. The underlying NIZKPs, NIZKPoKs and Ext-and-Equiv Com scheme are defined in Appendix based on an ideal Ext-Com, an ideal Equiv-Com and a global NPRO, which enable the respective simulatability.

3.4.1.1 Outer BitComs for input and output bits

The outer BitCom schemes ($\mathscr{B}_{\mathrm{A}}$, $\mathscr{B}_{\mathrm{B}}$) are defined as extractable, being instantiable via GM BitComs and ElGamal BitComs, respectively for IFC and DLC (row 2). While these schemes are unconditionally binding, when using their base definition, in practice the open phase is not being used for the outer BitComs. Thus, in practice, externally to the S2PC-with-Coms protocol the BitComs can have an open phase different from a simple disclosure of its "randomness" and of the committed value. For example, the external open phases may instead be performed by disclosing the committed bit and then giving a (non-malleable) ZK argument that the disclosed value is the correct one — in this case the schemes become equivocable and no longer unconditionally binding. This alternative opening is useful to ensure, if desirable, composability when considering external protocols and concurrent executions.

The initial outer BitComs produced by each party may require a NIZKP of correctness, depending on the instantiation (row 3). ElGamal BitComs require such a NIZKP, because then cannot be directly verified as correct — a commitment to a non-bit is (per DDH assumption) computationally indistinguishable from a BitCom

GM BitComs can be determined as correct by simply checking group membership in the respective multiplicative group, which can be done without knowing the trapdoor.

The initial outer BitComs σ'_i are parsed (stringized) into a single BitString commitment σ'_{set}, for each wire type ($set \in \{I_{\mathrm{A}}, I_{\mathrm{B}}, \mathrm{O}_{\mathrm{B}}\}$) (row 4). In the IFC instantiation based on GM BitComs this is just a logical parsing, as the BitString Com is simply a vector of the original BitComs. (A conceivable improvement based on Paillier encryption is not described herein.) However, in the DLC instantiation all the ElGamal BitComs (per wire type) are homomorphically combined into a single ElGamal BitSring commitment, thus reducing the overall size of the outer commitment. The parsing is described in the stages Parse Outer BitComs of P_{B} and Parse Outer BitComs of P_{A} of the protocol. The randomness σ_{set} associated with each BitString Com is also parsed by the respective party,

from the respective randomness σ_i of BitComs (with $i \in set$).

For the purpose of simulatability of S2PC-with-Coms, the simulator needs to extract the committed bits and the "randomnesses" used to produce the initial outer commitments. In the global-CRS model, the simulator does not have any trapdoor so it needs to obtain the bits and randomness directly from respective NIZKPoKs (row 5). In the IFC case this corresponds to a NIZKPoK of pseudo-square-roots of the GM BitComs that constitute the vector of BitComs. The DLC instantiation can be resolved with a NIZKPoK of an ElGamal opening, which allows the simulator to extract the randomness and the committed value, even without the prover knowing the respective trapdoor.

In the global-PKI model, for IFC it is enough to send a single NIZKPoK of Blum integer trapdoor §A.2.3, from which the simulator gains the ability to subsequently extract pseudo-square-roots from any square and decrypt GM BitComs. For DLC the committer party sends along a NIZKPoK of ElGamal opening (§A.3.4) for each initial outer-Com. If for a particular instantiation the NIZKPoK does not get cheaper when the prover knows the trapdoor, then the scheme is the same regardless of CRS or PKI setting. (row 6). It is worth noticing that in DLC the trapdoor does not allow extraction of the "randomness" of commitments, because a discrete-log computation would still be required from the simulator.

Outer-Com permutations. The coin-flipping that determines outer-Com permutations is asymmetric in the view of each party. The contribution that each party gives to decide the permutation is an outer-Com of zeros for her own wires, and is an outer-randomness for the wires of the other party. For this reason, each party also gives, in respect to her own wires, a NIZKP of commitments to an all-zeros bit-string (row 7). Furthermore, for the purpose of simulatbaility (since the simulator needs to know the final outer randomness), each party also gives a NIZKPoK of outer-randomness associated with the contribution to the permutation of her own wires. In a GCRS model this is explicitly required for each outer-com (row 8), whereas in a GPKI the requirement is reduced if a NIZKPoK of trapdoor has already been provided. Specifically, in IFC the new NIZKPoK is avoidable, because the trapdoor suffices to extract square-roots, whereas in DLC it is enough to do a new NIZKPoK of DL of the first component of the ElGamal bit-string Com for each wire type (row 9).

3.4.1.2 Intermediate BitComs of P_A

Intermediate BitCom scheme for forge-and-lose. The forge-and-lose technique is sustained on a BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) used to commit the input bits of P_A, with a trapdoor that may be disclosed to P_B if P_A misbehaves in respect to the output of P_B. The scheme is thus required to be extractable (Ext) (row 10), and with an extraction-trapdoor explicitly known by P_A (row 11) (but see Remark 3.3).

The forge-and-lose technique is actually sustained by a pair ($\mathscr{B}_{\mathrm{FLA}}$, $\mathscr{B}_{\mathrm{FLB}}$) of dual BitCom schemes, because it also involves Equiv-Coms of the output bits of P_B. Thus, some of the ZK sub-protocols related to the scheme used for input bits of P_A also serve to the scheme used for output bits of P_B. Specifically, it is assumed that the parameters of the later scheme can be derived deterministically from the parameters of the former (row 12).

If the outer BitCom scheme $\mathscr{B}_A$ is defined by a GCRS (and thus with trapdoor unknown to any party) then the intermediate BitCom scheme $\mathscr{B}_{\mathrm{FLA}}$ with trapdoor known to P_A is necessarily different from the one used for outer BitComs. The correctness of the new scheme is verified by P_B, e.g., either via a NIZKP of correctness of the parameters (e.g., correct Blum integer in IFC), or if possible by a local verification without trapdoor (e.g., in DLC, verifying that the proposed generators are in the expected group) (row 13).

In the case of a GPKI setup defining the parameters of the outer BitCom scheme $\mathscr{B}_A$ with trapdoor of P_A, the schemes can (but do not need to) be the same, thus avoiding an extra NIZKP of correct intermediate BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) (row 14).

If the schemes are different (e.g., necessarily in GCRS), then a NIZKP is required to prove that the bits committed by the two schemes are the same (row 15). If the schemes are the same, and since there is no duplication of BitComs, said NIZKP is not needed (row 16).

While homomorphic properties are not inherently essential to support the forge-and-lose technique, they may facilitate the efficiency of the NIZKP of consistency of committed bits.

Intermediate BitCom scheme for connectors of input of P_A. In comparison, the BitCom scheme $\mathscr{B}_{\mathrm{ConA}}$ used to support the connectors of input of P_A requires the homomorphism, but it is irrelevant whether or not a trapdoor is available. The scheme can be the same or different — the use of a different BitCom scheme may be motivated by communication efficiency. For example, the (optional) support for an optimization based on "short-term binding"

commitments (described in §B.4.1.3) requires a different scheme, with shorter parameters, necessarily unconditionally hiding (row 18). Since the binding of the intermediate BitComs used for connectors of input of P_A only needs to be ensured during the protocol execution, it is possible to improve communication by using an unconditionally hiding BitCom scheme with shorter security parameter (e.g., for 96 bits of cryptographic security) that during the protocol execution ensures the binding of P_A to the respective connectors. The gain must be weighed against the cost of the additional BitComs and NIZKPs.

If the scheme ($\mathscr{B}_{\text{ConA}}$) for connectors is different from the outer scheme $\mathscr{B}_A$, then the correctness of the new scheme needs to be validated, e.g., by means of a NIZKP in the IFC case or by local verification in the DLC case (row 19). Also, if the scheme is different the consistency of the BitComs across different schemes also needs to be proven via a respective NIZKP (row 20). If, instead, the two schemes ($\mathscr{B}_A$, $\mathscr{B}_{\text{ConA}}$) are the same (row 17), then no new NIZKP or NIZKPoK is required in regard to the connectors of input of P_A.

If all three schemes ($\mathscr{B}_A$, $\mathscr{B}_{\text{FLA}}$, $\mathscr{B}_{\text{ConA}}$) associated with input bits of P_A are different, then the NIZKP of consistent committed bits can be done only once across the three schemes, instead of two times (once for each intermediate scheme) (row 21).

3.4.1.3 Intermediate BitComs of input of P_B

For input bits of P_B, IFC uses a 2-out-of-1 OT, whereas DLC uses a 1-out-of-2-OT (row 22). The former depends on an Equiv BitCom scheme with trapdoor known by P_A, whereas the first uses an Ext BitCom scheme with trapdoor known by P_B (rows 23 and 24).

In the GCRS case, where the trapdoor of outer commitments is not known by any party, the OT BitCom scheme needs to be different (row 25). In the IFC instantiation this may hinder the reduction of number of rounds of the protocol, because the parameters of the OT BitCom scheme need to be provided by P_A, along with a respective NIZKP of correctness. Conversely, in the DLC case the parameters supporting the 1-out-of-2 OT are instead decided by P_B, and so do not require an extra communication round. Furthermore, they also do not require a NIZKP of correctness, if group membership can be decided without the trapdoor.

In the GPKI model each party already knows and trusts the public parameters of the other party are correct (row 26). Thus, even in the IFC case the public parameters associated with P_A can be directly used by P_B to produce BitComs for the 2-out-of-1 OT. With DLC parameters, the definition of the OT BitCom scheme ($\mathscr{B}_{\text{OT}}$) also does not require prior

interaction with P_A, because P_B uses his own commitment scheme defined by the GPKI.

If the intermediate BitCom scheme $\mathscr{B}_{OT}$ is different from the outer BitCom scheme $\mathscr{B}_B$, then a NIZKP is needed to prove consistency of the bits committed by the different BitCom schemes. This is necessarily the case for a GCRS setup (row 27). With a GPKI model this is only needed for an IFC instantiation (using a 2-out-of-1 OT), being avoided in the DLC instantiation (using a 1-out-of-2 OT) (row 28).

3.4.1.4 Intermediate BitComs of output of P_B

The intermediate BitCom scheme $\mathscr{B}_{FLB}$ used for output wires of P_B, simultaneously supporting the connectors and the forge-and-lose technique, is equivocable in both IFC and DLC instantiations (row 29). It is the dual of the BitCom scheme $\mathscr{B}_{FLA}$ used to support the forge-and-lose technique in regard to the input bits of P_A (row 30). In the IFC instantiation it can be the same as the OT BitCom scheme $\mathscr{B}_{OT}$ used for intermediate BitComs of input of P_B, but for DLC it is different.

The intermediate BitComs for the output wires of P_B do not require a NIZKP of correctness, because of the cut-and-choose approach (that allows checking correctness of inner BitComs of check instances) and the homomorphic properties (that allow checking correctness of a correct relation between inner and intermediate BitComs) (row 31). Since for output bits of P_B the outer BitCom scheme $\mathscr{B}_B$ and the intermediate BitCom scheme $\mathscr{B}_{FLB}$ are different, a NIZKP is still required for P_B to prove consistency of the bits committed by the final outer BitComs and the obtained intermediate BitComs (row 32).

3.4.2 Communication complexity

This subsection estimates the communication complexity of S2PC-with-Coms of two different circuits, assuming a global PKI setup and 128 bits of cryptographic security. The reported values are based on instantiations with: group elements with 3,248-bits for IFC and FFC and 264 bits for ECC; with pre-images with 3,248 bits (square-roots) for IFC and 256 bits (exponents) for FFC and ECC; with CR-hashes, PRG seeds and garbled gates with 256 bits.

The first circuit implements AES-128 (the *advanced encryption standard* blockcipher, with key and plaintext with 128-bits each), using 6,800 multiplicative gates [Bri13] and 128 wires for the input of each party, and 128 wires of output of P_B to hold the enciphering

of the input of P_B using as key the input of P_A, which uses optimized components with state-of-the-art circuit optimization [BMP13]. The second circuit implements SHA-256 (the *secure hash algorithm*, with 512-bit input), using 90,825 multiplicative gates [Bri13] and 256 wires for the private input of each party and 256 wires of output of P_B to hold the (SHA-256) hash of the concatenated inputs.

Comparing configurations. The concrete estimated communication values are summarized in Table 3.7, in total and aggregated per type of protocol component. It shows, side-by-side, IFC and DLC (FFC and ECC) instantiations (row 1), several levels of statistical security ($\sigma \in \{40, 96, 128\}$) (row 2), and different cut-and-choose configurations (rows 3–6). The case of 40 bits of statistical security (columns D–I) is a common benchmark, large enough for practical purposes where the cut-and-choose partition is defined by the receiver (P_B) after the sender has committed to the protocol elements. If the cut-and-choose partition is decided non-interactively by P_A, e.g., as a NPRO hash of an Equiv-Com, as a way to reduce the number of communication rounds, then the statistical security parameter should be increased up to an equivalent cryptographic security parameter. The cases of 96 and 128 bits of statistical security (columns J–K) are considered for short and long durations of protocol execution, where the former can be considered if the Equiv-Com includes in its pre-image a fresh nonce that was learned only in the protocol. Each level of statistical security can be achieved with different numbers of garbled circuits (row 4) and conditions on the number of check and evaluation instances (rows 5 and 6).

Total communication. To securely evaluate the exemplified AES 128 circuit, with 128 bits of cryptographic security, 40 bits of statistical security and using 41 GCs out of which at most 20 are evaluated, the protocol requires about 11, 8 and 5 Mega bytes, for IFC, FFC and ECC instantiations, respectively (row 13). The corresponding communication for the SHA-256 circuit is about 70, 66 and 60 Mega bytes (row 22). By tweaking the cut-and-choose parameters, the same security can be obtained without exceeding 8 evaluation circuits, but now with up to 115 check circuits (in this example, the overall 123 circuits equates the minimal number of circuits that would be needed without the forge-and-lose technique). ECC yields the best results, requiring about 2.2 and 24 Mega bytes, respectively for S2PC-with-Coms of AES-128 and SHA-256 (column I).

Table 3.7: Communication of S2PC-with-Coms of AES-128 and SHA-256

A	B	C	D	E	F	G	H	I	J	K	
Parametrization	Type of cryptographic instantiation		IFC	FFC	ECC	IFC	FFC	ECC	ECC		1
	Bits of Statistical security (σ)		40			40			**96**	**128**	2
	C&C param-eters	Condition on # eval instances	$1 \le e \le \sigma/\mathbf{2}$			$1 \le e \le \sigma/\mathbf{5}$			$1 \le e \le \sigma/\mathbf{5}$		3
		$s \equiv$ # GCs	41			123			272	365	4
		$(v_{\min}; v_{\max})$ (# check circuits)	(21; 40)			(115; 122)			(253; 271)	(340; 364)	5
		$(e_{\min}; e_{\max})$ (# eval circuits)	(1; **20**)			(1; **8**)			(1; **19**)	(1; **25**)	6
AES-128 circuit	Set initial outer BitComs (kB) (including needed NIZKPs and (NI)ZKPoKs)		582	1,171	169	582	1,171	169	169		7
	Set intermediate BitComs (kB) (including needed NIZKs)		274	52	4.2	274	52	4.2	4.2		8
	C&C of GCs	Connectors (kB)	4,289	2,366	457	1,716	947	183	434	571	9
		Communicated GCs (kB)	4,434			1,774			4,212	5,542	10
		Communicated RSC seeds and RSC Equiv Com and opening (kB)	1.5	1.1	0.77	4.5	4.2	3.8	8.2	11	11
	Coin-flip and permutation of outer Coms (kB)		1,128	240	56	1,128	240	56	56	56	12
	Total communication (kB)		10,709	8,265	5,120	5,479	4,188	**2,189**	4,883	6,353	13
	(Not GCs) / Total		59%	46%	13%	68%	58%	19%	14%	13%	14
	Total commun. if local-PKI and simple S2PC		8,984	7,734	5,012	3,753	3,657	**2,081**	4,775	6,245	15
SHA-256 circuit	Set initial outer BitComs (kB) (including needed NIZKPs and (NI)ZKPoKs)		738	2,156	294	738	2,156	294	294		16
	Set intermediate BitComs (kB) (including needed NIZKs)		430	104	8.5	430	104	8.5	8.5		17
	C&C of GCs	Connectors (kB)	8,569	4,732	913	3,428	1,893	365	867	1,140	18
		Communicated GCs (kB)	58,292			23,317			55,377	72,865	19
		Communicated RSC seeds and RSC Equiv Com and opening (kB)	1.5	1.1	0.77	4.5	4.2	3.8	8.2	11	20
	Coin-flip and permutations of outer Coms (kB)		1,856	240	56	1,856	240	56	56	56	21
	Total communication (kB)		69,887	65,525	59,563	29,773	27,714	**24,044**	56,611	74,375	22
	(Not GCs) / Total		17%	11%	2.1%	22%	16%	3.0%	2.2%	2.0%	23
	Total if local-PKI and simple S2PC (kB)		67,382	64,890	59,447	27,269	27,079	**23,928**	56,494	74,259	24

Group parameters: See a finer-grained analysis in Tables B.4 and B.5. IFC and FFC use 3,248-bit group elements, whereas ECC uses 256-bit elements; IFC uses 3,248-bit pre-images (modular square-roots), whereas FFC and ECC use 256-bit pre-images (exponents). **Circuit parameters:** AES-128 and SHA-256 use respectively 6,800 and 90,825 non-linear Boolean gates; AES-128 has 128 wires for the input of each party and for the output of P_B, whereas SHA-256 has 256 wires for the input of each party and 256 wires for the output of P_B. **Legend:** kB (Kilo bytes, i.e., a thousand of bit octets); "Not GCs" (all elements that do not arise from sending evaluation GCs, e.g., BitComs, NIZKPs, (NI)ZKPoKs, connectors, coin-flipping, RSC seeds and Equiv-Com).

Connectors. Across the three types of instantiation (IFC, FFC, ECC), IFC is the one with most expensive connectors, namely because of the connectors of input and output of P_B, for which it communicates a quadratic number of group-elements (squares or square-roots), i.e., proportional to the product of the statistical security and the number of evaluation

instances. The FFC instantiation is the second largest, with the main contribution coming from connectors of input wires of P_B (with communication of large group elements), but much shorter communication for other connectors, which involve communication of shorter pre-images. The ECC instantiation has short connectors for all wire types.

Outer commitments. Interestingly, the communication needed for setting the outer commitments (rows 7 and 16) is larger for FFC than for IFC. This is due to the larger communication complexity in FFC associated with producing the needed NIZKPoKs of ElGamal openings and NIZKPs of correct ElGamal BitComs.

Overhead from "not GCs". A comparison of values across the two circuits exemplifies how the proportional overhead from "not GC" elements (row 14 for AES-128 and row 23 for SHA-256) reduces with the increase of the proportion of number of garbled gates over the number of outer wires. For example, in regard to the configuration with only 8 evaluation circuits (columns G–I), in the AES-128 circuit the number (384) of outer wires divided by the number (6,800) of garbled gates is about 5.6%, whereas for the SHA-256 circuit it is about 0.85%. Correspondingly, in the IFC instantiation in column G the overhead from non-GC elements decreases from about 68% to 22%; similarly, in the ECC instantiation in column I the reduction is from 19% to 3.0%. In spite of the relatively large contribution of NIZK sub-protocols and connectors, in all except one of the cases shown in the Table the communication proportion due to the non-GC elements is less than two thirds. When the number of evaluation circuits decreases (e.g., columns G–I in comparison with columns D–F), the proportion increases (rows 14 and 23) but the concrete communication decreases — this is because the communication due to GCs decreases at a faster rate.

Overhead from "GCs". The significance of the communication contribution of GCs is unavoidable as the underlying circuit size increases, as specially noticeable with the SHA-256 estimation (row 22). Still, the RSC technique also allows a very significant reduction in communication, by virtue of reducing the number of communicated garbled circuits. This is at the computational expense of over-increasing the number of *check* instances. When the overall number of garbled circuits is minimal (columns D–F), a simple S2PC based on a traditional cut-and-choose, i.e., without the forge-and-lose technique, would require at least the triplet of the garbled circuits, besides overheads from other elements. For tradeoffs

that reduce the number of evaluation circuits, the comparison is even more favorable. For example, without the forge-and-lose the use of 123 circuits (columns G–I) for 40 bits of statistical security require at least 49 evaluation circuits, i.e., more than six times more than the sufficient eight circuits when using the forge-and-lose.

For the applications of oblivious cipher or hash evaluation, the communication complexity can conceivably improve greatly by changing the underlying cipher of hash function. For example, Albrecht et. al [ARS+15] propose a family ("LowMC") of low multiplicative-complexity ciphers, including examples for 128 and 256 bits of security (and key-size and block-size), respectively requiring more than eight times fewer and three times fewer multiplicative gates than AES-128. A cryptographic hash could also be based of some of those ciphers, thus with much less multiplicative complexity than SHA-256. Within the cut-and-choose of garbled circuits approach, another conceivable improvement may be derived by devising new protocols with better statistical guarantees that enable further reduction of the overall number of garbled circuits or garbled gates, e.g., via linked executions as hinted in §6.3.1.

Total if simpler S2PC. Rows 15 and 24 show the total communication that would be required in case of a simple S2PC without interest for the outer commitments, and (in case of IFC) if allowing a local PKI that would avoid the need for NIZKPoK of trapdoors and/or of openings, or similarly in case of global PKI where the NIZKPoK of trapdoor would already have occurred external to the protocol. For example, the differential for the IFC instantiation would be of more than 1.7 Mega bytes, whereas for ECC would be of just about 0.11 Mega bytes. (Note: the actual differential could be made smaller for IFC if using larger cut-and-choose parameters for the Ext-and-Equiv-Com, allowing a smaller communication rate.)

Further analysis. Appendix B.5 contains a more detailed analysis of communication complexity, including concrete parameter sizes for 128 bits of cryptographic security (Table B.1), the size of each NIZK sub-protocol (Table B.2), the directional communication associated with connectors (Table B.3), and a finer-grained analysis of the concrete communication of S2PC-with-Coms of the AES-128 and SHA-256 circuits (Tables B.4 and B.5). A better insight about the major contributions of overhead (i.e., besides garbled circuits) motivates pursuing further improvements, some of which are suggested in Section 6.3.

3.4.3 Computational complexity

The time duration of an S2PC execution is of practical interest for its applicability in the real world. In the lack of a concrete implementation of the protocol described herein, a good related metric is the computational complexity, as a measure of the required number of certain primitive operations in relation to the parameters that define the problem specification, (e.g., the circuit being obliviously evaluated and the type of commitments) and the security parameters. The time of an execution is then a function of the available computational power (number of operations per unity of time) per processing unit, of the number of simultaneously available processing units, of the parallelizability of needed operations, and of the time taken for communications.

This subsection makes an estimation of the asymptotic number of main group operations (i.e., associated with BitCom schemes) required for the defined S2PC-with-commitments protocol. The estimate is made as a function of the parameters of interest: the number ℓ of outer wires (i.e., of input and output) of each circuit; the statistical security parameter σ or the related number s of circuits used in the cut-and-choose; the cryptographic security parameter κ. The terms "unitary," "linear" and "quadratic," with respect to number of some type of operation, are expressed with the meaning that the actual number of operations is upper bounded by a constant function, a linear function and a quadratic function, respectively, of one (or of a linear combination) of the mentioned parameters.

Of significance is the measure of operations (multiplications) required in quadratic number (e.g., number of outer wires times number of circuits) or those (exponentiations) required in linear number (e.g., linear in the number of wires, and/or in number of circuits) but nonetheless expensive. The original IFC-based S2PC-with-BitComs protocol [Bra13] was designed to not exceed a linear number of exponentiations. This was in contrast with other DLC protocols whose required number of exponentiations by both parties is proportional to the number of circuits multiplied by the number of input wires (e.g., [LP11]), though in compensation those exponentiations are supported in groups with smaller moduli length and sub-groups of smaller order. The linear number of exponentiations is retained in this dissertation, even though more BitCom schemes and NIZKs are involved. The new DLC instantiation requires a quadratic number of exponentiations, but with possible improvements for certain types of wire.

In summary: the IFC instantiation requires a number of exponentiations linear in the

number of input bits of P_A and in the statistical security parameter, and a number of multiplications and multiplicative-inversions proportional to the product of the number of outer wires (input and output) and the statistical security parameter; the DLC instantiation requires a quadratic number of exponentiations (but improvable with certain optimizations).

Connectors of input of P_A. In IFC, the connectors of input of P_A based on BitComs require a unitary number of multiplications for each connector. In DLC, the bit-string-Com construction requires one exponentiation for each circuit, with each such exponentiation accounting at once for all input bits of P_A.

Connectors if input of P_B. The IFC-based 2-out-of-1 OT requires P_A to compute one square-root per input bit of P_B, which is approximately computationally equivalent to one exponentiation modulo each prime factor. The number is independent of the number of garbled circuits and is amortizable across multiple executions that use the same input. The number of multiplications is quadratic, i.e., unitary per input wire of each circuit.

For DLC the described 1-out-of-2 OT requires a unitary number of exponentiations for each connector, i.e., overall quadratic (linear in the number of input bits of P_B times the number of circuits). The use of OT extension may greatly improve the computational complexity associated with connectors of input bits of P_B, with the number of exponentiations becoming linear in the statistical parameter (and amortizable across multiple executions), but its use requires an initial interactive phase between the two parties. This is justifiable whenever the initial rounds of interaction are not a problem and/or when a multiplicity of executions between the same parties requires a very large number of OTs.

Connectors of output of P_B. In IFC, generating the connectors involves generating two BitComs per output wire of each circuit, i.e., overall a quadratic number of multiplications. Furthermore, in the respond stage it requires computing two multiplicative inverses for each output wire of P_B, for each evaluation circuit (548). The verification (for check instances) and evaluation (for evaluation instances), does not require multiplicative inverses but also requires a quadratic number of multiplications. If following the forge-and-lose path (in case of malicious behavior by P_A), P_B needs to determine the class of one group element (i.e., decrypting a GM BitCom) per input bit of P_A. Since in this case P_B would know the respective trapdoor, the computation essentially amounts to computing one Legendre symbol

modulo each prime, for each of the input bits of P_A.

In DLC, the complexity of the described procedure requires P_A to produce a quadratic number of Pedersen BitComs (520), i.e., a unitary number of exponentiations per output wires of P_B of each circuit. However, the verification of BitComs performed in the evaluation stage can be easily modified into a statistical verification that only requires a linear number of exponentiations. If the forge-and-lose path is followed, then the decryption of all input bits of P_A can be achieved with a single exponentiation followed by a division, i.e., an ElGamal decryption (of the single ElGamal BitStringCom obtained from an homomorphic combination of the ElGamal BitComs of the input of P_A).

Both in IFC and DLC, since P_B must not disclose whether or not the forge-and-lose path was followed, the actual time of even an honest computation should be as high as the one when the forge-and-lose path is followed.

Equiv Coms and Ext Coms. Both in IFC and DLC, each Equiv-Com and each Ext-Com used in the RSC technique and/or in the underlying NZIKPs and NIZKPoKs can be performed with a unitary number of exponentiations. Also, a simulatable coin-flipping (Figure C.1) based on an Ext&Equiv Com scheme can be implemented with Equiv-Coms (commit and open) in unitary number and with Ext-Coms (commit and open) in number linear in the statistical parameter, independently of the length of the bit-string being committed. Thus, the number of exponentiations is not more than linear for the coin-flipping used to produce permutations for the outer BitComs, and the one used for the NIZKPoK of Blum integer trapdoor (Section A.2).

NIZKPoKs of trapdoor. In IFC, the NIZKPoK of Blum integer trapdoor (Figure A.2) requires the prover to compute square-roots and Jacobi Symbols in number linear in statistical security parameter. An eventual additional NIZKP of correct Blum integer (Figure A.1). In DLC, each NIZKPoK of ElGamal openings requires also a number of exponentiations linear in the security parameter.

NIZKP of same committed bits. Both in IFC and DLC, any NIZKP of same committed bits (i.e., at once for all input bits or all output bits of a party) requires producing BitComs in number linear in the statistical parameter, and homomorphically combine a quadratic number of BitComs (linear in the statistical parameter times the number of BitComs). Overall, in

IFC this represents a quadratic number of multiplications and no exponentiation. In DLC it represent a quadratic number of multiplications and a linear number of exponentiations.

Garbled circuits and RSC Equiv-Com. The garbling of each circuit requires a number of block-cipher executions (e.g., AES) linear in the number of garbled gates. This can be done extremely fast with current technology, e.g., hardware-acceleration enabling several million garbled gates per second. However, the implementation of the RSC technique, whose goal is to reduce communication, may reduce the overall throughput, as it also requires hashing all circuits. The real cost of the RSC technique may be better clarified with concrete implementations, pipelining the garbled circuit generation and hashing. Concrete results may help deciding the optimal cut-and-choose parameters that simultaneously improve time and reduce communication complexity, for a given statistical security goal.

3.5 Security analysis

This section analyzes the security of the S2PC-with-Coms protocol, within the ideal/real simulation paradigm. §3.5.1 establishes the simulatability statement. §3.5.2 discusses practical variations of the setting in which security could be proven. §3.5.3 gives a proof sketch, with a high-level intuition for how to construct simulators. §3.5.4 discusses several clarifying remarks about definitional aspects that otherwise remain implicit. The actual proof of security is deferred to Section B.3 in Appendix, specifying simulators and showing that they lead the executions in the ideal world to be indistinguishable from those in the real world.

3.5.1 Security statement

Security is stated in the hybrid model with access to an ideal extractable multi-commitment (Ext-MCom) functionality $\mathcal{F}_{\mathrm{X}}$, an ideal equivocable multi-commitment (Equiv-MCom) functionality $\mathcal{F}_{\mathrm{Q}}$, and a (global) non-programmable random oracle (NPRO), and with the parameters of the outer commitment schemes (used to produce the commitments that are part of the protocol output) being defined by a (global) trusted setup. The NPRO is required only to enable non-interactive versions of the underlying sub-protocols, namely an Ext-and-Equiv-Com with non-interactive phases, and NIZKPs and NIZKPoKs, and could otherwise be avoided. While the ideal Ext-MCom and Equiv-MCom functionalities can be directly instan-

tiated based on a local CRS, the security statement based on these ideal multi-commitment functionalities makes it more obvious that all sub-protocols can be instantiated based on the same short CRS, instead of having to require an independent CRS for each sub-protocol and/or without requiring more interaction to stretch an initial CRS [CR03].

Theorem 1 (UC security of the S2PC-with-Coms protocol, in the ($\mathcal{F}_{\mathbf{X}}$, $\mathcal{F}_{\mathbf{Q}}$, NPRO)-hybrid model).

(i) Let the parameters of the cut-and-choose of garbled circuits be defined by a minimal $e_{\min}$ and a maximal $e_{\max}$ number of evaluation instances and an overall polynomial number s of garbled circuits satisfying the statistical security goal σ (see row 10 in Table 3.1).

(ii) Let the ideal $\mathcal{F}_{\text{MCom}}$ (used to commit outer-randomness permutations for wires of P_A) the ideal $\mathcal{F}_{\text{GMCF-1}}$ (used to coin-flip outer-Com permutations for wires of P_B), the NIZKPs and the NIZKPoKs be instantiated with simulatable protocols, with statistical security negligibly small in the computational security parameter κ, based on ideal functionalities $\mathcal{F}_{\text{X}}$ for an Ext-MCom scheme, $\mathcal{F}_{\text{Q}}$ for an Equiv-MCom scheme, and a global NPRO H.

(iii) Let the parameters of the outer Ext-Com schemes be defined by a global trusted setup.

(iv) Let all needed cryptographic primitives (PRG, CR-Hash, garbling scheme) be secure in respect to the computational security parameter κ.

Then: *the protocol described in Section B.2, and at high level in Section 3.1, securely emulates the ideal functionality $\mathcal{F}_{S2PCwC}$ of S2PC-with-Coms, in the presence of static and computationally active adversaries, with error probability negligibly close, in the computational parameter κ, to a value negligibly small in the statistical parameter σ.*

3.5.2 Practical variations

Variation 1. The theorem preamble asks (in item ii) that the statistical security of the underlying sub-protocols equates the cryptographic security (e.g., 128 bits). This is so that the underlying sub-protocols may be implemented in their non-interactive versions, i.e., so that each phase of the Ext-and-Equiv Com scheme, and each NIZKP and NIZKPoK requires a single message (from sender/prover to receiver/verifier). At the cost of more interaction, the NPRO may be avoided and the statistical security of the underlying sub-protocols can be made as small as the intended statistical security of the main protocol (e.g., 40 bits). Conversely, in order to avoid interaction for the decision of the cut-and-choose partition, and

thus achieve a protocol with only three communication steps, the partition may be decided by the garbled circuit constructor with the help of the NPRO, if the overall statistical security parameter of the protocol is made as high as the computational security parameter (e.g., 128 bits). (See Remark 2.4.2 for yet another alternative, with non-interactive decision of the partition and retaining an independent statistical security parameter.)

Variation 2. In a standalone setting, if rewinding is deemed possible in the simulation, then the ideal Ext-MCom and Equiv-MCom functionalities can be removed, and the whole protocol be implemented in the plain model. As mentioned in §2.2.2, in the CRS setting the global setup that defines the Com-scheme parameters can be replaced by a fixed protocol parameter. Instead, at the cost of more interaction it could be decided by a coin-flipping protocol, with efficient instantiations for DLC but unreasonably costly for IFC. The PKI setting can also be avoided by having each party select her own outer commitment scheme parameters, and then informing it to the other party. In the 2-out-of-1 setting this would necessarily require an extra communication message where $\mathrm{P_A}$ informs $\mathrm{P_B}$ of the scheme to be used for oblivious transfer.

Variation 3. By removing the outer-Com schemes from the output, and consequently being able to remove the coin-flipping components of the protocol, and avoiding the final permutation of outer-Coms, the adjusted protocol securely emulates the regular S2PC functionality, in the presence of static and computationally active adversaries. In this case all intermediate BitCom schemes may be defined per protocol execution.

Remark 3.8 (On the hiding of bits of $\mathbf{P_B}$). It the original paper, the protocol for S2PC-with-Coms, simulatable with rewinding, used Blum BitComs for the outer bits, thus allowing the underlying S2PC to be unconditionally hiding of the circuit input bits and circuit output bits of $\mathrm{P_B}$. This is not the case in the description in this dissertation, in a setting of simulation without rewinding, because of the choice of using computationally hiding extractable commitments for the outer bits.

3.5.3 Proof sketch

In the static corruption model, in the two-party setting, the simulator $\mathcal{S}$ can (as a simplification) be considered as the corrupted party in the ideal world (see §2.2.1), being activated to start a protocol execution of an ideal S2PC-with-Coms functionality $\mathcal{F}_{\text{S2PCwC}}$, as defined in §2.2.5 (Figure B.4 and Figure 2.3). In a simulated execution, $\mathcal{S}$ takes advantage of the ZKPoKs to extract the circuit inputs bits and the randomness of the initial outer commitments of the malicious party in the simulated execution. If the malicious party in the simulated execution aborts before the step where the other party would have learned the output, then $\mathcal{S}$ emulates an abort in the ideal world. Otherwise $\mathcal{S}$ sends the extracted circuit input bits to the ideal $\mathcal{F}_{\text{S2PCwC}}$ in the ideal world. As a result, $\mathcal{F}_{\text{S2PCwC}}$ locally evaluates the Boolean circuit and produces commitments of all input bits and output bits, and then replies back with the circuit output bits of $\mathcal{S}$ (i.e., the malicious party in the ideal world) and the respective commitments and randomnesses, as well as the commitments of the input and output of the other ideal party. Then, again in the simulated execution, the simulator takes advantage of the simulatable coin-flipping of permutations for wires of $\mathrm{P_B}$, the simulatable Ext-and-Equiv Com of permutations for wire sets of $\mathrm{P_A}$, and the extractable initial outer-Coms of wire sets of $\mathrm{P_A}$, to induce the needed permutations that will permute the initial outer Coms into the final outer Coms decided by the ideal functionality.

To induce the circuit output bits, $\mathcal{S}$ in the role of $\mathrm{P_A}$ in the simulated execution learns the cut-and-choose partition and then equivocates the *evaluation* garbled circuits to output exactly the circuit output that was decided by the ideal functionality. Conversely, $\mathcal{S}$ in the role of $\mathrm{P_B}$ in the simulated execution sends to $\mathrm{P_A}$ the needed final offset bits of the private output of $\mathrm{P_A}$, and give a fake ZKP that they are the obtained circuit output bits.

Then, the simulator outputs in the ideal world whatever the malicious party (i.e., the real adversary) outputs in the simulated execution. Actually, in the UC framework where the environment ($\mathcal{Z}$) is allowed to interact with the adversary at arbitrary moments in time, the simulator always leaves open a channel to relay the messages between $\mathcal{Z}$ and the black-box adversary in the simulated execution.

Related to the order of received outputs, there is a main difference across the simulators for different parties. If $\widehat{\mathrm{P}}_{\mathrm{B}}^{*}$ is the corrupted party in the ideal world (i.e., controlled by $\mathcal{S}$), then the ideal functionality sends the output to $\mathcal{S}$ in the ideal world before it sends it to the ideal $\widehat{\mathrm{P}}_{\mathrm{A}}$. Thus, $\mathcal{S}$ in the simulated execution still needs to determine whether the malicious

P_B^* in the simulated execution (controlled by te real adversary $\mathcal{A}$) allows the other party (the honest P_A impersonated by $\mathcal{S}$) to compute a proper final output. If it allows, then $\mathcal{S}$ in the role of ideal $\widehat{P}_B$ sends an `OK` message to the ideal functionality (383). Otherwise it sends an `abort` message (384), which will make the ideal $\widehat{P}_A$ also finish with an `abort`. Conversely, in the case of a malicious P_A^*, whenever the simulator $\mathcal{S}$ receives the output from the ideal functionality it follows that the ideal $\widehat{P}_B$ has already outputted in the ideal world. $\mathcal{S}$ is then left to induce the malicious P_A in the simulated execution to obtain the needed output.

In the case of a malicious P_A^*, the challenging part of the proof is proving soundness, i.e., that a malicious P_A cannot prevent an honest P_B from learning a correct output if the simulator in the role of honest P_B can learn it when using an arbitrary input in the simulaton. Specifically, since the simulator simulates an honest P_B with arbitrary input (because it does not have access to the actual input used by P_B in the ideal world), it follows that the soundness error probability must not depend (i.e., up to a negligible value in the security parameters) on the combined inputs of P_A and P_B, which for a generic Boolean circuit also means not depending on the final circuit outputs.

3.5.4 Several remarks

Remark 3.9 (On the properties of outer-Com schemes). The defined ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ allows non-UC Com schemes as outer-Com schemes, as long as they are hiding and binding. Indeed the functionality was defined with an explicit construction of real commitment schemes (see Remark 2.6), as a way to ensure that the output Coms are random without control by any of the parties. Nonetheless, since a NIZKPoK of the openings is explicitly required by the protocol, the augmented use in the protocol makes the commitments extractable (both the committed value and the used "randomness"). Actually, if the outer-Com scheme parameters were provided by a local setup (impersonatable by the simulator), some BitComs could be directly extractable even without the ZKPoKs, e.g., GM BitComs (for both the committed bits and the used "randomness") and ElGamal BitComs (for the commited bits, but not for the "randomness"). Similarly, non-malleability with respect to commitment is implied by non-malleability of the underlying NIZKPoK. On the other hand, equivocability of the commitments is not an explicit requirement, as it is not meaningful, because the commitments are not opened within the context of an execution.

Remark 3.10 (On obtaining the public parameters of the outer Com schemes). For simplicity, in the proof it is assumed that the queries to and the replies from the ideal global setup functionality $\mathcal{F}_{\text{SetupComs}}$ are not visible by $\mathcal{Z}$. This means that the simulator can, without implying distinguishability between the ideal and real worlds, obtain the public parameters of both parties, independently of the malicious party in the simulated world also making such queries. This assumption could be avoided with a slight change in the SETUP stage. It could instead be defined that the initial input of both parties would include authenticated public parameters of both parties (i.e., authenticated by the global setup based on a public key that is a fixed parameter of the protocol), and that in the ideal world the parties would send the parameters to the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$, who would check they are consistent across the two parties. The parties could then still query the global setup to obtain their private parameters (i.e., in case of a PKI setup), needed for the useful ZKPoK of trapdoor. These queries could be visible by $\mathcal{Z}$, as long as the secret parameters included in the replies were ideally encrypted (i.e., part of the private component of messages), to prevent them from being learned by $\mathcal{Z}$.

Remark 3.11 (A single CRS per protocol execution). The use of the ideal/real simulation paradigm enables simulatability when replacing each internal ideal functionality by a respective simulatable concrete sub-protocol instantiation. While each such sub-protocol is proven simulatable based on a local setup, it is useful to consider how to instantiate all of them based on a single call to the same setup (e.g., a CRS), with respective state shared by all sub-routines (see [CR03]). This can be achieved in this protocol by reducing all ideal sub-functionalities to the same kind of sub-functionality and then consider a functionality that essentially implements multiple independent copies of said sub-functionality. Particularly, the simulatability of each NIZKP, NIZKPoKs and Ext-and-Equiv Com can be reduced to using separate Ext-MComs and Equiv-MComs, which allow multiple calls.

Then, by using a pair of Dual Com schemes (e.g., Pedersen and ElGamal; or Blum and GM), i.e., with same extraction-trapdoor and equivocation-trapdoor, but with the respective public parameters being defined by a local CRS (i.e., the trapdoor being unknown to any real party), it follows that the same short CRS is enough to implement all underlying Ext MComs and Equiv MComs, and thus enough to enable simulatability of the overall S2PC-with-Coms execution. This means that the NIZKP and NIZKPoK transcripts inside each S2PC-with-Coms execution will be validated based of the same setup parameters (e.g., a local CRS).

Remark 3.12 (Uniqueness of session identifier). The claims of simulatability are supported on the assumption that the session identifier is unique. In other words, the indistinguishability game only allows $\mathcal{Z}$ to request one protocol execution per each contextual tuple, even though it may run external concurrent executions with different identifiers. Indeed, the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ is defined to ignore subsequent calls based on the same contextual information, namely the same pair (*sid*, *ssid*, P_A, P_B). If $\mathcal{Z}$ were able to make multiple calls with the same context, then a lack of the non-repetition property in the real world would allow $\mathcal{Z}$ to distinguish between the two worlds — the real world would be the only one where two executions with the same context would successfully return a non-abort output. Without the non-repetition of *ssid* rule, the real world would require an additional mechanism to ensure non-repetition of identifiers (e.g., based on timestamps, counters and/or statefull cache, and/or involving an interactive coin-flipping phase for determining an unique identifier suffix composed of randomness from both parties). The problem with an additional interactive phase is that it would make the protocol inherently more interactive. Another approach, perhaps more promising, is to redefine the ideal S2PC functionality to incorporate a mechanism where the two parties augment the session identifier across the sequence of messages, and in the end return the augmented identifier. Correspondingly, the ideal functionality would manage the dynamic update of the identifiers across the messages. In this way, several executions activated by the $\mathcal{Z}$ using the same (session and sub-session) identifiers would nonetheless lead to an execution where the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ would see different (augmented) identifiers and thus would no longer imply distinguishability with the real world. These possible variants are not further discussed in the dissertation.

3.5.5 Linked executions

The outer-Coms outputted by the S2PC-with-Coms protocol may be useful to enable general linkage of S2PC executions. On a follow-up S2PC-with-Coms, the outer-Coms obtained by each party in a previous execution can be reused in the next execution, without need for additional ZKPs or ZKPoKs. For example, for protocols defined as a sequence of small S2PC sub-protocols in the semi-honest model (e.g., [LP02]), security can be enhanced to resist also the malicious model, by naturally using the input and output of previous executions (or transformations thereof) as the input of the subsequent executions.

Another simple example is a dual-path execution, where the parties play two executions

with exchanged roles, with each party using the same input in both executions, but each party evaluating circuits with only her own output. While the protocol in this dissertation described a two-output solution based on a single-path approach (i.e., only P_B evaluated the garbled circuits), there may be applications where a dual-path execution allows better tradeoffs, e.g., more balanced communication in each direction.

It is also possible to prove, outside of the garbled circuits, arbitrary transformations and relations that involve only the bits of one party. A party may commit new bits for a subsequent S2PC and prove that they satisfy certain non-probabilistic polynomial verifiable relations with the private input and output bits of previous S2PCs. These proofs can take advantage of the homomorphic properties of outer-Coms. For example, for XOR homomorphic outer-Coms, linear Boolean gates (e.g., XOR and NOT) can be implemented by a simple group operation, not requiring any additional ZKP. Verification of multiplicative operations can be reduced to ZKPs that out of a number of BitComs there are a certain minimal number of BitComs committing to a 1 or a 0. For example, proving that a certain BitCom commits to the AND of the bits committed by two other BitComs can be reduced to a simple ZKP that there are at least two zeros committed in a triplet of BitComs, with the triplet being built from a XOR-homomorphic combination of the original three BitComs. (The first bit is the AND of the two last if and only if there are at least two zeros in the triplet composed of the first bit and of the XOR of the first bit with each of the other two bits [Bra06]. A different method can be found in [BDP00].) Besides Boolean relations, the use of additively-homomorphic bit-string Coms may also allow more efficient transformations based on arithmetic operations, e.g., modular sums and multiplications.

If in a sequence of executions the output bits of some intermediate execution bits are not relevant for the final output, then the coin-flipping for randomization of outer-Coms does not have to apply to those bits. A PKI setting allows further simplifications in regard to the input bits of P_B — if in a 2-out-of-1 OT instantiation the OT BitCom scheme $\mathscr{B}_{OT}$ is the same as the forge-and-lose BitCom scheme $\mathscr{B}_{FLB}$ used for output bits, then those intermediate output BitComs can be reused for the next 2-out-of-1 OT, without exponentiations.

3.6 Developments to the forge-and-lose technique

3.6.1 Improvements in this dissertation

The protocol described in this dissertation is based on the protocol presented in the original paper that introduced the *forge-and-lose* technique for S2PC [Bra13], but contains substantial improvements and technical modifications, as follows:

- **RSC technique and no rewinding in proof.** The protocol is now directly described with a random seed checking (RSC) technique, instead of referring to it as an ad-hoc optimization. Instead of regular commitments (one commitment per challenge index) and simulation with rewinding, the protocol now explicitly uses a (single) RSC Equiv-Com, thus enabling equivocation of garbled circuits without having to use rewinding. Hence, the cut-and-choose challenge no longer needs to be defined via a two-party coin-flipping, but can rather be simply decided by P_B. It was also mentioned that the reduction in number of communicated garbled circuits allowed by the RSC technique (and complementary by the forge-and-lose technique) is possible even with a non-interactive cut-and-choose, by using an erasure code at the level of garbled-circuits (idea from [AMPR14]) and without P_A learning the cut-and-choose partition.
- **Coin-flipping sub-protocol.** Due to simulatability reasons, the two-party coin flipping used to decide random permutations of the outer BitComs is now based on a generalized coin-flipping (into a well) where one party learns a pre-image (a randomness to a commitment of 0) and the other learns the image under a one-way function (the commitment to 0). In the original paper it was sufficient to use a simpler coin-flipping, with symmetric output, because there the outer Coms of each party were based on parameters with trapdoor known by the other party and because with IFC the trapdoor was sufficient to extract the randomness. In the IFC instantiation, the coin flipping is now based on a new and more efficient Ext-and-Equiv Com (described in the next chapter [Bra16]), instead of using a sub-protocol instantiation requiring a linear number of exponentiations and communication expansion rate (asymptotically) greater than one for committing the contribution of P_A. In the new DLC instantiation, where the outer-Coms are compacted using bit-string commitments, a specialized protocol takes advantage of additively homomorphic properties. The new sub-protocols are now also suited to simulatability without rewinding.
- **BitCom schemes.** The new notation for BitComs ($\mathscr{B}$) allows better separability between the BitCom schemes and trapdoors across different wire types, and distinguishing between

outer BitComs (part of the output) and *intermediate* BitComs (the outside part of connectors). The previous requirement of unconditionally hiding BitComs and unconditionally binding BitComs with trapdoor was now generalized to extractable (Ext) BitComs and equivocable (Equiv) BitComs. The description of all BitComs and connectors was adapted to allow integer-factorization cryptography (IFC) and discrete-log based cryptographic (DLC) primitives, instead of just the former one.

- **BitCom variations for input of $\mathbf{P_A}$.** The connectors for input bits of P_A were previously described based on an unconditionally hiding XOR-homomorphic BitCom scheme; now the description accepts any (e.g., Ext or Equiv) XOR-homomorphic, or even XOR pseudo-homomorphic (see §2.3.2), BitCom scheme. P_A can now directly use the same Ext-BitCom scheme (with trapdoor) that enables the forge-and-lose technique, whereas previously an unconditionally hiding scheme was used for connectors and an unconditionally binding scheme was used for the forge-and-lose technique, augmented by a ZKP of equivalence of committed bits. In the suggested DLC instantiation, the communication associated with connectors can be further reduced, by using bit-string commitments that commit to permutation strings at once, instead of one BitCom per permutation bit.
- **BitCom variations for input of $\mathbf{P_B}$.** For connectors for input bits of P_B, the original paper only considered the IFC setting, where a 2-out-of-1 OT method was used at the level of BitComs. The new description of connectors now also considers the possibility of 1-out-of-2 OT (as in traditional S2PC protocols), namely with a DLC instantiation.
- **ZKPs and ZKPoKs without rewinding and without interaction.** The analysis of ZKPs and ZKPoKs is deferred to Appendix A. All ZK sub-protocols were adjusted to not require rewinding in the respective simulation, and to be instantiable in a non-interactive setting (i.e., NIZKPs and NIZKPoKs). For example, this includes NIZKPoK of Blum integer trapdoor and of DL, NIZKP of good Blum integer, NIZKP of same committed bits and NIZKP of good ElGamal Com.
- **Communication complexity.** The tables benchmarking the communication and computational complexity have been updated, now considering the RSC technique with single global hash, different types of instantiation (including IFC and DLC), and more efficient sub-protocols, e.g., coin-flipping and different (and non-interactive) ZKPs and ZKPoKs, as well as recent improvements (from related work) on garbled circuit constructions.
- **Simulatability.** The analysis of simulatability has been revised to the new protocol structure, now removing all explicit rewinding. Two possibilities are now considered

in terms of trusted setup: a global-PKI (GPKI) or a global-CRS, directly defining the parameters of the outer BitCom schemes. Additionally, a local CRS is considered to aid with simulatability, namely to help instantiate the needed NIZKPoKs, coin-flipping and/or the RSC Equiv-Com. The final outer BitComs of each party are now associated with an Ext BitCom scheme for which the respective party knows the trapdoor, whereas before (when using unconditionally hiding BitComs with trapdoor) this was inverted. In a simulation setting that allows rewinding the local CRS can be avoided. (As before, the S2PC protocol can be simplified and the global setup can be avoided in a simple S2PC setting, i.e., if outer commitments are not required.)

3.6.2 Subsequent work

Subsequently to the original forge-and-lose paper [Bra13], other works have also used the paradigm of non-interactively recovering the input of P_A, in case of malicious behavior by P_B, to also allow reducing the communication complexity of S2PC based on garbled circuits.

Forge-and-lose based on DLC. A forge-and-lose type of technique was used by Afshar et al. [AMPR14] for non-interactive (i.e., with two messages) S2PC based on cut-and-choose. They devised a DLC-based technique (to which they call "cheating recovery"), whereas the original forge-and-lose paper had only considered an IFC construction (requiring only a linear number of exponentiations). The revised description in this dissertation, which generalizes the constructions to work with more generic Ext-BitComs and Equiv-BitComs, considers a new DLC instantiation, with several improvements in complexity. For example: for input wires of P_A, they require one ElGamal commitment per wire per circuit, whereas in this dissertation the wire keys are instead random bit-strings (decided by the underlying garbling mechanism), which are connected to the outer commitments via connectors that only require one BitCom per bit (without repetition per circuit) and one BitStringCom per circuit (without repetition across wires); for output wires, they use two distinct group elements (exponentiations) per wire per circuit, whereas in this dissertation the connection only communicates one Pedersen commitment per output bit (without repetition across circuits) and then two pre-images (exponents) per connector (i.e., per output wire per circuit). This dissertation additionally integrates a description of the 2-output case (where P_A also learns a circuit output), and the augmented case of S2PC-with-Coms, where both parties receive Coms of all circuit input and output wires, and each party receives the randomness needed to open her respective inputs and outputs.

Number of garbled circuits in a non-interactive setting. While the original IFC-based forge-and-lose technique [Bra13] is inherently non-interactive with respect to the recovery of the input of P_A in the evaluation stage, its integration into the cut-and-choose approach did not focus on a non-interactive decision of the cut-and-choose challenge. The obvious approach of using a random oracle (or, with more sophistication, using a NPRO) to let P_A decide the cut-and-choose partition would require increasing the the statistical security parameter (e.g., 40 bits) up to equating a short-term or long-term computational security parameter (e.g., to 96 or 128 bit, respectively for short term or long term durations of execution), and/or consider other tradeoffs (e.g., a purposely slow NPRO). In contrast, Afshar et al. [AMPR14] suggested a different method to decide and enforce the cut-and-choose partition, based on a more involved combination of oblivious transfer and erasure code, preventing an increase of the number of garbled circuits in a non-interactive setting. In their technique, P_A does not get to learn the cut-and-choose partition, because the decided type of each instance (*check* vs. *evaluation*) is hidden within an oblivious transfer. It is nonetheless worth emphasizing that, by allowing interaction (two more messages) to decide the cut-and-choose partition challenge, the original forge-and-lose method, as well as the "secure evaluation of cheating" (with additional interaction in the evaluation stage) method of Lindell [Lin13], already achieved 40 bits of statistical security with 40 circuits (if letting the number of evaluation instances be completely variable), or 41 circuits if limiting the number of evaluation instances to at most 20, without requiring additional OTs for the cut-and-choose nor an erasure encoding and decoding of garbled circuits. For real-time online protocols, the additional messages might already even be required to establish a reliable communication protocol.

Forge-and-lose in mini-crypt. As mentioned in Remark 3.3, Frederiksen et al. used the forge-and-lose approach in a manner that intentionally avoided the use of trapdoor commitments [FJN14a, Fre15], and for that matter all number theoretic assumptions, and using only "lightweight primitives" (though still requiring OT and coin-flipping). Their technique is based on an erasure code (with decoding based on polynomial interpolation) performed at the level of circuit output wires, and also requires hashing the original circuit output inside the garbled circuit and output them as output wires, as a way to expand the number of inconsistent output wires. The use of Equiv-Coms in the protocol in this dissertation does not require augmenting the underlying Boolean circuit, and is abstract to the garbling scheme, and is within an overall S2PC-with-Coms protocol (i.e., beyond

simple S2PC of Boolean circuits). It is not clear how the communication and computational complexity of the two protocols may compare for different parameter sizes (input and output lengths) and concrete computational implementations.

Interactive recovery from cheating. In the line of dual execution approaches, Kolesnikov et. al [KMRR15] devised a protocol where the number of garbled circuits evaluated by each party is equal to the number of bits of statistical security (e.g., as in [HKE13]), and additionally reducing the leakage to a single bit even in the statistically negligible case of adversarial success. As in [Lin13], the protocol requires an interactive recovery phase after the main bulk of evaluation of garbled circuits. In contrast, the forge-and-lose technique allows a non-interactive recovery phase and requires evaluation of circuits by only one party, though in case of adversarial success (negligible in the statistical security parameter) the honest party is induced to learn a completely incorrect output, and thus possibly leak more than one bit.

Cut-and-choose at the level of gates. Improving over prior Lego techniques [NO09, FJN+13], a more recent technique called Tiny Lego, by Frederiksen et. al [FJNT15], also performing a cut-and-choose at the level of garbled gates within a single circuit, reduced further the overall number of garbled gates in comparison with a cut-and-choose of garbled circuits, by introducing an authenticator mechanism in each wire, which changes the criterion of correctness of each bucket of gates to have a single correct gate, instead of a majority.

Implementations Benchmarks. Very recent works implement a combination of state-of-the-art techniques to reduce the amount of time it takes to perform S2PC evaluations, with myriad tradeoffs, e.g., between offline and online phases, and comparing multi-execution settings vs. single executions. This is the case of implementations by Lindell and Riva [LR15], Rindal and Rosulek [RR16], Wang, Malozemoff and Katz [WMK16], and Nielsen, Schneider and Trifiletti [NST16]. While those works have not been analyzed in this dissertation, it is worth mentioning that they report impressive time performances, e.g., less than 100 milliseconds per execution, and (when considering tradeoffs between offline and online stages), with an online throughput below 1 millisecond per execution (not counting network latency). The dissertation focused more on a detailed estimation of communication complexity.

Chapter 4

Simulatable commitments and coin-flipping

This chapter describes research results with applicability to two-party coin-flipping and useful within the S2PC-with-Coms protocol. The original paper [Bra16] had considered two different simulatability settings: with rewinding and without rewinding. The later setting, necessary for universal composability, is the only one considered here. The essential construction is an efficient extractable-and-equivocable commitment scheme, in the static and computational malicious model. One meaningful improvement in this dissertation is the distinction between different types of extractability, namely with respect to aborting actions by a malicious sender.

The protocol can be parametrized to achieve asymptotic communication-rate arbitrarily close to 1 in each phase, i.e., the number of bits exchanged when committing or opening a large (i.e., asymptotically increasing) bit-string is larger than the bit-string at most by a multiplication factor that can be defined arbitrarily close to 1. The length of collision-resistant (CR) hashing input and of pseudo-random generation (PRG) output can also be reduced asymptotically to a rate arbitrarily close to 1. The approach, denoted *expand-mask-hash*, uses the PRG and the CR-Hash function to combine separate extractable commitments and equivocable commitments (associated with short bit-strings) into a unified extractable-and-equivocable property amplified to a larger target length, amortizing the cost of base commitments.

The scheme has communication complexity comparable to recent state-of-the-art constructions, while following a distinct design approach. Notably it does not require explicit use of oblivious transfer (thus allowing tradeoffs with less interactivity) and instead uses a *cut-and-choose* method, which can be instantiated non-interactively with a non-programmable random oracle. The scheme also uses an erasure encoding (instead of stronger error correction codes)

combined with the cut-and-choose structure, thus allowing the size of each instance to be reduced proportionally to the number of instances. The protocol and proof is defined in a hybrid model with access to ideal schemes for an Ext-Com (not necessarily Equiv) and an Equiv-Com (not necessarily Ext). In other words, for each type of base commitment scheme (Ext or Equiv) the simulator is not required to use the complementary property (Equiv or Ext, respectively).

Organization. An introduction has been given in Section 1.4. Section 4.1 reviews related work. Section 4.2 describes the initial intuition for a new construction based on a cut-and-choose approach, sufficient to devise a communication-inefficient scheme. It also highlights the distinction between different types of extractability, motivating possible variations in the protocol. Section 4.3 explains optimizations based on an authenticator and an erasure code, which together enable an Ext-and-Equiv scheme with asymptotic communication rate arbitrarily close to 1. Section 4.4 briefly discusses some coin-flipping applications useful to the S2PC-with-Coms protocol. Several details and protocol specifications are deferred to Sections C.1 and C.2 in Appendix.

4.1 Related work

4.1.1 Basic primitives

One-way permutations or functions are enough in theory to achieve many useful cryptographic primitives, such as PRGs [HILL99, VZ12], one-way hash functions [NY89, Rom90], some types of commitment schemes [Nao91, DCO99] and ZK proofs of knowledge (ZKPoK) [FS90b]. CR-Hash functions can also be built from other primitives [Sim98], such as claw-free sets of permutations [Dam88b] or pseudo-permutations [Rus95]. Based on such primitives, coin-flipping can be achieved in different ways, e.g., based solely on one-way functions [Lin03, PW09] (with rewinding). In several simulatability settings, coin-flipping can be more directly based on higher level primitives, such as bit or multi-bit Ext&Equiv-Com schemes (e.g., [CF01, DN02, DC03]) and even from coin-flipping protocols with weaker properties [HMQU06, LN11]. In the computational model (considered herein), there are known theoretical feasibility results about coin-flipping, covering the stand-alone and the UC security settings. For example, in the UC setting it is possible to achieve coin-flipping extension, i.e., coin-flip a large bit-string when having as basis a single invocation of an ideal functionality realizing coin-flipping of a

shorter length [HMQU06]. This dissertation shares the concern of achieving properties in large strings based on functionalities associated with short strings, but focuses on a base of a few short commitments (not needing to be simultaneously Ext and Equiv) and has a motivation of improving efficiency. Implications between different primitives (e.g., see [DNO10] for relations between OT and commitments) are not analyzed herein.

Only in very recent research works (including this one) have UC commitment schemes been devised with an amortized communication cost, with asymptotic communication rate close to 1. The new UC-Com scheme is directly applicable to coin-flipping, allowing an asymptotic amortized communication of three bits per flipped coin (see Remark 4.5 for approaches with two bits per flipped coin).

In spite of very efficient realizations of OT-extension [ALSZ15] and free-XOR techniques [KS08b] for garbled circuits, a coin-flipping based on a direct (generic) approach of S2PC of bit-wise-XOR would still induce, in communication and computation, a multiplicative cost proportional to the security parameter, by requiring one *minicrypt* block operation (e.g., block-cipher evaluation) per flipped coin. In contrast, in the approach herein each block of bits (e.g., equal to the security parameter) requires a unitary number of minicrypt block operations (e.g., close to 1 block-cipher for the PRG and 1 CR-Hash).

The idea of combining commitments with a CR-Hash (*hash then commit*) and commitments with a PRG for efficiency reasons is not new. The former resembles the *hash then sign* paradigm, and it also has applications to non-malleable commitments [DCKOS01]. The later resembles hybrid encryption, where a symmetric key (the analogous to the PRG seed) would be encrypted with a public key system (the analogous to the commitment) and then the message would be encrypted with a symmetric scheme (the analogous to the one-time-pad masking using the PRG expansion). The work herein explores ways of combining both techniques, aimed at achieving simulatability in coin-flipping and UC commitment schemes.

4.1.2 UC commitment schemes

When simulations-with-rewinding are not possible, the simulatability of flipping even a single coin using the traditional template requires simultaneous Ext and Equiv properties of the underlying commitment scheme [CR03]. Canetti and Fischlin [CF01] developed non-interactive UC commitments, requiring a unitary number of asymmetric operations per committed bit. The construction assumes a CRS setup and is based on the equivocable bit-

commitment from Di Crescenzo, Ishai and Ostrovsky [DCIO98]. Canetti, Lindell, Ostrovsky, and Sahai [CLOS02] proposed other non-interactive schemes from general primitives, with adaptive security without erasures. Damgård and Nielsen [DN02] then improved with a construction denoted *mixed commitment scheme*, that is able to commit a linear number of bits using only a unitary number of asymmetric operations, and using a linear number of communicated bits. For some keys they are unconditionally-hiding and *equivocable*, whereas for other keys they are unconditionally-binding and *extractable*. Di Crescenzo [DC03] devised two non-interactive Ext&Equiv-Com schemes for individual bits, in the public random string model (suitable to UC). One construction is based on Equiv-Com schemes and NIZKs, the other is based on one Ext-Com and one Equiv-Com schemes. Damgård and Lunemann [DL09] consider UC in a quantum setting and solve the problem of flipping a single bit, based on UC-Coms from [CF01]. Lunemann and Nielsen [LN11] consider also the quantum setting and achieve secure flipping of a bit-string based on mixed commitments from [DN02]. They consider how to amplify security from weaker security notions of coin-flipping (uncontrollable, random) up to full simulatable (enforceable). The use of Ext-Com and Equiv-Com schemes, together with a cut-and-choose and encoding scheme has been previously considered by Damgård and Orlandi [DO10] to enable efficient constructions. They combine these techniques to enhance security from the passive to the active model for secure computation of arithmetic circuits, in a model where a trusted dealer is able to generate correlated triplets. While they achieve efficient constructions for multiparty computation (also including more than two parties), the efficiency is not amortized to communication rate 1.

Efficient non-interactive UC Coms have been previously devised in the *random oracle* model [HMQ04], requiring the ability of the simulator to program the output of th random oracle in order to enable equivocation. Any instantiation of the programmable random oracle by a concrete cryptographic hash function would break equivocability therein, e.g., breaking deniability in some applications. In a different work, constant rate commitments have been achieved in the more ambitious setting of generalized UC (with a globally available setup) [DSW08], using an *augmented* CRS setup and a *non-programmable* random oracle (NPRO).

In the UC setting, more efficient commitment schemes have been proposed for short strings, based on specific assumptions such as DDH intractability, e.g., [Lin11, FLM11, BCPV13, Fuj14], achieving a *low* (but greater than one) constant number of group elements of communication and of exponentiations to commit to a group element. Still, the trivial extension of these protocols for larger strings would imply a linear increase in said number of

asymmetric operations (modular exponentiations), without amortization. Some of the above schemes achieve adaptive security, whereas this dissertation considers only static security.

Recent works with comparable communication efficiency. More aligned with the efficiency goal in this dissertation, recent independent works also achieve asymptotic communication rate 1 or arbitrarily close to 1. [GIKW14] additionally considers selective openings; [DDGN14, CDD+15] additionally consider homomorphic properties and verification of linear relations between committed values; [CDD+15] achieves, comparably to the method herein, linear computational complexity. These protocols are based on a hybrid model with access to an ideal oblivious transfer (OT) functionality. In contrast to OT, the cut-and-choose mechanism described in this dissertation does not hide from the sender the partition of (check and eval) instances in the commit phase.

Also, [GIKW14, DDGN14] rely on *secret sharing schemes* with error-correction or verifiability requirements ([CDD+15] works with any linear code), whereas the construction in this dissertation uses a simpler erasure code (enough to allow the simulator to recover the committed message from correct extracted fragments), with corresponding communication benefits. In contrast to the requirement of full fledged error-correction codes in other constructions, erasure codes suffice in the construction described herein, due to an *authenticator* mechanism (which can be based on a CR-Hash) that enables the simulator to anticipate (before the actual open phase) whether an extracted fragment is correct or not, thus simply ignoring erroneous fragments when reconstructing a message. While such property would be somewhat trivial in an isolated context of Ext-Com schemes, the difficulty overcome here is in simultaneously allowing the fragments to be equivocable, even though the base commitments are not simultaneously Ext&Equiv. This is achieved by combining equivocable pseudo-random masks that are applied in a one-time-pad fashion to mask the fragments.

Also, the *authenticator* mechanism allows a cut-and-choose with fewers instances than what an error correction code would imply (e.g., see Table C.1). A more recent result [FJNT16] improves the complexity of the OT-based protocols (also for additively homomorphic commitments), using an additional *consistency check* mechanism to also allow a simpler erasure code. Still, any OT instantiation in the CRS model would require interaction (at least in a setup phase), whereas the protocol herein can be directly instantiated in a CRS model without interaction if using a non-programmable random oracle, and using a basis of non-interactive Ext Coms and non-interactive Equiv-Coms.

The work herein requires computation time linear in the size of the string being committed, if assuming an underlying linear time computable PRG and CR-Hash and erasure code, and with the last one being considered after fixing a communication rate of the commit phase that determines the minimal cut-and-choose and erasure code parameters. In practice the computational cost of the erasure code may be of practical significance and this is left to evaluate. A more in depth exploration of linear time primitives was explored in [CDD+16], including in regard to erasure codes, there achieving asymptotic rate-1 and linear time regardless of the erasure code parameters.

A concrete comparison between different methods — qualitative (e.g., implications between primitives) and quantitative (actual instantiations and implementations) — is left for future work. For example, for OT base methods there is a cost associated with the initial OT setup phase (e.g., 640 exponentiations in [GIKW14]). In this work a concrete instantiation of Ext or Equiv commitments is not explored in depth, but the complexity is naturally upper bounded by that of full-fledged UC-Coms for short strings, e.g., requiring fewer than a dozen group elements per base commitment [BCPV13]. The overall number of commitments of short strings will depend on the erasure code parameters, defined to meet the goals of statistical security and communication rate.

In summary, this work is focused on the design of protocols that explore the duality between Ext and Equiv commitments, without considering OT as a primitive. About OT only two notes are mentioned here from other work: it is known that UC-OT implies UC-Coms in myriad setup models [DNO10, Fig. 1], e.g., in the *uniform*, the *chosen* and the *any common reference string* models (U/C/A-CRS), and in the *chosen* and *any key registration authority* models (C/A-KRA), whereas the reverse implication is proven only in a narrower set of models (e.g., U/A-CRS, A-KRA) [DNO10, Table 1]; while [GIKW14] shows that "the existence of a semi-honest OT protocol is necessary (and sufficient) for UC-Com length extension," the UC scheme in this dissertation does not make explicit use of OT and can also be seen as a UC-Com length extension (if replacing the Ext-Com and Equiv-Com schemes with an Ext&Equiv-Com scheme) — these two results do not superpose, since [GIKW14] only allows a single call to the ideal Com-scheme, whereas the extension herein requires several calls.

4.2 Intuition for combined Ext and Equiv

This section describes and analyzes at high level a simple but inefficient Ext-and-Equiv Com scheme for large strings (§4.2.1), based on a few commitments of short strings. The analysis then motivates the discussion of different types of extractability (§4.2.2), and considers variations in protocol structure (§4.2.3), serving as a warmup for the description of a more efficient scheme in the subsequent section.

4.2.1 Cut-and-choose warmup

The commitment scheme, between a sender P_S and a receiver P_R, is based on a cut-and-choose approach. After P_S sends several (n) *instances* (in number equal to the statistical parameter σ) of pairs of short commitments to P_R, P_R checks the correctness of some (the *check*) instances to gain *some* confidence that some of the others (the *evaluation* instances) are correct.

For simplicity, the procedural description considers simultaneously two cases (A and B), depending on whether the Equiv-Coms applied to short hashes are also extractable or not. In summary, the use of Ext&Equiv-Coms will improve statistical security and also the type of extractability of the overall scheme, as will be detailed when making the simulation analysis.

Procedure. A sketch of the scheme is depicted in Illustration 4.1.

- **Initialization.** The parties are activated to play a commitment scheme, with an agreed goal of number of bits ($\sigma = n$) of statistical security, with a domain of committable values, and with P_S knowing as private input a *target string* m to commit.
- **Commit phase.** For each instance index (j), P_S samples a random seed s_j and uses it to calculate a pseudo-random string (mask s'_j) with the target length ($|m|$). Then, P_S sends to P_R an Ext-Com $\overline{s}_j$ of the seed and either a (non-extractable) Equiv-Com (case 1) or an Ext&Equiv-Com $\overline{h}_j$ (case 2) of the CR-Hash h_j of the mask s'_j. Upon receiving all instances P_R *cuts* the set of instances (each instance being a pair of commitments), into two random complementary subsets, and *chooses* one (J_V) for a *check* operation and the other (J_E) for an *eval* operation. Then, P_S opens the check seeds and check hashes from the respective commitments, and uses each eval mask to produce a XOR-masking t_j (i.e., via a one-time pad) of the *target string* m, and sends the respective *maskings* to P_R. P_R checks that each opened check hash h_j is indeed the CR-Hash η_j of the PRG expansion s'_j of each opened check seed s_j. Otherwise P_R rejects the commit phase, outputting `abort`.

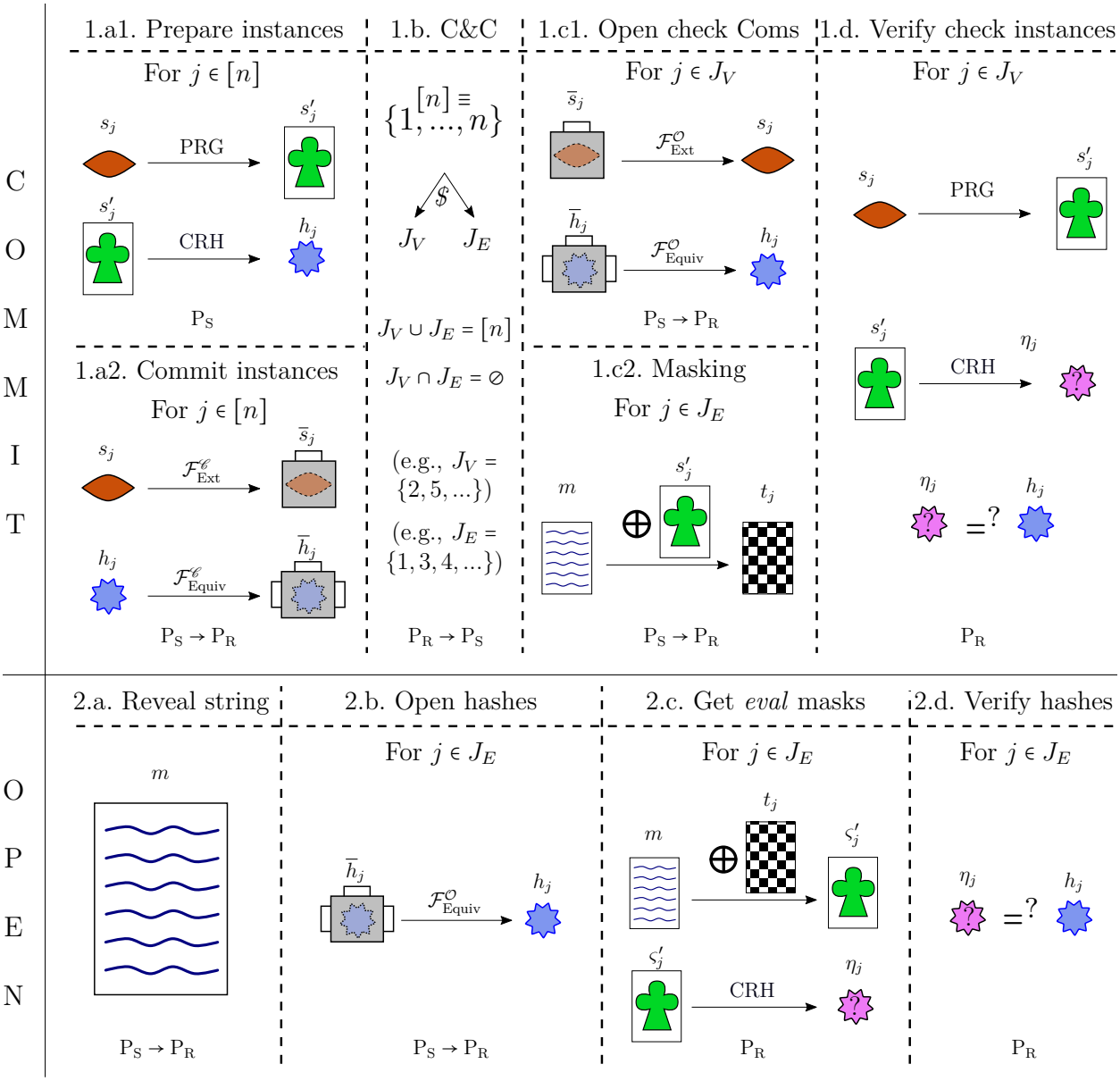


Illustration 4.1: Procedure sketch of inefficient Ext&Equiv Com scheme. Legend: C&C (cut-and-choose); CRH (collision-resistant hash); eval (evaluation); (seed s_j); (Ext-Com $\overline{s}_j$ — like a vault with a single opening); (seed expansion s'_j — like a tree growing from a seed); (hash — like a smashed paper); (Equiv-Com — like a vault with several openings); (Ext&Equiv-Com $\overline{h}_j$ — simultaneously extractable and equivocable); (message m being committed — like a text file); (message masking t_j — like a masked text file); $\mathcal{F}_{\text{Ext}}$ (ideal extractable-commitment); $\mathcal{F}_{\text{Equiv}}$ (ideal equivocable-commitment); $\mathscr{C}$ and $\mathcal{O}$ (commit and open phases of a commitment scheme); msg (committed string); P_R (receiver); P_S (sender); PRG (pseudo-randomness generator).

- **Open phase.** P_S reveals the committed string m and opens the commitments $\overline{h}_j$ of the *eval* hashes. P_R computes all tentative eval masks s'_j, one for each *eval* instance, namely

the XOR of the revealed string m with each respective masking η_j. Then, $\mathrm{P_R}$ accepts the revealed string m if and only if the (tentative) hashes η_j of the tentative masks are equal to the hashes h_j opened by $\mathrm{P_S}$. Otherwise it rejects the opening, outputting `abort`.

Analysis of regular properties.

- **Hiding.** In the *commit* phase, the string is hidden from $\mathrm{P_R}$ by one-time-pad maskings.
- **Binding.** In the *open* phase, $\mathrm{P_S}$ is bound to a single string, by collision resistance of the committed CR-Hash of the masks. Any successful opening, if possible, must open the string that for each evaluation instance is equal to the XOR of the respective hash pre-image known by $\mathrm{P_S}$ and the respective *masking*.

Analysis of simulatability properties. In spite of high communication complexity — approximately the product of the target length $|m|$ by the number e of *eval* instances — the scheme has useful simulatability properties (Ext and Equiv), as hinted in Illustration 4.2.

- **Equivocation.** In the *open* phase, the equivocator-simulator ($\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$), impersonating an honest $\mathrm{P_S}$ in a simulated execution with a malicious black-box $\mathrm{P_R}$, can open any desired fake string (i.e., not previously committed), by revealing the string and then *equivocating* the opening of the hashes of the needed masks, without revealing the inconsistent evaluation seeds.
- **Extraction (case A).** (If the Equiv-Coms of the hashes are non-extractable.)
  - **Procedure.** In the *commit* phase, the extractor-simulator $\mathcal{S}$ ($\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$), impersonating an honest $\mathrm{P_R}$ in a simulated execution with a black-box possibly-malicious $\mathrm{P_S}$, *extracts* all committed *eval* seeds. The cut-and-choose and the reception and verification of check seeds is done as in the real protocol and does not further interfere with the extraction procedure, except if the cut-and-choose leads to a direct abort, because of detecting bad check instances. Then, for each eval instance, $\mathcal{S}$ calculates a tentative mask ς'_j as the PRG-expansion of the extracted seed s_j, and uses them to unmask the eval maskings, thus obtaining a tentative string μ_j. Finally, $\mathcal{S}$ searches for a tentative string m that is consistent (i.e., the same) in a majority of the evaluation instances. If a consistent majority exists, then the extraction procedure outputs such string as the decided extracted message. Otherwise, if a majority does not exist, then the extraction procedure outputs a *delayed abort* signal; this means that $\mathrm{P_S}$ will with overwhelming probability fail to successfully open any value in the open phase; still, $\mathcal{S}$ does not abort the simulation,

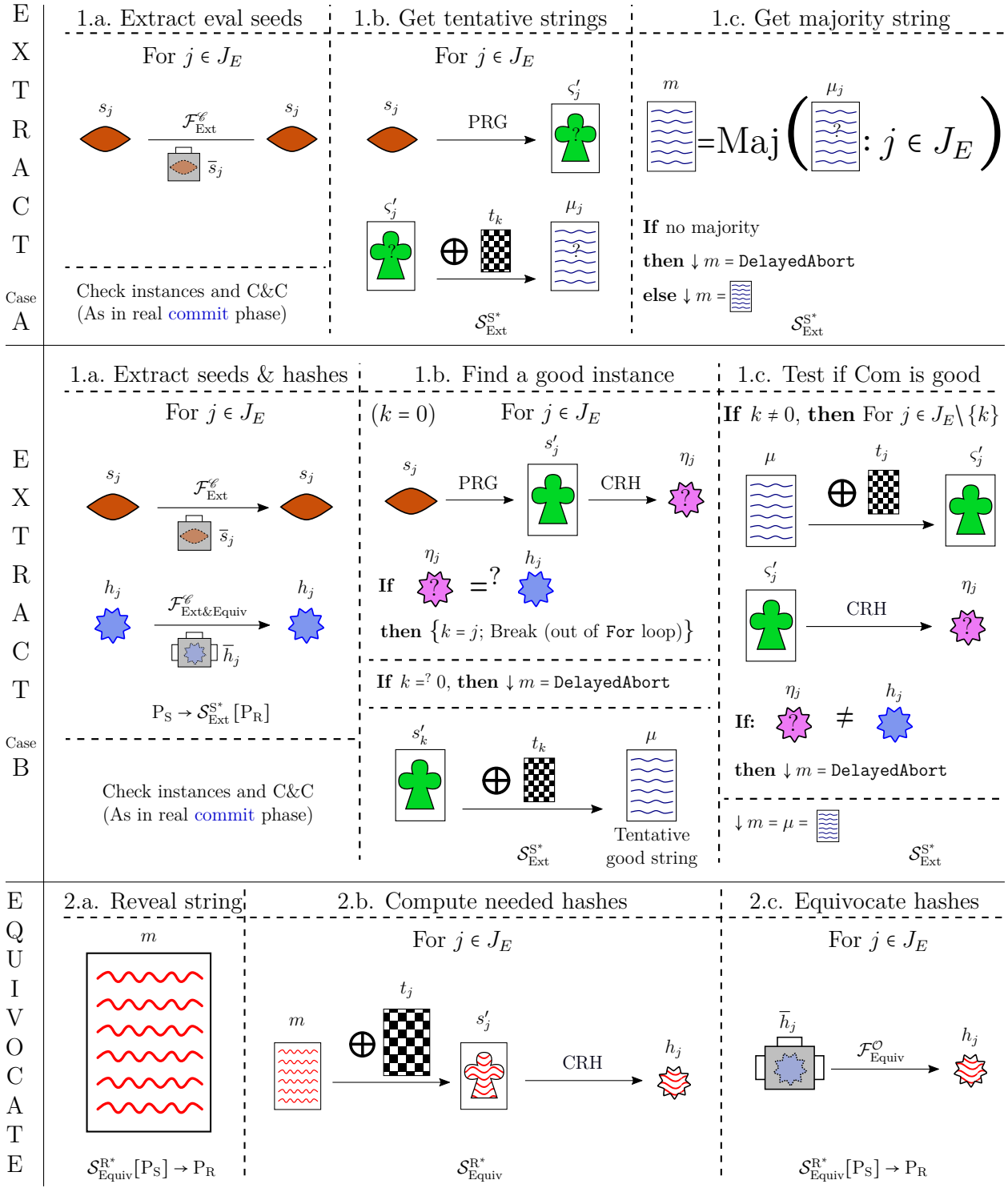


Illustration 4.2: Simulation sketch of inefficient Ext&Equiv Com scheme. Legend of Illustration 4.1 also applies. $\mathcal{S}^{p^*}_x[y]$ (simulator with goal x (Ext of Equiv), impersonating entity y (P_R or P_S, resp.) in a simulation with access to a black-box malicious P_p (P_S or P_R, resp.)). (fake message to equivocate); (fake mask); (hash of fake mask); $\downarrow m = \cdot$ (end of the extraction procedure, outputting $m = \cdot$, useful for $\mathcal{S}$ to decide what to do in the ideal world).

because an honest P_R would also not abort at this point — in the ideal world $\mathcal{S}$ proceeds by committing (via the ideal commitment functionality) to an arbitrary string.

  - **Statistical security.** The simulation may fail only if, with negligible probability in the overall number of instances, the majority value accepted by $\mathcal{S}$ is in fact different from a value that P_S is able to later successfully open. However, this circumstance would happen only if P_S would have built enough bad instances and be "lucky" that none of them were selected for check. The probabilities are similar to the case of S2PC protocols based on cut-and-choose approaches that require a majority of correct garbled circuits. For example, 123 instances would be the minimum (with 74 selected for check) for 40 bits of statistical security, i.e., so that the error probability would be less that 2^{-40}.

- **Extraction (case B).** (If hashes were committed with Ext&Equiv Coms.)
  - **Procedure.** The procedure starts similar to case 1, but $\mathcal{S}$ additionally extracts the hash of each evaluation instance. Then, for each seed it computes the respective tentative mask and from there it computes a respective tentative hash. Then, for each eval instance, $\mathcal{S}$ calculates a tentative mask ς'_j as the PRG-expansion of the extracted seed s_j and computes the respective tentative hash η_j. If the tentative hash η_j is equal to the extracted hash h_j of the respective instance j, then this is assumed as a good instance and the `For` loop (i.e., the procedure iterating across all eval instances) does not need to continue. Then, for this assumed-good instance $k = j$, $\mathcal{S}$ uses the calculated mask s'_k to unmask the respective masking t_k and thus obtain a *candidate* string μ, which is guaranteed to be the only one (if any) that may be successfully opened in the subsequent open phase. However, it might still be that P_S has maliciously acted in a way that it is not technically able to open the string, namely if the masking t_j and the committed hash h_j of other eval instances are inconsistent with the currently candidate string μ. Thus, $\mathcal{S}$ proceeds to verify if there exists consistency, as follows. For each eval instance j other than the candidate one (even instances for which the previous consistency check, in step 1.b, might have failed), P_S computes, via an XOR, what is the tentative mask ς'_j consistent with the respective masking t_j and the candidate string μ; then, $\mathcal{S}$ computes as tentative hash η_j the CR-Hash of the tentative mask ς'_j and checks whether or not it is consistent with the respective extracted hash h_j. If any verification fails in this step, then $\mathcal{S}$ knows that a future successful opening is impossible, i.e., that P_S will at most be able to "prove" that it cannot technically open any string. Still, $\mathcal{S}$ does not abort the execution, because an honest P_R would also not abort at this point, and instead assumes that this is the case of *delayed abort* (i.e., it may in the ideal world

commit to an arbitrary string). If all verifications succeed, then $\mathcal{S}$ is assured that the candidate string μ is indeed the string m that was correctly committed.

– **Statistical security.** Extraction of an incorrect value may happen only with negligible probability in the statistical security parameter (the number of instances), only if a malicious $\mathrm{P_S}$ constructs bad instances in number equal to the *eval* instances, such that the committed seed is inconsistent with the respective committed hash, and simultaneously all bad instances are selected for *evaluation* and all good instances are selected for check. This may prevent the simulator from even guessing a single candidate correct string. The statistical analysis for failed executions — number of instances vs. statistical security — is similar to the case of cut-and-choose of garbled circuits with forge-and-lose technique (Table 3.2). For example, with 41 instances and limiting the number of evaluation instances to be at most 20, the error probability is less than 2^{-40}.

Remark 4.1 (Decision of the cut-and-choose partition). The cut-and-choose partition does not need to be decided via a simulatable coin-flipping because equivocation is directly based on the equivocability of the Equiv-Coms of the hashes, which directly allow equivocation of the masks of all evaluation instances. Thus, to $\mathrm{P_R^*}$ the actions of $\mathcal{S}_{\mathrm{Equiv}}^{\mathrm{R}^*}[\mathrm{P_S}]$ "appear" as correct independently of the cut-and-choose partition.

4.2.2 Nuances of extractability

The scheme just described has extractable and equivocable properties, while still allowing a malicious sender $\mathrm{P_S}$ to commit, with noticeable probability, to (what may be called) a *delayed abort*. The delayed abort is a state in the end of the commit phase, induced by a malicious sender $\mathrm{P_S^*}$, whereby the corresponding open phase will with overwhelming probability lead an honest $\mathrm{P_R}$ to an abort, for any malicious behavior of $\mathrm{P_S^*}$ in a played open phase (i.e., except when $\mathrm{P_S^*}$ does not play the open phase). In other informal words, even if $\mathrm{P_S^*}$ would "regret" his past malicious behavior, after a successful delayed abort in the commit phase, $\mathrm{P_S^*}$ would no longer be able to successfully open a committable value (except with negligible probability). The *delayed abort* state is undetectable to $\mathrm{P_R}$ during the commit phase (or otherwise $\mathrm{P_R}$ could immediately abort, making it a regular abort), and $\mathrm{P_S}$ is later able to "prove" an incapability to open the Com.

Cases A and B have different flavors of extractability, because of the respective non-extractability vs. extractability of the Equiv-Coms of the hashes. First, when in case A the

simulator extracts an actual message, it may still be the case that P_S has committed to a delayed abort. Specifically, even if the seeds and hashes were properly committed, P_S may still decide after a successful cut-and-choose to produce one or a few incorrect maskings. Such delayed abort would be undetected to $\mathcal{S}$ in case A, but would be detected in case B. Second, for the same number of instances, case B also guarantees better statistical security.

The possibility of delayed abort motivates a distinction between several nuances of real Com schemes, depending on how delayed aborts may (or may not) affect extractability:

A. Except-if-abort extractability ($\mathbf{Ext_{ExcIfAb}}$). The simulator $\mathcal{S}^{S^*}_{Ext}[P_R]$ ($\mathcal{S}$, impersonating P_R playing with a black-box malicious P^*_S) is guaranteed overwhelming probability of correct extraction if P^*_S has not sabotaged the commit phase with a *delayed abort*. However, P^*_S is able with noticeable probability (possibly with overwhelming probability) to induce $\mathcal{S}$ to extract a committable value (and believe that P^*_S might still be able to successfully open it) when in fact P^*_S is "committing" to a delayed abort, i.e., a commitment that technically guarantees (with overwhelming probability) that a respective open phase cannot lead P_R to accept a committable value.

B. Post-verifiable extractability ($\mathbf{Ext_{Post}}$). $\mathcal{S}$ is able with overwhelming probability to detect a delayed abort and in the remainder cases extract the committed value. Since P_R does not detect a delayed abort during the commit phase, $\mathcal{S}$ does not abort in the commit phase whenever it detects a delayed abort, as it would otherwise break simulatability.

C. Pre-verifiable extractability ($\mathbf{Ext_{Pre}}$). Delayed aborts are technically prevented (except with negligible probability), because P_R detects with overwhelming probability any such attempt and thus directly aborts in the commit phase. For example, this is what is achieved in commit phases that contain a ZKPoK of a valid opening of a Com scheme with non-interactive open phase.

The cases A and B of the cut-and-choose warmup scheme described in §4.2.1 have extractability of the respective *except-if-abort* and *post-veriable* kinds. This suffices for two-party commitments (i.e., with single receiver) in the static corruption model. If a malicious P_S performs delayed abort, $\mathcal{S}$ may nonetheless commit to an arbitrary string during the commit phase in the ideal world. Then, in the open phase when P_S reveals a delayed abort (if it does), then $\mathcal{S}$ in the ideal world also requests an abort to the ideal functionality. The worlds are indistinguishable because the real protocol instructs an honest real receiver to not distinguish between a common abort (e.g., by sending invalid messages in the open phase) vs. a delayed abort. The case of pre-verifiable extractability is considered in §C.1.3.

4.2.3 Postponing the verification of check instances

Both cases ($\text{Ext}_{\text{ExcIfAb}}$ and Ext_{Post}) of the described protocol can be adjusted by postponing the verification of check instances to the open phase. This also allows reducing the number of commitments (Equiv or Ext&Equiv) of hashes, down to a single commitment of a *global hash*, i.e., of a hash of the concatenation of all masks, instead of one Equiv-Com per hash of each mask. This is possible because in the open phase the receiver is indeed supposed to learn all masks (even though it can only confirm correctness of the check instances with respect to the seeds). Conditioned to a future successful open phase, the adjusted scheme still enables, in the commit phase, an extraction procedure.

In the $\text{Ext}_{\text{ExcIfAb}}$ case the simulator can still find a consistent majority of tentative strings obtained from unmasking the eval maskings, based on the masks obtained as PRG expansion of the extracted seeds. In comparison with the advanced verification of check seeds, the delayed verification is different with respect to the effect of inconsistent check seeds, because it only leads to an abort in the open phase. Still, this difference is not relevant when focusing on $\text{Ext}_{\text{ExcIfAb}}$, because a malicious sender may in any case induce a delayed abort even if all check instances are correct and after knowing the cut-and-choose partition, by sending incorrect maskings for the eval instances. In other words, the predicat (yes or no) of ability to do delayed abort does not change; what changes is the set of ways in which to do it.

In the initial Ext_{Post} case, the adjustment of delaying till the open phase implies a slightly more complex extraction procedure. The simulator can still extract seeds and hashes and verify in advance whether or not they are consistent — any such inconsistency will denote correspond to a delayed abort. However, for the eval instances an inconsistency might not imply a delayed abort, because the respective seeds will not be opened. Thus, to ensure correct extraction in spite of the possibility that some eval masks might be inconsistent with the respective eval seeds, $\mathcal{S}$ may have to verify the tentative global hash induced (in the simulation) by every possible eval tentative string. In comparison with the extraction procedure described in Illust. 4.2, the adjusted procedure would require computational complexity larger by up to (in the worst case scenario) a multiplicative factor equal to the number of evaluation instances (which is nonetheless acceptable).

Remark 4.2 (Corrective remark about ability to do a delayed abort). The original extended abstract [Bra16, §5.4.1] contained a remark distinguishing the except-if-abort ($\text{Ext}_{\text{ExcIfAb}}$) and the post-verifiable (Ext_{Post}) nuances of extractability. After describing an

$\text{Ext}_{\text{ExcIfAb}}$ protocol using a global hash, it remarked the possibility of improving extractability via an increase of Equiv-Coms (one per instance). However, the remark overlooked that a malicious P_S that builds correct instances of Ext-Coms of seeds and Equiv-Coms of hashes (thus ensuring that the cut-and-choose check stage is validated), is still able, after learning the cut-and-choose partition, to sending incorrect maskings to P_R, thus effectively committing to a delayed abort, if the Equiv-Coms of the hashes are not themselves extractable.

4.3 Improving communication complexity

This section improves the commitment scheme (both cases A and B) of the previous section to achieve asymptotic communication rate arbitrarily close 1. Besides the PRG, the CR-Hash and the cut-and-choose involving Ext-Coms and Equiv-Coms (or Ext&Equiv-Coms) of short strings, the new protocol embeds two main ingredients:

- **authenticators**: allow the simulator to anticipate whether individual tentative eval masks (obtained as the PRG expansion of extracted seeds) are consistent with the respective maskings, thus gaining assurance about correct extraction. This is relevant for the previous $\text{Ext}_{\text{ExcIfAb}}$&Equiv scheme (case A), or for the global hash version (i.e., where there is a single committed hash) of the Ext_{Post}&Equiv scheme (case B).
- an **information dispersal algorithm** (IDA): (based on a threshold erasure code) allows *splitting* the target string into smaller *fragments*, and allows *recovery* of the original string from a sufficient portion of those fragments; based on the IDA the size of each *instance* of the cut-and-choose can be reduced proportionally to the number of instances.

4.3.1 Authenticator aid

In comparison with the $\text{Ext}_{\text{ExcIfAb}}$&Equiv-Com protocol described in the previous section (§4.2.2), based on non-extractable Equiv-Coms of hashes and Ext-Coms of seeds, statistical security can be increased by improving the ability of the simulator $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ to decide whether isolated evaluation instances are *good* or *bad*. Specifically, the technique hereafter, based on *authenticators*, allows $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ to decide whether a mask calculated as the PRG expansion of an extracted seed of an evaluation instance is incorrect or is the only one that might enable a future successful opening of the overall commitment. (Deciding that a mask is correct still does not prevent a delayed abort, because the hash committed with a non-extractable

Equiv-Com may still be incorrect and that can only be verified in the open phase.)

With the authenticator capability, still to describe, $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ will extract an incorrect string (for now ignoring the case of delayed abort) only if all *check* instances are *good* and all *evaluation* instances are *bad*, i.e., only if a malicious P_{S}^* anticipates the exact cut-and-choose partition. The new rationale about probabilities is similar to that in recent techniques that also changed the criterion of correctness regarding the number of correct garbled circuits needed in a general cut-and-choose based S2PC protocols, e.g., the "forge-and-lose" [Bra13], and other "cheating recovery" techniques [Lin13, HKE13]. Essentially, the success criterion changes from "at least a majority of correct evaluation instances" to "at least one correct evaluation instance." For example, 40 bits of statistical security can now be obtained with 41 or 123 instances, by respectively limiting *evaluation* instances to be at most 20 or 8. Since only evaluation instances are (asymptotically) relevant in terms of communication, with 123 instances this corresponds to a 6-fold reduction in communication.

The intended verifiability of each evaluation instance is achieved by using a short *authenticator* that allows $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ to verify whether or not each extracted seed is consistent with each respective anticipated tentative string. Specifically, when $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ extracts a seed and uses its seed-expansion to unmask the respective masking received from P_{S}, only two things may happen: either (i) $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ gets a correctly *authenticated* string, which must be the only one that P_{S} can later successfully open, i.e., this is a *good* instance; or (ii) $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ gets an incorrectly *authenticated* string, implying that a successful *opening* by a malicious P_{S}^* will reveal a mask different from the seed-expansion, i.e., this is a *bad* instance. Also, in order to allow equivocation by $\mathcal{S}_{\text{Equiv}}^{\text{R}^*}[\text{P}_{\text{S}}]$ (when impersonating P_{S} playing against a black-box P_{R}), the authenticator is masked by an equivocable mask.

The authenticator for each instance cannot simply be a CR-Hash of the string or of its masking, lest P_{S}^* would be able to compute in advance the offset between the authenticator of a bait string (for the simulator) and the authenticator of a target string intended for actual opening in case of luck. This offset would allow P_{S}^* to build an instance that would lead the simulator to unmask an authenticated bait string even though P_{S}^* would be able for that instance to later open a different mask that would lead P_{R} to the different authenticated target string. While this would still lead to an overall incorrect extraction with negligible probability in the number of instances (because it would require a majority of bad evaluation instances), it would not be improving over the scheme already described without authenticators.

The authenticator can be achieved with the help of a *universal hash family*, such that the

authenticator (in this case an unpredictable hash of the string) of each instance is unpredictable until the hash of the mask (or a global hash of the concatenation of masks) is committed. The goal is to ensure that P_S cannot produce a masking for which different authenticated strings can be obtained after two different unmaskings: (i) the unmasking using the PRG-expansion of the committed seed; and (ii) the unmasking using any other arbitrary mask selected by P_S before the authenticator was known. This property can be enforced by introducing a random unpredictable value (a *nonce*) that P_R discloses to P_S^* only after P_S^* becomes bound (i.e., after committing) to the seeds and hashes. This nonce acts like an identifier of the hash from the universal hash family, i.e., the authenticator becomes implemented as a non-trivial function that cannot be predicted before the input is fixed, thus making it infeasible for P_S^* to produce a masking for which two different unmaskings yield authenticated strings.

Concretely, the authenticator can for example be an algebraic field-multiplication between the nonce and a CR-hash of the string. If the image space of the CR-Hash is the set of bit-strings of some fixed length (e.g., 256 bits), the nonce can be uniformly selected from the non-null elements of a Galois field with characteristic 2, modulo an irreducible polynomial of degree equal to the hash length. This ensures that the authenticators of any two known strings (which by assumption would necessarily have different CR-Hash) would have an unpredictable offset. A successful forgery by P_S^* would require guessing this offset, in order to make the real committed mask have such (bit-wise XOR) offset with the PRG-expansion of the committed seed. (The full version of the paper discusses authenticators in more detail, including variations of assumptions — here it suffices to say that in a `strict` mode of implementation, where P_S also produces an Equiv-Com of the string, the security of the authenticator can be based solely of collision resistance of a CR-Hash function, or in some optimized procedures be based on other practical correlation-robust type of assumptions).

If assuming the availability of a NPRO, then the authenticator can be defined as a sufficiently large NPRO image (e.g., a 256-bit string) when using as input the concatenation of the CR-Hash of the string and the nonce. In fact, the NPRO can be further used (with the help of an additional commitment) to make the commit phase non-interactive altogether, letting P_S also decide the cut-and-choose partition based on the previous commitments of seeds and hash, and on the authenticator of each instance.

4.3.2 IDA support

Communication can be drastically reduced by using a threshold *information dispersal algorithm* (IDA) [Rab89]. The IDA enables splitting (i.e., *dispersing*) the original long string m into a number e of shorter fragments, such that the original string m can be reconstructed from any subset with at least a threshold number t of good fragments, each with a *reduced length*. Thus, the instances of the cut-and-choose can correspond to maskings of authenticated fragments, instead of several maskings of the authenticated full string. As the string length $|m|$ increases, the asymptotic communication complexity rate is thus proportional to the number e of evaluation instances divided by the recovery threshold t — this quotient can be made as close to one as desired.

Isolated fragments of the IDA do not need to semantically hide the original string, as would a full-fledged secret-sharing scheme [Sha79, Kra94], because in the commit phase each *evaluation* instance of the cut-and-choose only contains a masked version of the respective authenticated fragment. The IDA also does not need to support error-correction of semantic errors [RS60], because the *authenticator* mechanism gives $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ (in the role of P_R) the ability to detect errors and thus simply discard bad fragments. More simply, a threshold erasure code (t-out-of-e) can be used, so that $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ *recovers* the string m from any subset with a threshold t number of good fragments. The encoding can be based on XOR operations, with linear time complexity. The decoding only needs to be somewhat efficient, because only the simulator needs to decode; the real parties (P_S and P_R) only need to encode. A rateless code would also be possible, with appropriate probabilistic considerations — there are very efficient instantiations (e.g., [Lub02, Sho06]).

The statistical security changes again, with the new criterion for successful extraction requiring a number of *good* evaluation instances at least as high as the recovery threshold. The fragmentation also reduces the overall sum of lengths of all pseudo-randomly generated masks, i.e., of PRG outputs and CR-Hash inputs. Concrete parameters are given in Table C.1.

4.3.3 Description of protocol

For further intuition, Illustration 4.3 gives a pictorial sketch of the commit and open procedures, and Illustration 4.4 shows a sketch of the extraction and equivocation procedures.

Section C.1 in Appendix describes in detail (§C.1.1) the interactive version of the protocol, in the hybrid model with access to ideal Ext-Com and ideal Equiv-Com (or Ext-and-Equiv,

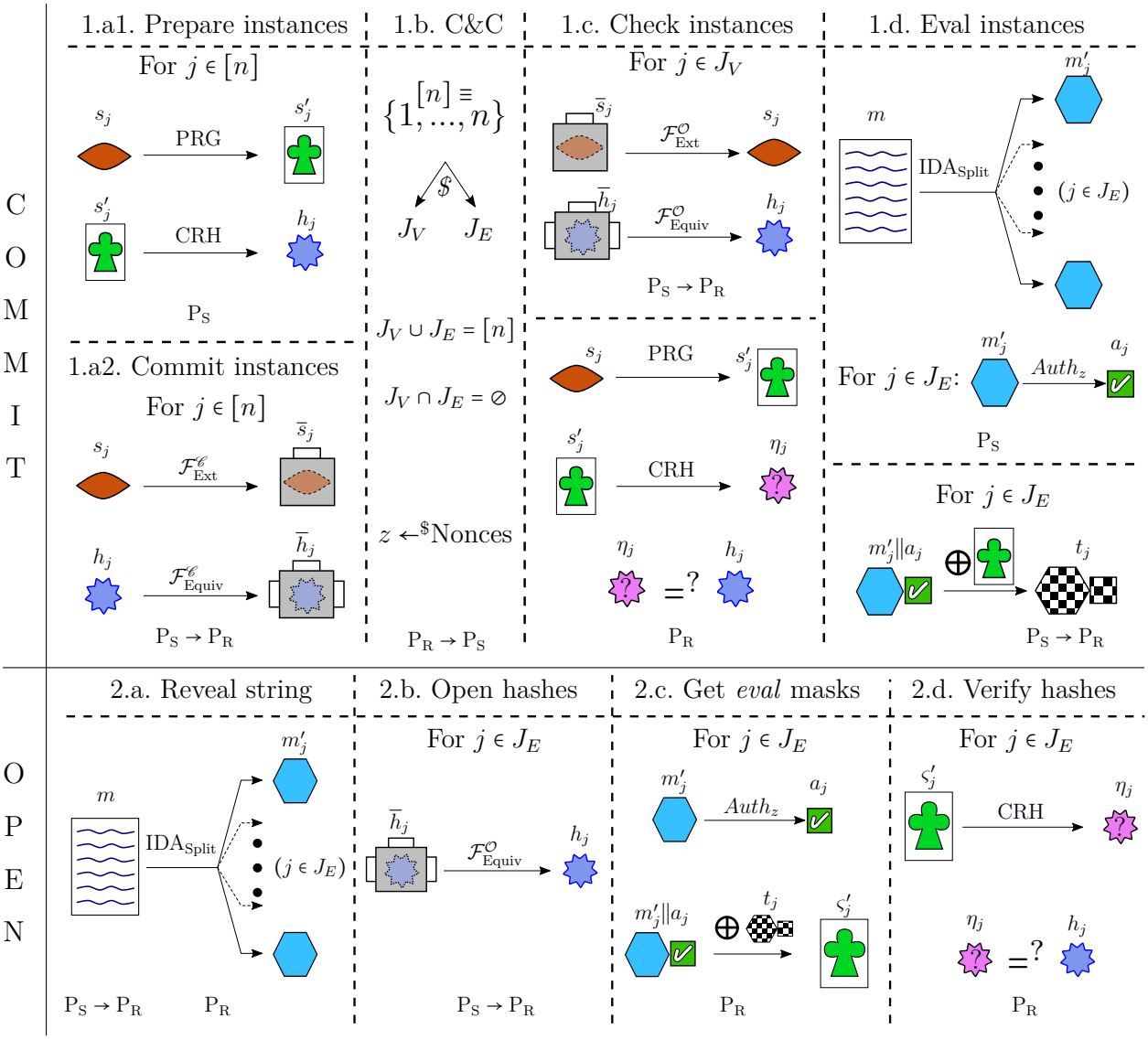


Illustration 4.3: Procedure sketch of rate-e/t Ext&Equiv Com scheme. Legends of Figures 4.1 and 4.2 also apply. (message fragment m'_j — can be combined with other fragments to recover the initial message); (authenticator value a_j — vouches for the correctness of the respective message fragment); (masking t_j of an authenticated fragment — the chess pattern denotes something masked); z (nonce for authenticator); α_z (authenticator function, AUTH).

in case B) functionalities applied only to short seeds and hashes, respectively. The simulation analysis (extractability and equivocability) is described in §C.1.2.

Remark 4.3 (An interactive version without explicit Equiv-Coms). The use of an Equiv-Com scheme with P_S as sender and P_R as receiver can be replaced by an Ext-Com scheme with P_R as sender and P_S as receiver, and a regular Com scheme (i.e., possibly neither

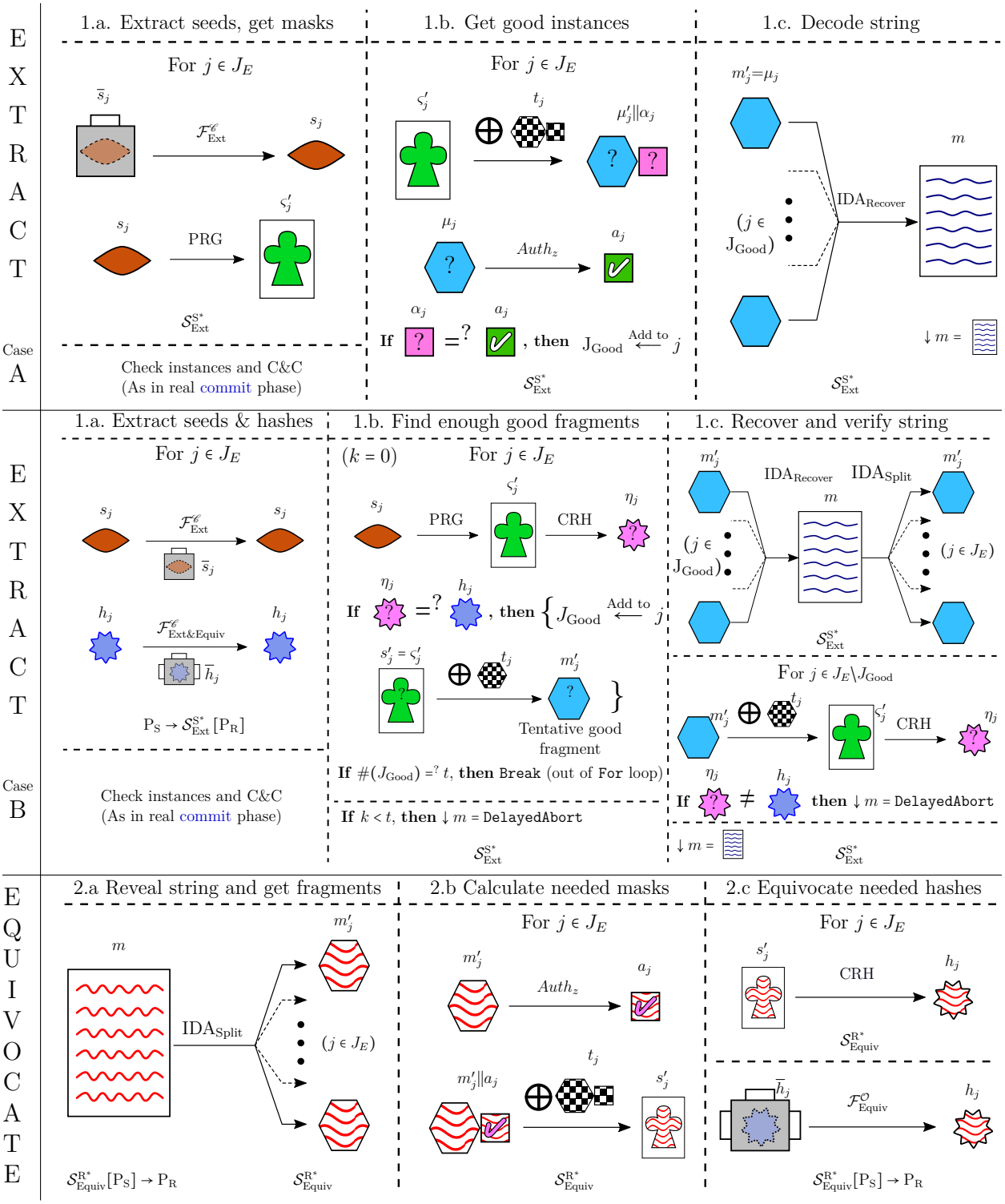


Illustration 4.4: Simulation sketch of rate-e/t Ext&Equiv Com scheme. Legends of Figures 4.1, 4.2 and 4.3 also apply. In the *extract 2* case, where the hashes were committed with an Ext&Equiv Com scheme, the authenticators are irrelevant and so are ignored.

Ext nor Equiv) with P_S as sender and P_R as receiver and further interaction (see a concrete construction in [Bra16, Section D.4]).

Remark 4.4 (Non-interactive commit and open phases). A protocol with two non-interactive phases can also be achieved, letting the cut-and-choose partition and the nonce value be computed by P_S non-interactively, based on a NPRO and an Equiv-Com scheme, similarly to how it is used for transforming interactive zero knowledge proofs into non-interactive ones §A.1.4. There, the NPRO uses as part of its input the actual commitments, which are valued once using real commitment schemes. This makes any eventual interactivity of the commitment scheme (in the commit and/or open phase) become implicit in the instantiations of the base commitment schemes (Ext and Equiv). The cut-and-choose and IDA (erasure code) parameters may have to increase, letting the statistical security parameter equate the cryptographic security parameter, to mitigate the new possibility that P_S could computationally try a brute-force trial-and-error attempt to exploit the probability of error that is negligible only in the statistical parameter. A respective protocol description is given in §C.1.4, based on a CRS.

4.3.4 Concrete configurations

The cut-and-choose and IDA threshold parameters are defined in order to match the statistical security goal and an asymptotic communication rate. Statistically, the optimal attack is to build the minimal number of bad instances that may prevent correct extraction. The attack is successful if all bad instances are selected for evaluation, and the number of remaining good instances is less than the required recovery threshold. This probability is the quotient between the number of partitions that induce error and the number of all cut-and-choose partitions.

Table 4.1 shows optimal parameters for 40, 96 and 128 bits of statistical security, and several goals of asymptotic communication rate. Asymptotically as ℓ increases, it is possible to configure the parameters to yield arbitrary high levels of statistical security and at the same time reduce the expansion-rate to values arbitrarily close to 1. For example, with $(n; e; t) = (119; 46; 23)$, the scheme achieves 40 bits of statistical security and an asymptotic communication expansion-rate $r = 2$ in the *commit* phase (the open phase always has an asymptotic rate 1). With $(n; e; t) = (775; 275; 250)$, the rate becomes $r = 1.1$, with the computed PRG output and the hash input being $r' = 3.1$ times the string length. Both r and r' can be brought arbitrarily close to 1. In appendix, Table C.1 makes a comparison with the

Table 4.1: UC commitment scheme parameters

$r_{\max}$ \ $\sigma_{\min}$	40 bits	96 bits	128 bits
∞	$n = 44$ $v = 25$ $e = 19$ $t = 1$ $r' = 44$ $r = 19$ $\sigma \approx 40.36$	$n = 100$ $v = 53$ $e = 47$ $t = 1$ $r' = 100$ $r = 47$ $\sigma \approx 96.09$	$n = 132$ $v = 68$ $e = 64$ $t = 1$ $r' = 132$ $r = 64$ $\sigma \approx 128.06$
8	$n = 53$ $v = 30$ $e = 23$ $t = 3$ $r' \approx 17.7$ $r = 7.67$ $\sigma \approx 40.19$	$n = 130$ $v = 74$ $e = 56$ $t = 7$ $r' \approx 18.6$ $r = 8$ $\sigma \approx 96.21$	$n = 175$ $v = 103$ $e = 72$ $t = 9$ $r' \approx 19.4$ $r = 8$ $\sigma \approx 128.30$
4	$n = 69$ $v = 41$ $e = 28$ $t = 7$ $r' \approx 9.86$ $r = 4$ $\sigma \approx 40.51$	$n = 169$ $v = 101$ $e = 68$ $t = 17$ $r' \approx 9.94$ $r = 4$ $\sigma \approx 96.20$	$n = 226$ $v = 134$ $e = 92$ $t = 23$ $r' \approx 9.83$ $r = 4$ $\sigma \approx 128.02$

$r_{\max}$ \ $\sigma_{\min}$	40 bits	96 bits	128 bits
2	$n = 119$ $v = 73$ $e = 46$ $t = 23$ $r' \approx 5.17$ $r = 2$ $\sigma \approx 40.00$	$n = 294$ $v = 186$ $e = 108$ $t = 54$ $r' \approx 5.44$ $r = 2$ $\sigma \approx 96.09$	$n = 393$ $v = 245$ $e = 148$ $t = 74$ $r' \approx 5.31$ $r = 2$ $\sigma \approx 128.03$
3/2	$n = 193$ $v = 121$ $e = 72$ $t = 48$ $r' \approx 4.02$ $r = 1.50$ $\sigma \approx 40.06$	$n = 474$ $v = 297$ $e = 177$ $t = 118$ $r' \approx 4.02$ $r = 1.50$ $\sigma \approx 96.03$	$n = 635$ $v = 401$ $e = 234$ $t = 156$ $r' \approx 4.07$ $r = 1.50$ $\sigma \approx 128.04$
11/10	$n = 775$ $v = 500$ $e = 275$ $t = 250$ $r' = 3.10$ $r = 1.10$ $\sigma \approx 40.01$	$n = 1902$ $v = 1198$ $e = 704$ $t = 640$ $r' = 2.97$ $r = 1.10$ $\sigma \approx 96.00$	$n = 2546$ $v = 1611$ $e = 935$ $t = 850$ $r' = 3.00$ $r = 1.10$ $\sigma \approx 128.01$

Each parametrization goal is defined by a maximum allowed asymptotic communication expansion rate $r_{\max}$ of the commit phase, and a minimum statistical security $\sigma_{\min}$ (in number of bits). The remaining parametrization, devised to achieve an actual statistical security $\sigma = \log_2(\mathrm{Bin}(n,e)) - \log_2(\mathrm{Bin}(n-b,e-b))$ not lower than $\sigma_{\min}$, and an asymptotic communication expansion rate $r = e/t$ (in the commit phase) not higher than $r_{\max}$, tries first to minimize the overall number of instances n in the cut-and-choose, and then minimize the number e of evaluation instances. From these parameters follows the optimal (and minimal) number $b = e - t + 1$ of bad instances needed to allow (though with negligible probability) an extraction error. The communication rate r is considered in asymptotic terms, with increasing length of the message being committed, thus amortizing the communication associated with the base short commitments. The rate $r' = n/t$ of the length of input and output of PRG and CR-Hashing performed by $\mathrm{P_S}$ is also considered asymptotically.

parameters of [GIKW14] and possible optimizations.

Remark 4.5 (Simulatable coin-flipping with two bits of communication per flipped coin). In a setting without rewinding, a simulatable commitment scheme applied to an arbitrary string requires communication rate at least one in each phase. Thus, a direct application in the traditional template of coin-flipping leads to communicating at least 3 bits per flipped coin. It is nonetheless possible to achieve coin-flipping with two bits per flipped coin (i.e., in an asymptotic amortized sense, as the target length increases). In a rewinding setting, this could be done with a simple protocol where one party uses an

Equiv-Com with short commitment and long opening and the other party uses an Ext-Com with long commitment and short opening [Bra16]. In a non-rewinding setting, this can be achieved with a generalized commitment, letting P_S commit (in an extractable way) to a random PRG seed and then verifiably open only the respective PRG expansion. For example, this could be achieved with a commit phase composed of an Ext-Com of a seed, and an open phase composed of revealing the PRG-expansion of the seed and giving a succinct zero knowledge argument [Gro10, GGPR13] that it is the correct PRG expansion. A different approach, based on linear-time linear-hashing and coding allows the same type of opening without a ZKP, but instead based on oblivious transfer [CDD+16].

4.4 Coin-flipping protocols

This section analyzes different ways in which to achieve simulatable coin-flipping as needed by the S2PC-with-Coms protocol. Essentially, a simulatable coin-flipping is required to allow both parties to obtain a random final outer-Com associated with each set of input and output wires of each party, while letting each party learn the respective outer-randomness associated with each set of input and output wires of the party. For a particular instantiation with a minimal round coin-flipping, the requirements change depending on which party (the first or the second) should learn the outer-randomness. Different protocols are devised for different instantiations (IFC vs. DLC).

4.4.1 Simple coin-flipping for the S2PC-with-Coms protocol

For the wire sets of P_A, the coin-flipping is merged with the decision of initial outer-Coms of P_A. Since P_B is the first party to learn the final output, it must hide his contribution to the permutation at least until P_A reveals her contribution. However, since the contribution of P_A would be needed at the same time as producing the initial outer-Coms, the later (which are already random if selected by an honest P_A) may directly be considered as incorporating the former. In other words, for these wire sets (input and output of P_A) it is enough that P_B decides the permutation between initial and final outer-Coms, if it commits to it before seeing the initial outer-Coms of P_A. The commitment has to be extractable and equivocable (as the one defined in Sections 4.3 and C.1), in order to allow simulatability of the final outer-Coms.

4.4.2 Generalized coin-flipping for the S2PC-with-Coms protocol

For the wire sets of P_B, the permutation between initial and final outer-coms needs indeed to be decided as a combination of contributions from both parties, assuming the initial outer-Coms of P_B are revealed in the first message. The simple coin-flipping is not enough, because, for simulatability reasons, P_B must learn an outer-randomness permutation, whereas P_A may only learn the respective outer-Com. This scenario is more suited to a generalized coin-flipping functionality, where a party can define a function to be applied to the coin-flipping outcome of the other party (see Section C.2.1). The intended functionality is thus one where the coin-flipping determines a random randomness, but then the output of P_A is the result of applying the Com of zeros as a function of the randomness. Different protocols are possible depending on IFC vs. DLC instantiation. (e.g., see Figure C.4).

In DLC, P_B learns two random exponents, whereas P_A learns two respective ElGamal Coms of 0. A specialized protocol is possible using as underlying commitment also the ElGamal Com scheme. Specifically, P_B uses ElGamal to commit the contribution of P_B, which is itself an exponent (randomness) of an ElGamal Com of 0. Here, it is possible to send the NIZKPoK in the clear, about the value committed by the ElGamal Commitment, i.e., without having to include the NIZKPoK inside another commitment, because the NIZKPoK does not reveal nothing about the committed randomness. In the last stage of the coin-flipping, the ElGamal Com is opened in an equivocable way (i.e., allowing equivocability to the simulator), by revealing the intended value and then forging a NIZKP that it is the correct value. The protocol is described in Figure C.6 in §C.2.3.

In IFC, P_B learns a vector of random square-roots, whereas P_A learns the respective squares. For the needed coin-flipping, P_B may commit to a vector of squares and to a NIZKPoK of the respective square-roots. Then P_A sends a random vector of square-roots, and finally P_B opens the commitment and also sends a NIZKP of correctness of the squares. Both parties can then combine the respective needed elements: square-roots for P_B and squares for P_A. This allows $\mathcal{S}$ impersonating P_B to equivocate the final opening and thus induce any desired final vector of squares, even if not knowing respective square-roots, because it can forge the NIZKPoK and NIKP. Correspondingly, $\mathcal{S}$ impersonating P_A is able to extract from the initial commitment the vector of squares and from the underlying NIZKPoK also the respective vector of square-roots — it is thus also able to induce any desired vector of square-roots. The protocol is described in Figure C.7 in §C.2.4.

Chapter 5

Privacy-preserving brokered identification

This chapter shows that S2PC with commitments can be used as a tool to resolve an apparent conflict between privacy of users and the operational utility of having a central broker mediating authentication of many users across identity providers and service providers. This serves as a complement to the technical S2PC techniques described in previous chapters, by exemplifying how secure computation may be pertinent in the design of real applications.

The chapter contains results of an analysis [BCDA15] of privacy and security problems of two nation-scale brokered identification system proposals: the *Federal Cloud Credential Exchange* (FCCX) in the United States and *GOV.UK Verify* in the United Kingdom, which altogether aim at serving more than a hundred million citizens. Considering the increasing importance of authenticating to online public/governmental services, the nation-scale brokered identification/authentication systems being proposed intend to reduce the burden of credential management by citizens, while seemingly offering desirable privacy benefits. For example, requirements derived from NSTIC [NST13, Req. 5] ask that "organizations shall minimize data aggregation and linkages across transactions" and call for "privacy-enhancing technology that ... minimizes the ability to link credential use among multiple service providers." Also, IAP ask that "No relationships between parties or records should be established without the consent of the Service User" (Principle 7.5), and that "My interactions only use the minimum data necessary to meet my needs" (Principle 4) explicitly referring to data processed at identity providers and service providers to fulfill requests "in a secure and auditable manner." This refers not only to "personal data," but also to "relationship data" that allows inferring relationship between the user and other providers.

Both systems propose an architecture where an online central hub mediates user authentications between identity providers and service providers. However, the initial analysis has shown that both FCCX and GOV.UK Verify suffer from serious privacy and security shortcomings, fail to comply with privacy-preserving guidelines they are meant to follow, and may actually degrade user privacy. Notably, the hub can link interactions of the same user across different service providers and has visibility over private identifiable information of citizens. In case of malicious compromise it is also able to undetectably impersonate users. Yet, within the structural design constraints placed on these systems there are technical solutions to the identified privacy and security issues.

Considering the great importance of technically preventing the hub from being able to track users across different service providers, this chapter is focused on showing that S2PC can be used to allow *unlinkability*, by the central authentication broker (the hub), of user pseudonyms across authentications at different service providers. This prevents the hub from building, inherently to the authentication protocol, the capability to track users across authentications. This is a relevant example of the importance of having S2PC within the conceptual toolbox of security practitioners. Other privacy and security problems related to a compromised hub are just briefly mentioned (e.g., attribute visibility, user impersonation and linkability of same-user transactions at the same service provider).

Remark 5.1 (A first step). The research on brokered identification described in this dissertation does not include a formal analysis of the proposed solutions. Instead, it is positioned as a first step in a process that should require further formalization toward a better system. Suggested future work includes using the ideal/real simulation paradigm, which will promote a more thorough reflection about privacy and security requirements, including tradeoffs between privacy and forensic abilities. Nonetheless, in the perspective of the privacy-preserving guidelines of underlying strategies, the proposed solutions already induce privacy properties that are strictly better than the current system that enables a mass surveillance capability.

Organization. An introduction has been given in Section 1.5. The remainder of this Chapter is organized as follows: Section 5.1 defines the brokered identification problem, the intervening parties and the kind of identification transactions proposed by FCCX and GOV.UK Verify. Section 5.2 infers aims and features of FCCX and GOV.UK Verify and

enumerates additional desirable system properties. Section 5.3 shows how to achieve weak unlinkability, including via an alternative protocol for S2PC-with-commitments. Several concluding remarks are elaborated in the next chapter (Section 6.2).

5.1 Background

This section describes the entities involved in hub-based *brokered identification* (§5.1.1), the role of *pseudonyms* and *attributes* in the envisioned *identification transaction* use-case (§5.1.2), and the approach followed by FCCX and GOV.UK Verify (§5.1.3).

5.1.1 The identity ecosystem

The problem of credential management arises in the context of an *identity ecosystem* composed of entities with different roles. Some of the wording below is borrowed from NSTIC and the FCCX documents [The11, Uni13].

- A *user*, also known as individual, citizen, or customer, is a "person engaged in online transactions;" the term *subject* can also be used to include non-person entities.
- A *relying party* (RP) "makes transaction decisions based upon ... acceptance of a subject's authenticated credentials and attributes." The term can be used to denote "individualized federal agency systems and applications," e.g., online government tax services, but its use can also be extended to private-sector service providers.
- An *identity provider* (IDP) is "responsible for establishing, maintaining and securing the digital identity associated with" a subject. It can be a "non-federal credential provider" approved by an accreditation authority to provide *authentication assertions* up to a certain *level of assurance* (LOA). Increasing LOA values (1, 2, 3, 4) increase the "level of confidence that the applicant's claimed identity is their real identity" — requisites vary with the country (e.g., US [FF11] and UK [CES14]).
- An *attribute provider* (ATP) is "responsible for ... establishing ... identity attributes," such as"legal name, current address, date of birth, social security number, email address."

5.1.2 Identification transactions

The identification transactions in consideration are those where a relying party (RP) identifies and authenticates a user based on the ability of the user to authenticate to an identity provider (IDP). For the RP, the goal of identification is to learn: (i) a *persistent anonymous identifier* (notation found in [Cyb11, Joh12, USP14]) of the user, hereafter simply denoted as *user pseudonym*, which is always the same when the user connects to this RP after brokered identification involving the same account at the IDP; and/or (ii) some personal attributes of the user, validated by the IDP, e.g., name, birth date, address. For the RP, the goal of authentication is to (i) gain confidence that the learned values are valid for the user with whom a session is established; and (ii) receive a *certified* assertion to that effect.

Link to a user account at the RP. A user may want to create or reconnect into a personal account at some RP. However, the user only knows her own username (e.g., an email address) at an IDP, and how to authenticate to the IDP (e.g., using a password). Internally, the IDP is able to associate that username with other identifiers of the same user. In particular, for each brokered identification scheme the IDP derives a new user pseudonym for external use with the respective hub. Since it seems that in practice a single hub is being developed for each of FCCX and GOV.UK Verify, the same symbol u can be used without ambiguity to denote the user pseudonym defined by the IDP for interaction with the hub. In both systems, this pseudonym is supposed to be pseudo-random and remain the same for all transactions with the same user.

Then, the RP learns from the hub a different user pseudonym v, persistently associated with the RP and with the user pseudonym u at the IDP, but without learning anything about the user pseudonym u at the IDP. The RP can then internally associate the received user pseudonym v to a local user account, which may contain further user information. The type of transformation and the (in)visibility of these pseudonyms is essential in determining the privacy of the scheme, namely the possible types of (un)linkability that can be inferred from user pseudonyms.

Attribute integration. As part of *identity proofing*, it may be necessary to transmit and/or verify attributes within a transaction, e.g., confirm minimal age before letting a user create an account. In the simplest case, the initial IDP (with whom the user authenticates) is able to validate the necessary attributes. In more complex interactions, attribute integration

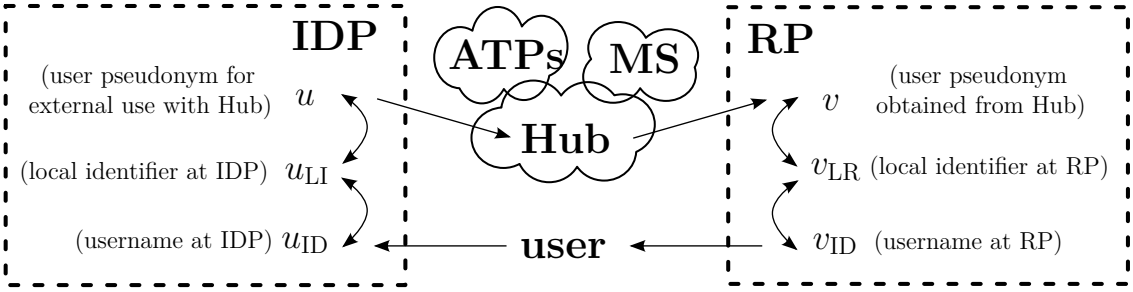


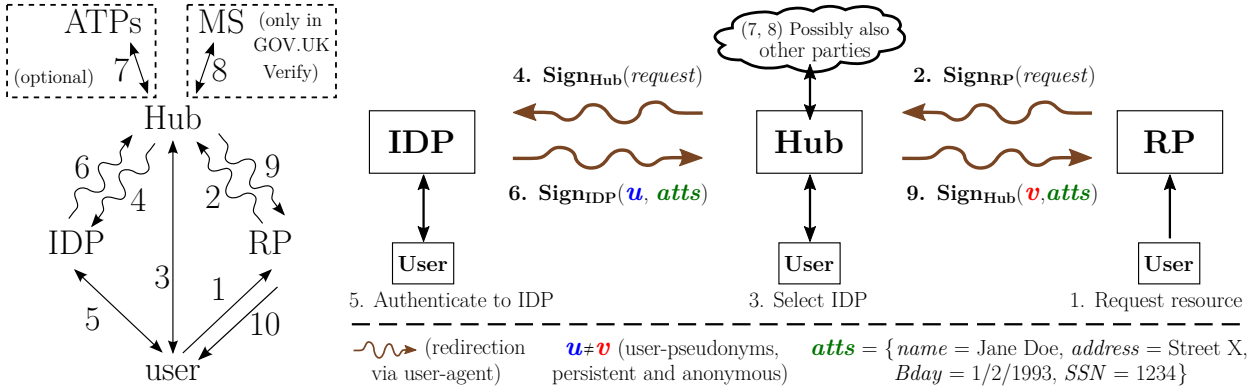
Illustration 5.1: Relation between user identifiers at IDP and RP. The new solution to prevent linkability by the hub is concerned with the transformation between u and v. For weak-unlinkability the hub should not learn anything about μ. For strong unlinkability the hub should also not learn anything about v. The other identifiers (u_{LI}, u_{ID}, v_{LR}, v_{ID}) are abstractions that remain implicit in the reminder of the analysis.

might have to involve *attribute enrichment*, i.e., attributes from different attribute providers (ATPs). For instance, a user logging into a hospital system would use the IDP to prove their identity, but could need an ATP to show proof of insurance. For simplicity, the case of additional ATPs is here ignored — this is not yet fully defined or developed in FCCX or GOV.UK Verify ([Gen14, Q&A],[Ide13, Step 9]).

5.1.3 Brokered identification

Hub and Matching Services. FCCX and GOV.UK Verify propose using a brokered identification scheme, where an online central hub actively mediates the communication and ensures interoperability between service providers (denoted as *relying parties*, RPs) and private-sector identity providers (IDPs), and possibly also involving additional *attribute providers* (ATPs). The high-level architecture is illustrated in Illustration 5.2. As a result of an *identification transaction* (links 1–10 in the figure), the RP identifies and authenticates the user. In GOV.UK Verify a *matching service* (MS) also helps validate assertions from IDPs. Both systems are based on arguable structural constraints, such as restricting the *user-agent* (a web browser) to a mostly passive role, except for selecting and authenticating to the IDP and relaying messages between other parties.

Upon receiving an authentication request from the RP (link 2) the hub helps the user choose an IDP (link 3) and then redirects the user (and the request) to the chosen IDP (link 4). Then, as a result of the user authenticating to the IDP (link 5), the IDP sends to the hub a signed assertion conveying a user pseudonym and attributes (link 6). In both systems, the hub (and in GOV.UK Verify also the MS) sees the pseudonym and the attributes in the



(a) Interaction flow **(b) Transmission of pseudonyms and attributes**

Illustration 5.2: Simplified identification transaction in FCCX and Gov.UK Verify. Legend: ***u*** (user pseudonym determined by the IDP); ***v*** (user pseudonym determined by the hub for the specific RP); **atts** (attributes); $\mathrm{Sign}_x(\cdot)$ (signature of content $\cdot$ by entity x). Metadata is left implicit (e.g., session and message identifiers).

clear. However, the systems differ in how they transform the user pseudonym between the IDP and the RP, and how they recertify the authentication assertion:

- In FCCX, the hub transforms the user pseudonym u received from IDP into a new user pseudonym v for the RP, varying with RP. The hub removes the signature of the IDP (and other metadata), re-signs the assertion, and sends it to the RP (link 9).
- In GOV.UK Verify, the hub relays the assertion from the IDP to the *matching service* (MS) indicated by the RP (link 8). The MS validates the signature of the IDP and derives a new (locally generated) user pseudonym v that is equal for all RPs that choose this MS. The MS also verifies that the locally-generated user pseudonym and attributes match to a local user account. Finally, the MS re-signs a new assertion and sends it to the hub (still (link 8)), who then re-signs the assertion and sends it to the RP (link 9).

A more detailed description of the two systems is given in the original paper [BCDA15]. Despite certain differences (namely the MS only existing in Gov.UK Verify), the two systems share striking resemblances. Indeed, it has been acknowledged that "identity services ... will need to align internationally over time" [Wre12]. The analysis herein abstracts from the differences and only takes in consideration the steps in Illustration 5.2b that take place between the IDP (steps 4 and 6) and hub and between the RP and hub (steps 2 and 9).

User limitation. A main design constraint in FCCX and GOV.UK Verify is that the role of the user-agent (a web browser) in the protocol is substantially passive. The active

participation of the user is limited to requesting a resource from the RP, selecting an IDP (from a list) and authenticating to the IDP. Communications between the RP and the hub, and between the IDP and the hub, are passively redirected through the user. The relayed authentication requests and assertions are signed by the originator and encrypted for the intended recipient. Overall, it is assumed that these mechanisms prevent network observers and the user from viewing the exchanged user pseudonyms and attributes, and/or otherwise trivial message manipulation attacks.

Types of transactions. While the exemplified identification transaction includes attributes and pseudonyms, NSTIC envisions identity solutions that enable a diversity of types of transactions, in respect to the type of anonymity/pseudonimity (and in a secure and auditable way). For example, in a transaction of type *anonymous with validated attributes*, the RP would not receive any persistent user-pseudonym, and would just obtain validation of certain attributes (e.g., an age range). In a transaction of type *pseudonymous without attributes*, the RP would only learn a persistent pseudonym, but would learn no user attributes. The solution for pseudonym unlinkability discussed in this chapter makes sense even for transactions that intend full identification (e.g., persistent pseudonym and identifiable attributes, including name and other personally identifiable information), because it is directed at preventing the hub from learning a user pseudonym defined at the IDP.

Out-of-scope alternatives. Privacy aside, linking to a user account at the RP could be achieved by a direct connection between RPs and IDPs (i.e., intermediated by a passive user), as accomplished with *OpenID Connect* [Ope]. However, this would not hide the IDP and RP from one another, which is a main explicit goal in the FCCX and GOV.UK Verify context. Alternatively, using group signatures and anonymous credentials [ISO13] would allow IDPs to sign assertions that RPs could validate as signed by an entity from within a specified group, but without knowing who. However, on its own, this approach would not provide a privacy-preserving way to transform user pseudonyms between the IDP and the RP and, without an external broker, would require more user involvement to mediate the communication between the IDP and RP. A group-signature based approach would also require group membership management and/or a mechanism for detection and isolation of compromised IDPs that could otherwise taint the trust in the system.

More interesting solutions would be possible if the user could actively aid the brokering

between IDP and RP, e.g., using cryptographic protocols based on privacy-enhancing technologies. It could take advantage of a trusted setup, e.g., tamper-resistant trusted hardware and/or open-source software authenticated by a party trusted by the user (as already happens when choosing a web browser). In spite of promising alternatives to the designs imposed by FCCX and GOV.UK Verify, the goal herein is to analyze the privacy and security problems of FCCX and GOV.UK Verify and propose repairs within their own design constraints.

5.2 System properties

The available FCCX and GOV.UK Verify documentation is incomplete in several aspects. The FCCX solicitation partially describes certain desirable privacy and security properties, but does not specify the transaction protocol. The GOV.UK Verify specification defines protocol steps in more detail, but a well-defined (formal) list of desired properties has not been found during the analysis described herein. Therefore, the set of properties that they seemingly intend to achieve (§5.2.1) is here inferred from their public descriptions.

The privacy and security of FCCX and GOV.UK Verify rely on a fully honest and uncompromisable hub. In contrast, a good solution should be resilient even against a corrupted hub, e.g., that is *curious* (about what it sees) and/or is *malicious* (about the actions it takes). For example, documentation related to Gov.UK Verify asserts that "it is important to understand the impacts that would result should the service be compromised" [CES12]. For this reason, the analysis hereafter considers two types of corrupted parties: *honest-but-curious* (i.e., acting honestly during transactions, but curious to derive information from the observed communications) and *malicious* (capable of deviating from the protocol specification). This gives rise to a number of additional desirable properties beyond those inferred from the two analyzed systems (§5.2.2).

A good protocol should also be resilient against malicious collusion, e.g., between RPs, or between RP(s) and the hub. Yet, to satisfy forensic requirements, certain special cases of collusions (e.g., collusion between the hub and an IDP, or between the hub, an IDP and an RP) may be legitimately allowed to reverse certain privacy properties, e.g., unlinkability, under very well-defined circumstances such as a specific court order targeting a given individual. Regrettably, in both FCCX and GOV.UK Verify the forensic capabilities of the individual hub could be abused and enable undetected mass surveillance, without knowledge or compelled collaboration from any IDP or RP.

This chapter focuses on unlinkability, though a complete solution should integrate other important properties, such as privacy of attributes, authenticity (i.e., resilience to impersonation), one-to-one traceability (of logs) and selective forensic disclosure. The aspects of "unlinkability" here discussed are related to user pseudonyms, ignoring linkability related to side-channel information, such as timestamps and Internet Protocol addresses. The mitigation of side-channels can be addressed in a lower-level specification and/or by prudent implementation guidelines and/or techniques outside of the scope of the protocol (see §5.4.5), but should nonetheless be explicitly considered.

5.2.1 Inferred properties

Authenticity. Upon completion of a transaction, the relying party (RP) should be assured that it has established a session with the user from whom it holds a fresh claim, and that the claims are valid (namely the user pseudonym v and attributes). A "session" can mean, for instance, an authenticated and encrypted tunnel. The root of trust for authenticity rests with the identify providers (IDPs) and with the hub or matching service (MS). Specifically, the hub/MSs trusts the authentication assertions received from IDPs and ATPs; the RP trusts the authentication assertions received by the hub/MS. In GOV.UK Verify the RP chooses which MS to trust, whereas in FCCX there is a single hub in which to rely. However, in both systems the authenticity can be broken by a malicious hub.

Edge unlinkability within a transaction. A key idea behind (privacy-preserving) brokered identification is to shield the "edges" of the authentication system (i.e., IDPs and RPs) from knowing about each other. The system is said to have *edge unlinkability within a transaction* if: (i) the IDP does not learn about who is the RP and MS; (ii) the RP does not learn about which IDP authenticated the user; and (iii, iv) the IDP and RP do not learn about the user pseudonyms at the other party.

Traceability. If an auditor challenges the legitimacy of an action taken by a party, with respect to a transaction, then the party should be able to justify it based on a verifiable preceding action. Specifically, for each authentication request sent from the hub to the IDP, the hub must have a respective request signed by the RP; for each authentication assertion sent from the IDP to the hub, the IDP must have a respective request signed by the hub; for

each assertion sent from the hub to the RP, the hub must have a respective assertion signed by the IDP; and for each user login at an RP, the RP must have a respective assertion signed by the hub. The intention of *traceability* (not explicitly discussed in the design documents) is inferred from the use of signatures in the proposed FCCX and GOV.UK Verify systems. Traceability is useful toward allowing *auditability* of the behavior of each isolated party. However, it is possible to achieve traceability more meaningfully than in FCCX and GOV.UK Verify, namely to promote better *accountability*. Specifically, *one-to-one* traceability would mean that, if the hub justifies one authentication request or assertion on the basis of another authentication request or assertion, then such justification is the only one possible [BCDA15].

5.2.2 Additional desirable properties

Table 5.1 shows several complementary properties of unlinkability and how they are or are not achieved in the analyzed systems. Section 5.3 describes one solution for weak unlinkability across RPs; Section 5.4 hints at approaches to achieve complementary unlinkability.

Unlinkability by the hub. The hub should not be able to link the same user across different transactions. Since the hub is part of the brokered identification system, this unlinkability property is required to satisfy the notion intended by NIST: "*unlinkability assures that two or more related events in an information-processing system cannot be related to each other*" [BSBC13]. NSTIC also asks that "organizations shall minimize data aggregation and linkages across transactions" [NST13, Req. 5], and IDAP principles says that "no relationships between parties or records should be established without the consent of the Service User." [Pri14, Principle 7.5]. Two weak nuances and a strong notion are considered:

- *Weak unlinkability across RPs:* the hub cannot link transactions of the same user (as defined by an account at an IDP) across different RPs. Since a user account at the IDP can be used to access many RPs, the user pseudonym defined by the IDP can be considered a global persistent identifier and thus should not be learned by the hub. Neither FCCX nor GOV.UK Verify satisfy this. In the UK, allowing global persistent identifiers conflicts with the political sensitivities that arguably lead to the rejection of identity cards [BD11]. In the US, NSTIC specifically calls for "privacy-enhancing technology that ... *minimizes the ability to link credential use among multiple RPs*" [NST13, Req. 5].
- *Weak unlinkability across IDPs:* the hub or MS cannot link different transactions facilitated

Table 5.1: Unlinkability properties across systems

A	B	C	D	E	F	G	
Properties \ **Systems**			**Direct connect**	**FCCX**	**GOV.UK Verify**	**Achievable**	1
Edge unlinkability within a transaction	RP identity is hidden from IDP		NO	YES		YES	2
	IDP identity is hidden from RP		NO	YES		YES	3
Unlinkability across same-user transactions	**Weak**	hub/MS cannot link user across RPs	—	NO		**YES** (§5.3)	4
		hub/MS cannot link user across IDPs	—	Y/N*	NO	**YES** (§5.4.3)	5
	Strong	hub cannot link user across transactions	—	NO		**YES** (§5.4.1)	6
	Edge	Change of RP is hidden from IDP	NO	YES		YES	7
		Change of IDP is hidden from RP	NO	N/Y*	NO	**YES**	8
		Colluding RPs cannot link pseudonyms	YES†	YES	NO‡	YES	9
Selective forensic disclosure#			—	NO		**YES** (§5.4.7)	10

Legend: "**YES**" is good for privacy; the two alternatives in cell E5 (Y=Yes or N*=No) are entangled with the complementary alternatives in cell E8 (N or Y*) — the second alternative (*) results from an optional *account linking* functionality [Uni13]; cell D9 (†) assumes that the IDP sends different user pseudonyms to different RPs; cell F9 (‡) considers RPs that have chosen the same MS; in row 10 (#), the property is meant in opposition to total disclosure by default.

by different user accounts at one or more IDPs leading to the same user account at a given RP. In GOV.UK Verify such linkage is performed by default by the MS (chosen by the RP), based on the user attributes. The user thus does not control who has the ability to link, nor when to allow linking — a clear privacy deficiency. FCCX offers, via an optional *account linking* feature, a tradeoff: endow the hub with the capability of *linkability across IDPs*, in exchange for allowing authentication to each RP from different accounts at IDPs. This tradeoff can nonetheless be avoided [BCDA15].

- *Strong unlinkability:* the hub cannot link transactions where the same user account at an IDP is used to access the same user account at an RP. Neither FCCX nor GOV.UK Verify satisfy this property.

Edge unlinkability across transactions. Edge unlinkability has been discussed in the context of a transaction; the notion can be extended *across* transactions as follows:

- *Across two transactions with the same user account at an IDP:* the IDP does not learn whether the accessed RP has changed or not. This property can be inferred from the FCCX and GOV.UK Verify designs.
- *Across two transactions with the same user account at a RP:* the RP does not learn whether the assisting IDP has changed or not. This property is achieved in FCCX when using an "account linking" option, but with a privacy tradeoff that could be avoided in a more sophisticated solution [BCDA15].
- *Across transactions with the same user account at an IDP but different RPs:* (i) several colluding RPs cannot link the user based on their lists of user pseudonyms; (ii) if several RPs colluding together know, from an external source, that respective user pseudonyms correspond to the same user, they are still not able to predict anything about the user pseudonym at another RP. This is satisfied in FCCX, but not in GOV.UK Verify where different RPs (that have chosen the same MS) receive the same user pseudonym.

Attribute privacy. The visibility of personal identifiable information, namely attributes, should be reduced to the bare minimum necessary for the purpose of each party and as consented by the user. For example, RPs should learn nothing more than "necessary" and requested (e.g., an age predicate, instead of a date of birth). Thus, there should exist capability to deal with predicates of attributes, but the actual definition of what is "necessary" is outside the system model considered in this analysis. Since the role of the hub is to help the interoperability of transactions, supposedly without interest about user information, it should not have visibility into the attributes being exchanged or verified. This is required by the FCCX procurement, but is not achieved [Gen14, Q&A 2.3]. The GOV.UK Verify specification simply defines a protocol where the hub and MS have visibility of attributes. In GOV.UK Verify the MS explicitly uses attributes to help link the user into a local account.

Resilience against impersonation. In spite of the trust placed in the hub to broker a transaction, a maliciously compromised hub should not be able to break authenticity. It should not be able to gain access to a (honest) user account at a (honest) RP. However, in FCCX and GOV.UK Verify a compromised hub is able to impersonate users.

Selective forensic disclosure. NSTIC [NST13, Req. 22] and IDAP [Pri14, Princ. 9] contain provisions about forensic capabilities and exceptional circumstances. They consider "forensic capabilities to ... permit attribution" and possible "exemption from ...[other] Principles that [the exemption] relates to the processing of personal data" if "necessary and justifiable in terms of" foreseen exceptions to the "Right to respect for private and family life" [Eur10]. With this in mind, a desirable property might be to have the ability to do a limited reversal of weak or strong unlinkability (or attribute privacy) in special cases where a subset of entities are compelled to aid in an adequate investigation, e.g., following a subpoena.

Assume the hub pinpoints a transaction related to a certain triplet (IDP, u, RP_1), where "user" is an agent with user-pseudonym u not known by the hub. Two types of selective forensic disclosure are envisaged below:

- **Coarse-grained.** Compelled collaboration of the IDP may allow the hub to gain full linkability of past (logged) transactions of the selected user with any RP (i.e., pinpoint all such transactions), but without affecting the unlinkability of other users.
- **Fine-grained.** Compelled collaboration of the IDP and some RP_2 with the hub may allow the hub to pinpoint past (logged) transactions of the same user with this RP_2, but (i) without IDP learning who is RP_1 and RP_2, (ii) without the hub learning about any other transactions of the user with any RP other than RP_2, and (iii) without breaking unlinkability of other users. In other words, edge unlinkability is preserved and weak unlinkability is selectively broken to investigate a user in connection to one or several selected RPs, without leakage about interactions with other RPs. If RP_1 is RP_2, then the IDP does not need to collaborate.

In sharp contrast, in FCCX and GOV.UK Verify both types of leakage happen by default and without any need for collaboration. This is a serious vulnerability, and could open the way to undetected mass surveillance.

5.3 Achieving weak unlinkability across RPs

In FCCX and GOV.UK Verify, the hub has full visibility of the user pseudonym (u) defined by the IDP. Thus, the hub — and whoever can gain control over it, legitimately or not — can link transactions of the same user, as defined by a user account at an IDP, across different RPs. This gives the hub excessive visibility into the activities of all users. It also

violates the selective forensic disclosure property, by allowing linkability without the help of the respective IDPs or RPs. Even if there are provisions about not storing some data (e.g., in GOV.UK Verify), a system should be at least secure in an honest-but-curious adversarial model, where seeing is equivalent to storing. In FCCX, the hub is supposed to log activities related to all transaction steps for at least one year online and 7.5 years offline [Uni13]. In any case, clear and specific information should be made public about existing security-related logging.

This section proposes a solution for weak unlinkability (by the hub) across RPs, involving interaction between IDP and the hub. The solution is directly applicable to FCCX, but not to GOV.UK Verify where the user pseudonym is transformed by the MS. Anyway, the proposal in the original paper [BCDA15] is that the GOV.UK Verify structure be changed to integrate this solution, since it is also shown that the MS does not ensure authenticity against a malicious hub, and poses additional threats against unlinkability.

The subsequent analysis assumes for simplicity a transaction without attributes. In fact, a privacy preserving solution should also hide the attributes from the hub, and this may be considered as a complementary problem.

5.3.1 Initial intuition

Linkability by the hub of user pseudonyms across RPs can be avoided by hiding u from the hub, and with the hub learning v (the user pseudonym to send to RP) as a pseudo-random value. The pseudonym v is persistent for each pair (u, r) composed of the user-pseudonym at IDP and of an identifier r of the RP, defined by the hub for interactions with the IDP. Also, the IDP must not learn anything about r and v. This is an instance of a secure-two-party computation (S2PC), where the hub learns a function of combined inputs but neither hub nor IDP learn the input or output of the other. The IDP provides u as input; the hub provides r as input and receives v as output.

The function can be a block cipher, with the key being the user pseudonym u held by the IDP, and with the plaintext being the RP identifier r held by the hub. By definition, the output v of a secure block cipher is indistinguishable from random, if the key (u) is random. The RP identifier r must be unpredictable by the IDP so that the output v is also unpredictable by the IDP.

For example, (Yao-protocol-based) S2PC of the AES-128 block-cipher is a *de facto*

standard benchmark in S2PC research [DLT14]. The respective circuit requires 6,800 garbed gates [Bri13], which is about 13 times less than for a SHA256 circuit. While a S2PC-AES execution is slower than a simple exchange of signatures, available benchmarks show that its computation can be achieved under a second [FJN14b]. Furthermore, after the user authenticates to the IDP, the IDP and the hub can execute the S2PC "behind the scenes," without user intervention, i.e., unlikely to add a usability burden.

If user pseudonyms are 256 bits strings, then a single AES-128 cipher evaluation is not enough and actually other block-ciphers may have circuit designs that enable even more efficient S2PC implementations. For example, one of the "LowMC" ciphers proposed by Albrecht et. al [ARS+15, Table 1, case 15] for 256 bits of security, with block-size and key-size of 256 bits, requires only 1374 AND gates. Using these parameters in a DLC instantiation based on ECC would enable reducing communication to about 1 Mega Byte, for an oblivious evaluation of the block-cipher, with each party (the IDP and the hub) also receiving ElGamal Coms of inputs and outputs, if assuming a statistical security of 30 bits, with a cut-and-choose with 98 instances and limiting the number of evaluation instances to at most 6. While the mentioned cipher is not standardized and needs more security analysis (as mentioned by the authors) [ARS+15], it does at least highlight the potential for communication improvement based on the design of the cipher.

Recent techniques allow a further amortization of the computational complexity of S2PC in a multiple execution setting, i.e., when evaluating many times the same circuit, and allowing various tradeoffs between offline and online phases [HKK+14, LR14]; this applies here as the hub and each IDP are involved in many transactions.

Even if S2PC of a block-cipher encoded in a Boolean circuit is not the perfect solution to achieve weak unlinkability across RPs, it is a proof-of-possibility demonstrating that weak unlinkability is achievable within the imposed constraints and thus ought to be required in privacy-preserving brokered identification systems.

5.3.2 Adding traceability

To achieve traceability, the IDP should also sign an assertion that allows the hub to justify subsequent actions related to v in this transaction. While theoretically this could be achieved by embedding a signature calculation within the generic S2PC module, it would prohibitively increase the cost of the computation. To avoid this, signatures can be computed outside of

the S2PC module, as follows. Each party signs and receives a signature of commitments that hide and bind the parties to the respective inputs used and the outputs obtained in the S2PC. For example, given the secrecy constraints, the IDP does not sign the unknown pseudonym v, but rather a commitment efficiently obtained in the S2PC. The commitment hides v from the IDP, but binds the hub to the value, preventing association with any other value. The process is also applied to the inputs of both parties, to commit the hub to a RP identifier (r), and to commit the IDP to the user pseudonym (u). These commitments need not be *opened*, except in an eventual audit that so requires.

The needed commitments can be obtained by a specialized S2PC protocol that directly provides commitments of the inputs and outputs of both parties, which is exactly the case of the S2PC-with-Coms protocol defined in this dissertation. (The original paper [BCDA15] also proposes an alternative (non-simulatable) protocol for S2PC with commitments, applicable to any generic black-box S2PC protocol of Boolean circuits, and any type of commitments. The mechanism therein uses external commitments as complementary input to a S2PC of a block-cipher circuit augmented with some embedded universal-hashing operation. There, a party may still use in the S2PC inputs different from those committed, but traceability is not affected because an audit would detect an inconsistency with overwhelming probability.)

5.3.3 Analysis

Security properties. Unlinkability follows directly from the hiding properties of the commitments and of the block cipher. By the S2PC properties, the IDP learns nothing about v or r. By the pseudo-randomness of a block-cipher with a random secret key, the hub learns nothing about u. Traceability for each party follows from the signature by the other party, containing values that the party can prove being unequivocally associated with only one particular input and/or output. Specifically, in a later audit phase, a party could verifiably open a commitment of her input used and/or output received in the S2PC.

Basic audit example. Consider an authentication assertion sent from the hub to the RP (link 9) in Illustration 5.2b, containing a pseudonym v and a session identifier n. (Attributes are ignored, as their transmission is complementary to the problem of transformation of pseudonyms). If an auditor challenges the hub about this transaction, the hub can prove correctness of behavior as follows. First, it shows the respective assertion signed by the IDP

(link 6), which (among other elements) should contain a different session ID n', and the commitments of inputs of both parties and commitments of output of the hub. Second, the hub proves a correct relation between the two session IDs and that the IDP is the expected one, e.g., based on a traceability solution that ensures that the IDP and the RP see different session identifiers, but which allows the hub to prove that the identifieres are related to the same transaction and the correct entities — see [BCDA15]). Third, the hub opens the commitments of his input and output, allowing the auditor to verify that the output is indeed the user pseudonym at the RP, that the input is a valid identifier of the RP, and that the respective commitments have been signed by the IDP.

Privacy-preserving audit actions. The above described basic audit action is not exactly privacy-preserving, as it requires the hub to reveal the identifier it used for the RP. This would allow a malicious auditor to later correlate this information with other audits. An alternative could be for example for the hub to simply prove that his committed input is a collision-resistant hash whose pre-image starts with a prefix equal to the RP public identifier (and then followed by an unpredictable string never revealed).

As another example, it is conceivable an audit action that asks the IDP to prove that in certain two transactions the user pseudonym u is the same (or different), but without actually revealing the user pseudonym(s). This would be efficient with commitments based on group operations (e.g., ElGamal commitments). Nonetheless, it would still be possible even for commitments based on symmetric primitives, using efficient zero knowledge proofs based on garbled circuits [JKO13]). The wide range of possible considerations corroborates that it would be useful to have available a specification of envisioned auditability use-cases, in order to better ponder possible alternatives of commitment schemes.

Formalizing a solution. The discussed weak-unlinkability solution is more privacy-preserving than what is proposed by FCCX, but it does not yet reach an adequate level of formalization for implementation. First, a proper solution should include an integration with other properties, as those discussed in the next section, and should be validated by a proof of security. Second, the kind of adversarial scenarios one may want to protect against should be further considered. In a motivating use-case scenario, two different user-pseudonyms (v_1 and v_2) at an RP may be leaked to public information, perhaps due to some unintended data breach related to an audit. Then, the IDP can easily find to which users (i.e., to which

user-pseudonyms at the IDP) they correspond, using the following simple procedure. For each leaked user pseudonym at an RP, the IDP builds a list of tentative RP identifiers (r), by inverting the block-cipher with each known user pseudonym u at IDP. Then, comparing the two lists, IDP finds the unique common element, which is the RP identifier used by the hub (and which should be unpredictable to IDP), and learn the respective user pseudonyms (u_1, u_2) at IDP (i.e., of the users affected by the data breach). Furthermore, this alone also lets the IDP predict any other user-pseudonym at the same RP. This particular issue does not affect the previously defined properties (Section 5.2), but it does not allow a gracious failure when there is a leakage of user pseudonyms at an RP. While this particular problem can be resolved by changing the block-cipher application, e.g., using as key an XOR of the two inputs, or using a blockcipher-based cryptographic hash function [PGV94, BRSS10], the example conveys the benefit that would arise from further formalization.

5.4 Other aspects of unlinkability

5.4.1 Strong unlinkability across RPs

In the above proposal for weak unlinkability across RPs, the hub still knows the user pseudonym (v) sent to the RP. This allows the hub to link the same user across different transactions associated with the same RP. In a more privacy-preserving solution, the hub would never receive a persistent user pseudonym. As an initial approach, this can be solved based on an ephemeral, secret and random key or mask m, shared between the IDP and RP and hidden from the hub (see §5.4.4). It is ephemeral in that it is generated for a one-time use, i.e., once for each transaction. Its secrecy can be derived from an appropriate key-exchange protocol. Its randomness can be enforced by the hub, via efficient two-party coin-flipping. Then, the hub and IDP implement a S2PC-based solution, similar in spirit to what was described for weak unlinkability, but with the following differences (here leaving implicit needed augmentations for traceability): the IDP also inputs the shared key into the S2PC; the hub inputs into the S2PC an authenticated enciphering of the RP identifier, as received from RP — calculated by the RP using the shared key; the S2PC deciphers the RP identifier, if and only if the shared key inputted by the IDP is the same as the one used by the RP — if the internal verification fails, then the S2PC outputs an error bit and nothing else; the hub then learns a masked and authenticated version of the user pseudonym for the RP, instead of

the value in clear. Then, the hub sends the output to the RP, who can verifiably open the persistent user pseudonym.

In the above-mentioned approach the ephemeral encryption key is common between the IDP and RP and so can be used to link the user accounts at IDP and RP if an honest-but-curious IDP and RP collude offline sometime after a transaction. This kind of linkability might be considered secondary in comparison with linkability by the hub, first because it requires collusion between the IDP and the RP outside of the protocol, and second because such linkability may already be facilitated by other identifiers, e.g., the timestamps at the IDP and RP are already very closely related, assuming the full transaction is performed in real time. Yet, it is possible to augment the mentioned approach to prevent the ephemeral key from being common at the IDP and RP. The hub may actively mediate the key-exchange and subsequent communication as a "blind" but active man-in-the-middle, such that the IDP and the RP learn different keys with certain homomorphic properties and the hub learns a kind of "homomorphic offset" that enables homomorphic re-encryption of messages between the IDP and RP, and vice-versa, but without learning any of the keys of the IDP and RP, nor being able to decrypt any message. Then, the S2PC module would receive from the IDP the user pseudonym and the local ephemeral key of the IDP, and receive from the hub the homomorphic offset and an authenticated-enciphering of the RP identifier encrypted with the ephemeral local key of the RP. As a result, the S2PC would return to the hub an authenticated enciphering of the user-pseudonym for the RP, using as key the key of the RP.

5.4.2 Unlinkability against colluding RPs

In FCCX, each RP receives from the hub a different and uncorrelated user pseudonym v. However, in GOV.UK Verify such user pseudonym depends only on the MS and on the user pseudonym at the IDP. In other words, v does not change with the RP (assuming the same MS). Thus, two externally colluding RPs (with the same MS) can trivially link the same user in different transactions across the two RPs. This can be avoided by letting v vary pseudo-randomly with the RP. If the MS were to know the RP, which would be detrimental to weak unlinkability, a trivial solution would be to calculate v as a value varying with RP. If the MS does not know the RP, then the user pseudonym could be changed at the hub (e.g., as proposed in Section 5.3, also achieving traceability). In GOV.UK Verify, the protocol is geared towards unequivocal identification, not selective disclosure, as it assumes a matching

dataset (MDS) of attributes. Thus, this solution in isolation (i.e., varying the user pseudonym with the RP) would not make linkability impossible, but only not easier than what is already possible without brokered identification.

5.4.3 Weak unlinkability across IDPs

Full unlinkability requires solving also the case of authentications across IDPs.

Multiple options for connection. It may be desirable to let a user access the same user account at RP via different accounts at IDPs. This can be achieved after a *matching registration* process at each RP, as follows: (i) first, the user connects using a certain user account at an IDP (i.e., a regular transaction); (ii) then, without disconnecting, the user and the RP proceed to a new transaction, with the user reconnecting but through a different account at the same or a different IDP (the same user may have different accounts at the same IDP); (iii) depending on the requisites of the user account at RP, the RP may perform a matching of user-attributes to guarantee, with the adequate level of assurance, that the different connections correspond to the same user; (iv) if the matching is successful, the RP links the two different user pseudonyms into the same local identifier. Thereafter, a transaction through any of the accounts at IDPs leads to the same account at RP. However, this procedure breaks *edge unlinkability in regard to changing IDPs*, i.e., the RP knows when the user is changing IDPs.

As an alternative, in FCCX an "account linking" option is provided for a user to link different user accounts at IDPs into a single local account at the hub. It is based on the above registration procedure, but replacing the RP by the hub. The change of IDPs is thus automatically hidden from all RPs, because the user pseudonym that the RP receives from the hub does not vary with the IDP. However, this solution breaks *weak unlinkability across IDPs* and it is further incompatible with weak unlinkability across RPs, as it explicitly allows the hub to link the user to a unique identifier. In other words, in FCCX the feature of allowing the user to connect to the same account at RP via several different accounts at IDPs comes at the cost of disallowing weak unlinkability.

In GOV.UK Verify, the MS has the task of matching into a local account the user pseudonym (u) and user-attributes ($atts$) received from the IDP. Thus, by default the MSs also break weak unlinkability across IDPs, besides breaking edge unlinkability in regard to

RPs that choose the same MS. Even though the MS is instructed to only save a hash version of each u, all are linked to a local identifier. This is compulsory, without choice by the user.

A privacy-preserving solution. The above mentioned tradeoffs can be avoided with a different solution that avoids linkability across different IDPs and at the same time allows the user to connect through multiple user-accounts at IDPs. Informally, the best of both worlds is possible — enabling a user to authenticate to the same RP through different accounts at IDPs, such that: (i) the hub cannot link the same user across different transactions; (ii) the RP does not know whether the same or a different IDP is being used; (iii) a single *matching registration* is enough to determine the default behavior for all RPs, but the user can avoid the feature on a per-connection basis; (iv) the user chooses which party to trust with the matching operation, instead of being imposed the hub (in FCCX) or (unknown) MSs chosen by RPs (in GOV.UK Verify). A solution outline is given in the original paper [BCDA15], based on an *identity integrator* service, which may be logically separated from the remaining actions but in practice may be offered by a kind of IDP.

5.4.4 Attribute privacy

If the role of the hub is simply to facilitate the identification transaction, supposedly without interest about private information of the user, then it should not see user attributes flowing between the IDP and the RP. However, this is not the case in the inferred authentication protocol depicted in Illustration 5.2. A solution approach to this problem can be based on encrypting the attributes in a way that the hub cannot decrypt but the RP can. Intuitively, one solution could be based on (the well known) Diffie-Hellman key exchange (DHKE), which would provide a secret key between two parties, without any passive eavesdropper being able to learn it. However, there are some difficulties. First, a good solution should be resilient to a man-in-the-middle attack by a possibly malicious hub. This kind or problem is typically thwarted with an authenticated DHKE, where both parties authenticate the exchanged messages. However, this direct approach is not possible because the IDP and RP do not know the public key of one another, lest it would break their pairwise anonymity. Instead, an auxiliary-channel DHKE can be performed, with the user transmitting a short string (e.g., 8 alphanumeric characters, renewed) between the RP and IDP, such that an active man-in-the-middle attack can only succeed and remain undetected if the hub guesses in advance the string of the ongoing session.

While the described solution is not directly related to generic S2PC, its actual implementation should consider an integration with the S2PC-based solution for the pseudonym unlinkability problem. For example, an ephemeral secret key exchanged between IDP and RP constitutes on itself an identifier of the transaction, which a colluding pair IDP+RP could hypothetically use to correlate the steps of the identification transaction. Whether or not such linkability is relevant is another matter. In fact, it is likely that a collusion of RP and IDP would already allow a correlation by timestamp of the transaction. Nonetheless, a more complex solution may allow the hub to actively intermediate the key-exchange protocol in a way that it effectively guarantees that the RP is able to decrypt a message using a key that is not correlatable with the key that the IDP as used to encrypt it.

5.4.5 Unlinkability of other identifiers

Side-channels or covert channels can be used with or without intention to leak identifiers that allow some types of linkability.

Pseudonym unlinkability vs. user-agent identifiers. Even with an inherent weak and strong pseudonym-unlinkability solution with respect to the authentication protocol, the hub may achieve via side-channel information the ability to link users across transactions. For example, on a typical usage a user may keep invariant her own Internet Protocol address and/or other information leaked about the user-agent (e.g., identifiers of the web-browser and plugins). The techniques to reduce such side-channel information are outside of the scope of the solutions described herein (e.g., connecting through an anonymized network, and using software that shields the amount of leaked information). Nonetheless, it is clear that it would be useful to have such concerns be made explicit, e.g., by specifying the records kept by the hub, and requiring that the users be informed of the respective potential for linkability.

Edge unlinkability vs. attributes. Several side-channels also exist between the IDP and RP, e.g., in the encoding of attributes and contextual records. Thus, it would be useful to devise explicit requirements about normalization, or active sanitization by the hub or MS, to avoid such side (or even covert) channels. They can for example be avoided by enforcing secure comparison of attributes, instead of direct transmission. If the RP already knows the candidate attribute values then the hub can mediate a secure comparison, by comparing

randomized hiding commitments of the attributes, as submitted separately by IDP and RP.

Edge unlinkability vs. session ID. SAML specifies that the authentication request ID n must be unique [OAS05, §1.3.4], but does not specify that it cannot be linkable to the issuer. Thus, if it is an invariant across a transaction, i.e., the same at RP and IDP (as required in GOV.UK Verify), then it can be trivially used to leak to the IDP who the RP is, thus breaking edge unlinkability. This could happen even in compliant implementations, e.g., if it is a counter (likely to be different across RPs and thus acting as a pseudonym), or even a concatenation of the RP identifier and a counter. A malicious RP could further define n as an enciphering of a randomized version of its own public identifier, thus revealing itself to an IDP possessing the appropriate key; no auditing by the hub (oblivious to the key) would distinguish this request ID from a genuinely random one. One solution against this is to use an efficient coin-flipping protocol between the hub and RP, to enforce a random session ID. Alternatively, and additionally avoiding linkage through session IDs even if comparing the databases of the IDP and RP, the hub may use with IDP an authentication request ID (n') different from the one (n) received from RP. Care is needed in this case to ensure traceability.

5.4.6 Traceability

The signatures exchanged between parties in FCCX and GOV.UK Verify provide some level of traceability if the hub is honest. If questioned by an auditor about a certain authentication assertion or request, the hub can show a related assertion or request (assuming respective logs are recorded). However, for the same request from RP or same assertion from IDP, a malicious hub may produce several requests and assertions. For example: (i) in GOV.UK Verify, the hub could undetectably send the assertion to two different MSs, to illegitimately obtain two user identifiers; (ii) in FCCX, the hub could undetectably produce assertions for two different RPs; (iii) in FCCX and GOV.UK Verify the hub could collude with a rogue IDP, to obtain a respective assertion, besides the legitimate one. Later, in a limited audit the malicious hub could use the most convenient justification, while hiding the legitimate and/or other illegitimate ones. This can be improved with a property of *one-to-one* traceability. The intuition is that each party commits to the details of the next action, before receiving the "justification" (i.e., the signed material) referring to the previous action. The commitments need not be opened during the transaction, but only in case of auditing. §5.3.2 has a sketch on how to achieve this in connection with weak unlinkability, with respect to pseudonyms.

5.4.7 Selective forensic disclosure

The coarse-grained nuance is trivially possible by the proposed solution for weak and strong unlinkability. For that, a compelled IDP just needs to let the hub know of all transactions associated with the same user account at IDP. While this breaks weak unlinkability for the user, it preserves the privacy of other users and so it is already strictly better in comparison with FCCX and GOV.UK Verify, where selectiveness of forensic disclosure is not possible (as linkable identifiers are leaked to the hub by default in all transactions). The fine-grained nuance is possible via a different procedure, based on a transaction flagged as a forensic investigation. Given a pinpointed transaction with (IDP, user, RP_1), and a targeted RP_2, the hub initiates a forensic transaction between RP_2 and IDP. This is a regular transaction but with two main differences. First, since the IDP is collaborative, it interacts with the hub as if the actual user (defined by u) had authenticated — as a result (and assuming the solution for strong unlinkability) the RP learns (but the hub does not) the respective user pseudonym at the RP. Second, the RP receives information that this is a forensic transaction — the information is signaled through the IDP, independently of the hub, in order to prevent a malicious hub from actually impersonating the user. This allows RP to control whether or not (depending on the subpoena) to give the hub access to the internal user account. Then, a collaborative RP may inform the hub about the past transactions (i.e., their session IDs) involving the same user account.

Authenticity against malicious hub. A maliciously compromised hub is capable of impersonating users in order to access their accounts at RPs. For example: the RP is not ensured that the final user is the agent that initiated the protocol; since the hub learns the user pseudonyms needed by RPs, it can reproduce assertions that would be accepted by them; for each user authentication with an IDP, the hub can concurrently continue the transaction with several RPs. Intuitively, the problems can be solved with techniques that: allow the RP to verify that the initial and final users are the same; and which prevent the hub from predicting the user pseudonyms; Some of these properties can be achieved based on an integration of the previously mentioned techniques, e.g., PINs (so that the authentication at the IDP is related to the selected RP); for strong unlinkability (so that the hub never gets to see which pseudonyms to impersonate).

Chapter 6

Conclusions

Secure two-party computation (S2PC) is a beautiful concept, possibly counter-intuitive at first encounter. Its exploration enables a distinctive reasoning about privacy and security problems, namely by virtue of the underlying ideal/real simulation paradigm. This dissertation described a new protocol for S2PC with commitments, with innovative components that improve efficiency and applicability, and resting with a clear vision that further improvements are possible. While the main contributions of the dissertation are technical, S2PC has a wide potential applicability in the area of privacy, deserving a reflection on matters that go beyond mathematics, informatics and cryptography.

The ability to achieve S2PC brings about pertinent questions concerning *trust*. On one hand, S2PC enables to a great extent avoiding the need for trusted third parties, i.e., those that are usually used to mediate interactions between two parties, but which otherwise would not have a personal interest in the private information involved in such interaction. On the other hand, there are arguable reasons (including sociological ones) why one might, in certain contexts, prefer a setting where trusted parties exist, possibly even if not initially backed up by corresponding trustworthiness. For example, in certain social settings there may be a founded belief that trust itself may promote an increase of trustworthiness, by making people rise to the occasion, i.e., becoming trustworthy up to the level of trust placed on them. Naturally, the opposite may and does also occur — malicious and unaccountable abuse of trust settings, namely in informatics systems not properly secured and audited. Beliefs aside, if S2PC can remove the need for trusted parties, it can also allow a secure interaction involving parties that would otherwise not be trusted for some roles.

Limiting the applicability of S2PC to the avoidance of trusted third parties could lead to a narrow perspective, promoting an unduly emphasis of negative use cases. It would thus be short-sighted to simply present S2PC as tool related to "hiding" information and about getting rid of trusted parties. It is pertinent to state a more positive and constructive note: *S2PC is a tool with the potential to empower users, and society as a whole, to elevate privacy to a dimension that is not focused on "having something to hide" but is rather more aligned with "having something to protect" — individuality and diversity, the freedom to choose and select what to share and express, including the capability to define the correlations of private-information that are allowed to external parties.* S2PC promotes this empowerment by allowing each party in an interaction to have a fine-grained control of what about its private input (in a computational context) is leaked when interacting with another party toward an established functional goal (learning a function of the combined inputs).

In a world with increasing tension between the right for privacy and the need to ensure security for the masses, S2PC (and more generally secure computation) may enable a useful compromise between privacy and an utility of combining private information from different parties. For example, it may enable verification of needed predicates about personal attributes (e.g., fit within an age range, healthy), without requiring disclosure of specific semantic values (e.g., birth date, list of diseases), thus voiding some otherwise arguable justifications for privacy violation. However, as S2PC overcomes technical challenges and becomes more practical, the progress in its adoption in broader use-cases depends also, to a large extent, on the progress to be made in expanding awareness about privacy, at individual and collective levels, as a shared concept, as a respected right, and exercised in a responsible manner, considerate of its dynamic and contextual nature. The study of S2PC provides knowledge and skills that may dissipate sometimes-illusory contradictions between privacy and sharing of information. Yet, these should be complemented with a strategy of education for privacy, promoting critical thinking and attitudes about privacy ... but that is a matter for other dissertations.

Organization. The remaining sections in this chapter give concluding remarks about the goals and contributions of this dissertation. Section 6.1 comments on the designed S2PC-with-Coms protocol and some of its components (e.g., simulatable commitment schemes), and highlights the diversity of instantiation possibilities and the flexibility for tradeoffs and engineering decisions. Section 6.2 reflects on the need for further research toward privacy-preserving brokered identification, exemplifying that S2PC can act as privacy-enhancer,

particularly mitigating a vector of mass surveillance (linkability) that is present in the analyzed brokered identification systems. Section 6.3 mentions envisioned technical improvements toward more practical S2PC-with-Coms.

6.1 A protocol for S2PC-with-Coms

As a main goal, this dissertation presented a complete protocol for S2PC with commitments (S2PC-with-Coms), combining an outer layer of commitments with an inner layer of S2PC based on a cut-and-choose of garbled-circuits. The contributions herein are an increment to the state-of-the-art in an area that has developed throughout a long line of research for more than three decades with contributions from many researchers in cryptography.

The protocol design, attentive to simulatability requirements and with an intention of improving efficiency and applicability, required the combination of many components, e.g., non-interactive zero knowledge proofs (NIZKPs) and proofs of knowledge (NIZKPoKs), oblivious transfers, coin-flipping, and extractable-and-equivocable commitments and BitCom schemes with trapdoor, while leaving flexible the possibilities of instantiation, and interfacing with black-box components, such as garbled circuits, PRGs and CR-Hash functions.

In comparison with the original S2PC-with-BitComs protocol that introduced the forge-and-lose technique [Bra13], the revised description in this dissertation generalized several aspects, with corresponding challenges. It was now described within the UC framework (namely simulatable without rewinding) and minimizing the interaction between parties. As a result, certain NIZKPoKs became more complex and the protocol requires setup assumptions. The new description allows several cryptographic instantiations, namely based on integer-factorization cryptography (IFC) and discrete-log cryptography (DLC). All ZKPs and ZKPoKs were revised, now being described in a non-interactive setting, based on separate Ext-Coms and Equiv-Coms, all of which can for example be based on the same short common reference string. Several engineering decisions are left open to implementation phase. For example, there are different requirements across types of trusted setup (GCRS vs. GPKI) and types of cryptographic instantiation (IFC vs. DLC), several computational vs. communication tradeoffs associated with parameters and selection method of the cut-and-choose partition, namely in non-interactive vs. interactive settings. An actual implementation requires awareness of what security parameters to use and where to include NIZKPs and/or NIZKPoKs, where to

generalize BitComs to BitStringComs, and ensuring that interaction timing does not disclose whether or not the forge-and-lose path was used.

S2PC-with-Coms is an augmentation of simple S2PC of Boolean circuits, in the sense that both parties also learn random commitments of all circuit input and output bits, and each party receives the randomness associated with the commitments of her own bits (with said commitments and randomnesses not being computed by the garbled circuit). Within the overall construction, several components stand-out as innovative, e.g., the forge-and-lose technique (already subsequently used in other works), a 2-out-of-1 oblivious transfer at the level of BitComs for an IFC-based instantiation of S2PC, a new simulatable commitment scheme (using the expand-mask-hash approach) instantiable in a non-interactive setting and with asymptotic rate arbitrarily close to 1 in each phase, a revised version of a NIZKPoK of Blum integer trapdoor, the abstraction of connectors (connecting commitments to garbled wire keys) that are statistically verified within the cut-and-choose approach.

The combined description of both types of instantiation highlights diverse contrasts. The research herein did not involve an actual computational implementation, but includes a detailed estimation of communication complexity components, which may be helpful to support engineering decisions during an implementation. While most S2PC protocols in the literature use DLC to somehow support the oblivious transfers and to ensure consistency of circuit input bits used within the cut-and-choose, this dissertation also considers the use of Blum integers, which allows different tradeoffs. For example, the trapdoor of (IFC-based) GM BitComs allows the simulator to directly extract bits and randomness from any BitCom, but the trapdoor of (DLC-based) ElGamal BitComs only allows direct extraction of bits, not the commitment randomness (this would require computing a discrete log). This means that the DLC instantiation requires several NIZKPoKs of discrete log, whereas in IFC a single NIZKPoK of Blum integer trapdoor is sufficient in a PKI setup. While extraction of committed bits would be enough in a simple S2PC (to let the simulator learn the input to use in the ideal world), such extraction is not enough for S2PC-with-Coms, where a real protocol needs to emulate a randomization of commitments. As a result, even though the described IFC instantiation requires more communication per commitment, there are components of the protocol where DLC primitives nonetheless incur a larger cost (e.g., in number of exponentiations).

Optimal tradeoffs remain to be decided depending on application contexts and implementation choices. For example, in regard to input bits of P_B (the circuit evaluator), the use of a

2-out-of-1 OT in an IFC instantiation makes it trivial to link sequential executions, without requiring more exponentiations. Conversely, in a DLC instantiation the straightforward linkage would either require more exponentiations (the second message of a two-step 1-out-of-2 OT) or an initial interaction associated with OT extension.

Achieving simulatability of random commitments turned out to contribute with a significant communication complexity to the overall protocol. Particularly, the decision of random outer-Coms permutations was achieved via a simulatable two-party coin-flipping, which requires an Ext-and-Equiv commitment. For a single-time S2PC without interest in commitments, several communication components can be ignored. For example, using elliptic curve cryptography, an estimated communication complexity for a S2PC of AES-128 circuit (without outer commitments) is about 2MB, whereas it is 3MB for S2PC-with-Coms.

Forge-and-lose. Since its inception, the forge-and-lose approach, i.e., an evaluation path allowing P_B to recover the input of a malicious P_A without further interaction upon evaluation of garbled circuits, has already been utilized with variations in different works (§3.6.2). For example: it was meanwhile used in a minimal-round protocol for S2PC [AMPR14], there with a construction based on DLC; another work [FJN14a] used a construction that does not rely on number-theoretic assumptions, namely without trapdoor commitments, but requiring internal changes to the underlying Boolean circuit. The original forge-and-lose technique was described based on Blum BitComs (which were also used to enable the 2-out-of-1 OT), but the description herein uses more generic Ext-BitComs and Equiv-Coms, thus also becoming instantiable in DLC based on Pedersen BitComs. It is foreseeable that the forge-and-lose approach (i.e., including variations to the described forge-and-lose technique) will continue finding applicability within protocols based on cut-and-choose. Specifically, the approach very generally allows an improvement of statistical security of cut-and-chooses implemented in contexts where selective failure attacks would otherwise require a majority of correct instances selected for evaluation.

Coin-flipping. Coin-flipping is one of the sub-protocols required in S2PC-with-Coms, to enforce the randomization of commitments. The lack of communication efficiency of older UC commitment schemes (a fundamental primitive for UC coin-flipping) motivated research toward more efficient coin-flipping. This lead to the development of a new UC commitment scheme, simultaneously extractable and equivocable, with both phases having asymptotic

rate arbitrarily close to 1 and being instantiable as non-interactive (i.e., requiring a single message) based on a non-programmable random oracle. Prior and contemporary constructions with similar communication rate require an initial interactive phase of oblivious transfers [GIKW14, DDGN14, CDD+15, FJNT16, CDD+16].

6.2 Toward privacy-preserving brokered identification

The analysis of the FCCX and GOV.UK Verify brokered identification systems evidenced severe privacy and security problems, raising concerns about mass surveillance and overreach to private information. In the analyzed systems, the hub is able to profile all users in respect to their authentications across different service providers. If compromised, the hub can even actively impersonate users to gain access to their accounts (and the associated private data) at service providers. This represents a serious danger to the privacy of citizens and, more generally, to civil liberties. The described vulnerabilities are exploitable and could lead to undetected mass surveillance, in sharp contrast with the views of the research community [IAC14] whose scientific advances enable privacy-preserving and secure solutions.

Based on the research findings [BCDA15], it is clear that in the interest of privacy and security the systems should be adjusted to address plausible threats and achieve resilience against a compromised hub. This dissertation discussed a solution based on S2PC-with-Coms, to address a main problem of linkability across authentications. Besides promoting privacy-preserving solutions for a real contemporary system, the application of S2PC as a privacy enhancer also motivates further research in the area.

A comprehensive solution for brokered identification still requires greater formalization. One would need a design specification and proper requirements, followed by a fully specified and unambiguous protocol, accompanied by a proof of security. As a first step, a formal framework for brokered authentication could perhaps be devised based on the ideal/real simulation paradigm. The framework would integrate all the security, privacy, auditability and forensic properties at stake, while considering an adversarial model in which any party, including the hub, may be compromised and/or collude with other parties.

The research did not only consider exposing privacy problems, but also the challenge of proposing a privacy-preserving alternative. The systems under analysis are subject to certain (arguable) design constraints that preclude certain cryptographic solutions (existing

in the literature) that would otherwise fit as a solution for privacy-preserving authentication. Nonetheless, the S2PC-based solution described herein shows that those design constraints should not be an excuse to forfeit desirable security properties, such as unlinkability by a central authority. Conceptually, once understanding that secure computation enables the realization of a counter-intuitive functionality (computing over inputs distributed across several parties, without sharing them), it becomes clear that certain operational goals (e.g., mediation of authentication transactions) do not need to overshadow privacy goals (e.g., unlinkability of same-user transactions across different service providers).

The clearer reasoning made possible by the ideal/real simulation paradigm also reveals the fallacy of the structural design of the analyzed systems: in order to avoid a problematic linkability by identity providers, the introduced online central entity gains an even wider linkability power. It is as if the ideal/real design paradigm was used in a thwarted manner, simply projecting into the real world the trusted party of the ideal world. This reasoning leads to a very pertinent question: *if the design of S2PC protocols is about getting rid of the "trusted party" artifact conceptualized for a protocol in an ideal world, then why should the hub even exist in the real world?* An attempt to answer the above question induces an explicit reasoning about the role of the hub. Besides enabling a mediation between identity providers and service providers, and users, while enforcing a mutual blinding between the two types of providers, *is there any additional role for the hub?* Perhaps it may be intended as facilitator of certain auditability and/or certain selective forensic actions. Even if some aspects of this role may be arguable, it is extremely important to make those intentions explicit and public, to allow a design and review that considers an informed balance between privacy, security, auditability and forensics. In this regard, the research presented herein exposes that the inferred protocols are exploitable to allow undetected and unaccountable surveillance of all users.

If the role of the hub is not to allow mass surveillance (and based on NSTIC and IDAP it is here assumed that the brokered identification projects should indeed be about providing a privacy-preserving solution to citizens, as well as be transparent about auditability and forensic capabilities), then its role should be made more explicit. Several possibilities of selective forensic analysis and auditability have been very briefly mentioned in this dissertation, in a context differentiated from mass surveillance. For example, the hub could be made able to perform forensic investigations, selectively per user and/or per the user accesses at particular service providers, but only when aided by other entities (e.g., identity providers and/or service providers) that may be independently incentivized to prevent abuse by a

compromised hub. At the same time, in regard to logged authentication transactions, the actions of identity providers and service providers may be made auditable. It is here suggested that envisioned use cases of auditability and forensic analysis should be formalized within the ideal/real paradigm. Particularly, if the hub has a place in the real world, being susceptible to adversarial corruption, then it ought to also have a corresponding role in the ideal world, but not as the ideal world "trusted party" (which by definition never gets compromised).

After the publication of the mentioned research results [BCDA15], a governmental institute in the US has published a draft white-paper [GLM15] calling out for development of more secure and privacy enhancing solutions that may address some, but not all, of the identified problems. A followup "public comment" [BCD16] has been submitted, based on the initial research [BCDA15], as a contribute to the revision of the white paper, in order to promote a more thorough and integrated solution to the identified problems.

Besides needed technical developments with respect to privacy and other properties, the expanding immersion of brokered identification in society also motivates a thorougher reflection about matters of *identity* vs. identification and authentication. For example, *when should an identity attribute be forced a valuation for the purpose of identification?* Of particular significance is the role of "identity providers" — which could, perhaps more accurately, be called "identification/authentication facilitators". While they may help users identify and authenticate to certain service providers, an excessive reliance on them may foreseeably have a degrading effect on the value of identity self-assertions. *In an extreme case where authentication would become mandatory and exclusive via brokered identification systems, could the* presence *of an individual become insufficient to* prove *its* existence*?* This too is a matter for other dissertations.

6.3 Further research on S2PC-with-Coms

6.3.1 Technical improvements

Overall, the described S2PC-with-Coms protocol leaves open a considerable number of opportunities to pursue complexity improvements, which deserve further analysis. Several technical improvements do not require changing the protocol structure and can focus on individual components; others may be derived from different approaches.

1. Better NIZKPoKs. The NIZKPoK of Blum integer trapdoor described in a non-interactive and non-rewinding setting, based on a short common reference string reusable by all other NIZKPs and NIZKPoKs in the S2PC-with-Coms protocol, requires communication of group elements in number linear with the statistical parameter, and computation of a linear number of extractable commitments and a unitary number (two) of equivocable commitments and non-programmable random oracle (NPRO) calls. Similarly, the NIZKPoK of DL requires communication of a linear number of group elements, which is much more expensive than the traditional Schnorr protocol possible in an interactive setting and if rewinding is allowed. It is thus natural to pursue solutions that would allow the needed NIZKPoKs with reduced communication, e.g., up to a unitary number of group elements, without significantly increasing computation.

2. Better extractability of discrete log (DL). In regard to the outer-Coms, the protocol requires extractability not only of the committed bits but also of the "randomness" used to produce the commitments. This is easy in the IFC instantiation, where a single ZKPoK of trapdoor enables the simulator to extract square-roots (i.e., the randomness used in BitComs) from subsequent BitComs. However, in DLC instantiations the simulator is not able to extract discrete logs, even if it knows the trapdoor of ElGamal commitments. For this reason, an added NIZKPoK of DL is produced for each outer ElGamal commitment, and also for the coin-flipping of outer-Com permutations for wire sets of P_B. An avenue of improvement of the overall protocol efficiency would be to devise a new and efficient commitment scheme that directly allows extraction of discrete logs from a trapdoor, and which retains the useful homomorphic properties.

3. Better bit-string Ext-Coms in IFC. In some components of the S2PC-with-Coms protocol, the use of DLC allowed compacting a vector of BitComs into a single bit-string commitment of size equal to a single BitCom, reducing the communication complexity with respect to some connectors and to the coin-flipping of permutations, and reducing the size of the final state of each party. For IFC, while Blum Equiv-BitComs can be easily generalized to a compact bit-string Com with homomorphic properties (see §B.4.1.2), the described parsing of GM bit-string Ext-Coms was only logical, not compressing their overall size. An efficiency improvement can conceivably be obtained by also compacting them into a bit-string Ext-Com with additive properties. This case was not explored herein, but an IFC-based additively homomorphic encryption scheme can be used to produce additively homomorphic Ext-Coms.

An envisioned starting point is the Paillier encryption scheme [Pai99], which can be based on Blum integers and requires twice the size of the Blum integer to commit to a bit-string. As another example, there is the encryption scheme of Benhamouda et al. [JL13], with shorter ciphertext expansion for a single committed short bit-string (e.g., requiring a single group element of the size of the modulus to commit up to 128 bits), but requiring a modulus obtained as the products of two primes where one prime has remainder 1 upon division by a power of 2. Both examples are based on assumptions similar to the intractability of deciding the residuosity degree (e.g., quadratic or higher) of certain group elements.

4. Better oblivious transfers (OTs) for connectors of input of P_B. It would be interesting to find a way to reduce the communication associated with the connectors of input of P_B, with an impact similar to the one obtained for connectors of input of P_A, where the number of communicated group elements was reduced by using bit-string commitments instead of BitComs. This may perhaps be achievable based on a generalization of the several 2-out-of-1 OTs (in IFC) or the 1-out-of-2 OTs (in DLC) into a respective $2n$-out-of-n OT (in IFC) or n-out-of-$2n$ OT (in DLC), without incurring a linear increase in number of communicated group elements, and such that a single group-element (e.g., a commitment to n bits) would allow deriving the needed n garbled keys.

On a different perspective, the protocol can become computationally more efficient by directly improving the 1-out-of-2 OTs or 2-out-of-1 OTs. An obvious approach is to use OT extension techniques [KK13, KOS15, ALSZ15] for 1-out-of-2 OTs, extending a small base of less-efficient OTs into many very-efficient OTs. As a disadvantage, bootstrapping the OT extension requires an initial interactive phase to implement the initial OTs. Nonetheless, there are applications where the slight increase of number of communication rounds is not problematic. It is left as an open problem how to devise an OT-extension technique applicable to the IFC-based 2-out-of-1 OT (namely with number of exponentiations independent of the number of OTs). This would mean allowing one party, knowing square-roots of many squares, but now knowing the integer factorization of the modulus, to help another party, knowledgeable of the factorization, to compute all square-roots with a sub-linear number of exponentiations.

5. Reducing complexity of connectors of P_A. The technique described for connectors of input of P_A allows that, for each challenge instance, all permutation bits (i.e., across all input wire indices) are compacted into a single BitString Com. However, the overall

construction (based on a pseudo-homomorphism) is still dependent on the initial generation of BitComs (one per input bit of P_A), to handle the conversion of an additive sum in a group of large characteristic into a sum (XOR) in a group of characteristic two and large order (bit-strings). The additional BitComs could be avoided if the bit-string Com was directly XOR-homomorphic. Recent UC Com schemes can be instantiated as homomorphic in an extension of a field of characteristic two, thus being potentially applicable [CDD+16], even though requiring a setup of oblivious transfers. However, it is less clear how to make those commitments more meaningful in a multi-party execution, where the PKI-based commitments from one party should be meaningful for other parties not directly involved in a S2PC, i.e., which did not participate in the setup OT of the respective Coms. Since they are based on an initial base of oblivious transfers, they also require a setup interaction. Thus, an interesting avenue of research is developing more efficient non-interactive commitments allowing homomorphic XOR of bit-strings.

6. More efficient non-interactive transformations. The described NIZKPs and NIZKPoKs (Appendix A) as well as the non-interactive transformation used in the Ext&Equiv-Com scheme (Appendix C) were all based on a NPRO transformation, whereby the prover decides a cut-and-choose challenge as a NPRO image of an equivocable commitment. For sub-protocols (e.g., a cut-and-choose of garbled circuits) built as a parallelization of binary challenges in number proportional to the statistical parameter, the respective transformation for the non-interactive setting requires increasing the number of binary challenges up to the number of bits of computational security (e.g., 96 bits for short-term protocol durations, 128 bits for longer term), even if the context of the overall protocol only requires a shorter statistical security parameter (e.g., 40 bits). It is here noticed that some of those NIZKPs and NIZKPoKs can actually take advantage of the interaction of the embracing S2PC-with-Coms protocol, to reduce complexity without increasing the overall number of rounds. This can be done with a technique where the challenges are previously committed via an oblivious transfer, as can be used for the cut-and-choose of the SP2C (see §2.4.2, describing the suggestion from [AMPR14]). The sequence of binary challenges of NIZKPs and NIZKPoKs produced by P_A can be committed as part of the initial message of P_B, letting P_A then reply with two answers such that P_B only receives the one selected via the OT. The same applies to the NIZKPs and NIZKPoKs in the second message of P_B, if P_A also initiates respective OTs in her previous message. However, the technique is not directly applicable to the NIZKPs and NIZKPoKs

contained in the first message of P_B. Still, it is conceivable that an improvement is possible based on a PKI setting, taking advantage of the public parameter of P_A already possessed by P_B. Also, when considering a multi-execution setting, where a pair of parties intends to perform many S2PC executions, an initial preparation of OTs can be used for subsequent NIZKPs and NIZKPoKs, i.e., useful even for the first message of P_B in each subsequent execution. Ignoring the cost of the additional OTs, this may represent a communication reduction of a factor between about 2.4 and 3.2 for the cost of NIZKPs and NIZKPoKs, when comparing 40 bits of statistical security vs. 96 bits (short term) or 128 bits (long term) of computational security, and a computational increase due to the additional erasure encoding and decoding.

7. Forge-and-lose and/or linkage at different levels. The forge-and-lose technique allowed a significant reduction of number of instances within a cut-and-choose. The original technique was developed at the level of garbled circuits. Conceptually, it is intuitive that this may be applied at different levels, internal to a circuit. As suggested in the original paper, an integration with the Lego approach may be promising, as there the cut-and-choose approach already enables a better statistical security for the same number of gates. As a different example, in a multi-execution setting, where the same circuit needs to be executed multiple times, the cut-and-choose can be applied across many circuits in advance, and thus the requirements for each execution in separate take advantage of a better statistical security [HKK+14, LR14]. Considering the linkage application suggested by the availability of outer Coms or BitComs of a S2PC-with-Coms, a different related approach that seems to be very promising is the case of linking together several identical sub-components of a circuit. This approach may be advantageous when considering a large circuit that can be described as a sequential iteration of very similar sub-circuits. For example, this may be applicable to block-ciphers and hash functions that are designed as a multi-round execution of a smaller circuit. The idea is to start by building a slightly larger sub-circuit that accounts for all differences between sub-circuits [KKW16], i.e., that can replace any sub-circuit, in order to be in a setting where each S2PC is indeed a sequence of many equal sub-circuits. Then, by linking them all together, with appropriate connectors, the initial cut-and-choose statistics will be based on the number of sub-circuits, and the likelihood that across different circuits always the same sub-circuit is incorrect. Given the large numbers of sub-circuits, a better statistical security can then be achieved with fewer circuits. This would also have to account for the cost of connectors between each pair of sequential sub-circuits, but at

least asymptotically (with increasing size of sub-circuits for the same number of input and output bits) it would yield better results. Taking this reasoning to the limit, the linkage could happen at the level of each individual gate, thus enabling a maximum number of (very small) instances in the cut-and-choose. Here the cost of connectors would become the bottleneck, but further improvements in instantiations of connectors may make this a feasible approach.

6.3.2 Applicability considerations

1. Linkage through commitments. S2PC-with-Coms provides an easy way to link several executions (§3.5.5). Since the randomness of outer Coms is never revealed, the same outer-Coms can be used directly in subsequent execution. The linkage occurs at a layer abstract to the garbled circuits, and its support on public-key commitments (e.g., based on a global CRS or PKI, where everyone knows the public parameters of every party) makes it applicable to multi-party settings. Conversely, the output wires of check circuits are not directly reusable across different executions. It is nonetheless conceivable that linkage may be achievable based on symmetric-key cryptographic primitives, possibly leveraging the garbled values (wire keys) used in garbled circuits. Since the protocol described in this dissertation defines connectors as abstract structures that connect public-key commitments to garbled keys and vice-versa, it is likely that there are possible synergies between linkage at the level of commitments and linkage at the level of wire keys. A recent formalization of reactive linkage at the level of garbled circuits appears in [NR16]. It is certainly worth exploring concrete applications where linkage of several small circuits may provide a benefit in comparison with a single evaluation of a much larger circuit (e.g., for S2PC of an optimal decision tree from a database partitioned across two parties [LP02]). It is likely that more efficient additively-homomorphic commitments may enable more efficient constructions, namely protocols that directly enable bit-wise XOR homomorphism for bit-strings, such as the scheme devised by Cascudo et al. [CDD+16] (in spite of its initial interactive phase for bootstrapping OTs).

2. Two-output with fairness. The ideal functionality described herein did not consider *fairness*, whereby either both parties would learn their respective circuit output or both parties would learn nothing about it. Interestingly, the described IFC instantiation is already explicitly based on Blum integers, which are naturally suited to known fairness solutions based on gradual release [GMPY06] (inherently interactive, after the whole evaluation of

garbled circuits). Essentially, the protocol would be augmented to have the circuit output bits of each party be XORed with a private random bit-mask of the other party. Each of the mask bits is then committed in a particular way, with an auxiliary Blum integer trapdoor known by the other party. Finally, the parties play a gradual release protocol for the trapdoors of the auxiliary Blum integers (one for each party), which once learned allows each party to automatically decrypt all circuit output bits. The *fairness* extension is trivial as described for the circuit outputs, but would be more intricate to implement if designed to also encompass the outer-Coms and respective openings. More efficient solutions are possible in the presence of a trusted arbitrer [KM16], which only needs to intervene in case one of the regular parties performs early abort.

3. Multiple executions. The estimation of communication complexity in this dissertation considered cut-and-choose configurations for an individual execution. However, in a multi-execution setting, where the same Boolean circuit is executed many times in several S2PCs, it is possible to benefit from statistical improvements of the cut-and-choose [HKK+14, LR14]. Essentially, this is due to the ability of making a single cut-and-choose with many more circuits, checking some of them for correctness and then dividing the remaining ones into several smaller subsets of evaluation circuits, one for each subsequent S2PC execution. As a result, the same statistical security can be obtained with fewer evaluation circuits per each execution. An interesting example of a multiple execution setting is that of nation-scale privacy-preserving brokered identification. For example, the initial solicitation of the FCCX brokered identification [Uni13] system requested that an actual implementation should be able to sustain up to one million transactions per hour. If supporting each authentication transaction with a S2PC of a block-cipher (Section 5.3) or some envisioned S2PC-based variation thereof [BCDA15], then the efficiency of each execution could greatly benefit from the statistical optimization related to the multi-execution setting. For example, the actual generation of millions of circuits, and respective check of a proportion of them, could take place several hours in advance from the actual subsequent evaluations, then allowing 40 bits of statistical security with just two evaluation garbled circuits of communication for each authentication transaction.

6.3.3 Definitional aspects

1. (Non-)transferability vs. transferability. A significant portion of the complexity associated with ZK sub-protocols is due to the techniques used to ensure non-transferability of proofs, a.k.a. deniability, which is also required by the ideal S2PC-with-Coms functionality. However, in some real applications transferability might be a non-problem, and might even be useful. For example, in the proposed application for brokered identification (Section 5.3), the outputted commitments are signed by both parties, exactly so that each party can later prove involvement of the other party. It is likely possible to obtain better efficiency when foregoing non-transferability, but a formal analysis requires changing the ideal functionality (e.g., see [CSV16] in the context of signatures and key-exchange). If non-transferability can be forfeited, then the local CRS used to bootstrap extractable commitments and equivocable commitments could be replaced by a CRS derived instead from a globally known string, e.g., broadcast by a trusted randomness beacon.

2. Bootstrapping the initial NIZKPoKs. It would be interesting to explore, for concrete applications, the simulatability consequences of using an initial ZKPoK simulatable with rewinding (in the plain model, i.e., without additional trusted setup) but not without rewinding, how to implement it in practice (e.g., how to prevent the "environment" from interacting with the parties) and under which circumstances it could possibly break non-transferability (but not other desirable properties). After bootstrapping the initial NIZKPoKs of trapdoor, the equivocability and extractability properties required by subsequent NIZKPs and other NIZKPoKs may take advantage of the global PKI setting. Equiv-Coms and Ext-Coms become simulatable (i.e., extractable or equivocable, respectively) based on the trapdoors, instead of requiring a local CRS or other setup. However, in a multi-party setting, the deniability would only be partial, e.g., a NIZKP built in this way would instead be perceived by a third party as the non-deniable logical disjunction (`or`) of the validity of the assertion `or` of the knowledge of the trapdoor. In other words, such NIZKP transcript could have only been produced either by a prover knowing a secret that validates the assertion, `or` by a malicious verifier knowing the trapdoor of the Equiv-Com scheme used in the proof transcript, but not by any third party oblivious to the trapdoor.

A clear efficiency improvement would also be possible in a simulatability setting with rewinding, e.g., evident when comparing Schnorr protocol [Sch91], which only requires a

constant number of exponentiations and communicated group elements to prove knowledge of a DL in a setting of simulation of rewinding, vs. the NIZKPoK of DL proposed in this dissertation (§A.3.3), which required a linear number of exponentiations and communicated group elements to enable extractability without rewinding. In other words, even if not devising a better NIZKPoK of DL without rewinding, efficiency may be improved if a certain application allows security based on simulation with rewinding.

3. Ideal vs. real commitments. The use of real commitment schemes in the ideal S2PC-with-Coms functionality (§2.2.5) was a design choice that enabled explicitly requiring random values for the outer commitments outputted by the parties. If the ideal functionality would instead embed ideal commitments, the real world would have a corresponding use of simulatable Com schemes, but the simulation game would not have to require inducing random commitment values (i.e., because now the ideal functionality does not compute actual commitment values). While this simplification of the ideal world could potentially enable better efficiency, it would give each party a certain control of the actual value of commitments. In a multi-party and multi-execution setting, this could potentially constitute a side-channel for transmission of information. Once again, a careful analysis of these aspects, in connection with an intended application, may clarify what changes of ideal functionality are deemed possible and/or appropriate, with corresponding possible efficiency improvements.

4. Adaptive adversaries. The adversarial model considered in this dissertation focused on static adversaries, which corrupt a party before the protocol execution. A natural extension to the work in this dissertation is to consider the adaptive setting and determine which changes may be necessary to ensure simulatability. Here, the adversary may decide which party(ies) to corrupt after observing part of the protocol execution.

References

[AF90] M. Abadi and J. Feigenbaum. *Secure circuit evaluation.* Journal of Cryptology, 2(1):1–12, 1990. DOI:10.1007/BF02252866. Previous version at the 5th Symp. Theoretical Aspects of Computer Science — STACS 88, vol. 294 of LNCS. DOI:10.1007/BFb0035850. (Cited on page 7.)

[ALSZ15] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. *More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries.* In E. Oswald and M. Fischlin (eds.), Advances in Cryptology — EUROCRYPT 2015, vol. 9056 of LNCS, pages 673–701. Springer Berlin Heidelberg, 2015. DOI:10.1007/978-3-662-46800-5_26. Also at eprint:ia.cr/2015/061 and eprint:ia.cr/2016/602. (Cited on pages 151 and 206.)

[AMPR14] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. *Non-Interactive Secure Computation Based on Cut-and-Choose.* In P. Nguyen and E. Oswald (eds.), Advances in Cryptology — EUROCRYPT 2014, vol. 8441 of LNCS, pages 387–404. Springer Berlin Heidelberg, 2014. DOI:10.1007/978-3-642-55220-5_22. (Cited on pages 73, 75, 144, 146, 147, 201, and 207.)

[ARS+15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. *Ciphers for MPC and FHE.* In E. Oswald and M. Fischlin (eds.), Advances in Cryptology — EUROCRYPT 2015, vol. 9056 of LNCS, pages 430–454. Springer Berlin Heidelberg, 2015. DOI:10.1007/978-3-662-46800-5_17. (Cited on pages 132 and 187.)

[Bar16] E. Barker. *NIST Special Publication 800-57 Part 1 Revision 4 — Recommendation for Key Management (Part 1: General).* National Institute of Standards and Technology, U.S. Department of Commerce, January 2016. DOI:10.6028/NIST.SP.800-57pt1r4. (Cited on pages 20, 30, and 122.)

[BB12] L. T. A. N. Brandão and A. N. Bessani. *On the reliability and availability of replicated and rejuvenating systems under stealth attacks and intrusions.* Journal of the Brazilian Computer Society, 18:61–80, 2012. DOI:10.1007/s13173-012-0062-x. (Cited on page 2.)

[BC87] G. Brassard and C. Crepeau. *Zero-Knowledge Simulation of Boolean Circuits.* In A. M. Odlyzko (ed.), Advances in Cryptology — CRYPTO 1986, vol. 263 of LNCS, pages 223–233. Springer, Heidelberg, 1987. DOI:10.1007/3-540-47721-7_16. (Cited on page 230.)

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. *Minimum Disclosure Proofs of Knowledge.* Journal of Computer and System Sciences, 37(2):156–189, 1988. DOI:10.1016/0022-0000(88)90005-0. (Cited on pages 54, 67, and 230.)

[BCC+15] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooij, T. Lange, R. Niederhagen, and C. van Vredendaal. *How to Manipulate Curve Standards: A White Paper for the Black Hat* `http://bada55.cr.yp.to`. In L. Chen and S. Matsuo (eds.), Security

Standardisation Research, vol. 9497 of LNCS, pages 109–139. Springer International Publishing, 2015. DOI:10.1007/978-3-319-27152-1_6. (Cited on page 39.)

[BCD16] L. T. A. N. Brandão, N. Christin, and G. Danezis. *A Public Comment on NCCoE's White Paper on Privacy-Enhancing Identity Brokers*. arXiv:1611:02968, 2015/2016. (Cited on page 204.)

[BCDA15] L. T. A. N. Brandão, N. Christin, G. Danezis, and Anonymous. *Toward Mending Two Nation-Scale Brokered Identification Systems*. In Proc. Privacy Enhancing Technologies, vol. 2015 of PETS, pages 135–155. De Gruyter Open, June 2015. DOI:10.1515/popets-2015-0022. (Cited on pages iii, 7, 15, 173, 178, 182, 183, 184, 186, 188, 189, 193, 202, 204, and 210.)

[BCPV13] O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. *Analysis and Improvement of Lindell's UC-Secure Commitment Schemes*. In M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini (eds.), Applied Cryptography and Network Security — 2013, vol. 7954 of LNCS, pages 534–551. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-38980-1_34. Also at eprint:ia.cr/2013/123. (Cited on pages 152 and 154.)

[BD90] J. Boyar and I. Damgård. *A discrete logarithm blob for noninteractive XOR gates*. DAIMI Report Series, 19(327), 1990. DOI:10.7146/dpb.v19i327.6717. (Cited on page 65.)

[BD11] P. Beynon-Davies. *The UK national identity card*. Journal of Information Technology Teaching Cases, 1(1):12–21, 03 2011. DOI:10.1057/jittc.2011.3. (Cited on page 182.)

[BDP00] J. Boyar, I. Damgård, and R. Peralta. *Short Non-Interactive Cryptographic Proofs*. Journal of Cryptology, 13(4):449–472, 2000. DOI:10.1007/s001450010011. (Cited on page 143.)

[Bea96a] D. Beaver. *Adaptive Zero Knowledge and Computational Equivocation (Extended Abstract)*. In Proc. 28th Annual ACM Symposium on Theory of Computing — STOC 1996, pages 629–638. ACM, New York, 1996. DOI:10.1145/237814.238014. (Cited on pages 12 and 43.)

[Bea96b] D. Beaver. *Correlated pseudorandomness and the complexity of private computations*. In Proc. 28th Annual ACM Symposium on Theory of Computing — STOC 1996, pages 479–488. ACM, New York, 1996. DOI:10.1145/237814.237996. (Cited on page 69.)

[Bea96c] D. Beaver. *Equivocable Oblivious Transfer*. In U. Maurer (ed.), Advances in Cryptology — EUROCRYPT 1996, vol. 1070 of LNCS, pages 119–130. Springer Berlin Heidelberg, 1996. DOI:10.1007/3-540-68339-9_11. (Cited on pages 44 and 67.)

[Bel02] Bellare. *A Note on Negligible Functions*. Journal of Cryptology, 15(4):271–284, 2002. DOI:10.1007/s00145-002-0116-x. (Cited on page 20.)

[Ber06] D. J. Bernstein. *Curve25519: New Diffie-Hellman Speed Records*. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin (eds.), Public-Key Cryptography — PKC 2006, vol. 3958 of LNCS, pages 207–228. Springer Berlin Heidelberg, 2006. DOI:10.1007/11745853_14. (Cited on page 29.)

[BFM88] M. Blum, P. Feldman, and S. Micali. *Non-interactive Zero-knowledge and Its Applications*. In Proc. 20th Annual ACM Symposium on Theory of Computing — STOC 1988, pages 103–112. ACM, New York, 1988. DOI:10.1145/62212.62222. (Cited on pages 33 and 233.)

[BG93] M. Bellare and O. Goldreich. *On Defining Proofs of Knowledge*. In E. F. Brickell (ed.), Advances in Cryptology — CRYPTO 1992, vol. 740 of LNCS, pages 390–420. Springer, Heidelberg, 1993. DOI:10.1007/3-540-48071-4_28. (Cited on page 231.)

[BGI16] E. Boyle, N. Gilboa, and Y. Ishai. *Breaking the Circuit Size Barrier for Secure Computation Under DDH*. In M. Robshaw and J. Katz (eds.), Advances in Cryptology — CRYPTO 2016, Part I, vol. 9814 of LNCS, pages 509–539. Springer Berlin Heidelberg, 2016. DOI:10.1007/978-3-662-53018-4_19. Also at eprint:ia.cr/2016/585. (Cited on page 77.)

[BHR12] M. Bellare, V. T. Hoang, and P. Rogaway. *Foundations of garbled circuits*. In Proc. 19th ACM Conference on Computer and Communications Security — CCS 2012, pages 784–796. ACM, New York, 2012. DOI:10.1145/2382196.2382279. Also at eprint:ia.cr/2012/265. (Cited on pages 68 and 71.)

[BK15] E. Barker and J. Kelsey. *NIST Special Publication 800-90A Revision 1 — Recommen-*

dation for Random Number Generation Using Deterministic Random Bit Generators. National Institute of Standards and Technology, U.S. Department of Commerce, June 2015. DOI:10.6028/NIST.SP.800-90Ar1. (Cited on pages 13 and 336.)

[Blu81] M. Blum. *Coin flipping by telephone*. A Report on CRYPTO 81, pages 11–15, 1981. Also at COMPCON'82, pp. 133-137, IEEE, 1982. Also as "Coin flipping by telephone – a protocol for solving impossible problems." at SIGACT News, 15:23–27, 1983, DOI:10.1145/1008908.1008911. (Cited on pages 11, 24, 39, 54, 62, 63, and 66.)

[BM90] M. Bellare and S. Micali. *Non-Interactive Oblivious Transfer and Applications*. In G. Brassard (ed.), Advances in Cryptology — CRYPTO 1989, vol. 435 of LNCS, pages 547–557. Springer New York, 1990. DOI:10.1007/0-387-34805-0_48. (Cited on page 82.)

[BMP13] J. Boyar, P. Matthews, and R. Peralta. *Logic Minimization Techniques with Applications to Cryptology*. Journal of Cryptology, 26(2):280–312, 2013. DOI:10.1007/s00145-012-9124-7. Also at eprint:ia.cr/2012/215. (Cited on page 129.)

[BMR90] D. Beaver, S. Micali, and P. Rogaway. *The round complexity of secure protocols*. In Proc. 22nd Annual ACM Symposium on Theory of Computing — STOC 1990, pages 503–513. ACM, New York, 1990. DOI:10.1145/100216.100287. (Cited on page 68.)

[BPW12] D. Bernhard, O. Pereira, and B. Warinschi. *How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios*. In X. Wang and K. Sako (eds.), Advances in Cryptology — ASIACRYPT 2012, vol. 7658 of LNCS, pages 626–643. Springer, Heidelberg, 2012. DOI:10.1007/978-3-642-34961-4_38. (Cited on page 233.)

[BR93] M. Bellare and P. Rogaway. *Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols*. In Proc. 1st ACM Conference on Computer and Communications Security — CCS 1993, pages 62–73. ACM, New York, 1993. DOI:10.1145/168588.168596. (Cited on pages 22 and 233.)

[BR95] M. Bellare and P. Rogaway. *Optimal asymmetric encryption*. In A. De Santis (ed.), Advances in Cryptology — EUROCRYPT 1994, vol. 950 of LNCS, pages 92–111. Springer, Heidelberg, 1995. DOI:10.1007/BFb0053428. (Cited on page 339.)

[Bra06] L. T. A. N. Brandão. *A Framework for Interactive Argument Systems using Quasigroupic Homorphic Commitment*. IACR Cryptology ePrint Archive, Report 2006/472, 2006. (Cited on page 143.)

[Bra13] L. T. A. N. Brandão. *Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique*. In K. Sako and P. Sarkar (eds.), Advances in Cryptology — ASIACRYPT 2013, Part II, vol. 8270 of LNCS, pages 441–463. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-42045-0_23. Also at eprint:ia.cr/2013/577. (Cited on pages iii, 6, 8, 11, 12, 55, 78, 83, 96, 108, 121, 122, 133, 144, 146, 147, 164, and 199.)

[Bra16] L. T. A. N. Brandão. *Very-efficient simulatable flipping of many coins into a well (and a New Universally-Composable Commitment Scheme)*. In C.-M. Cheng, B.-Y. Yang, and K.-M. Chung (eds.), Public-Key Cryptography — PKC 2016, Part II, vol. 9615 of LNCS, pages 297–326. Springer, Heidelberg, 2016. DOI:10.1007/978-3-662-49387-8_12. Also at eprint:ia.cr/2015/640. (Cited on pages iii, 6, 13, 14, 144, 149, 162, 169, and 171.)

[Bri13] Bristol Cryptography Group. *Circuits of Basic Functions Suitable For MPC and FHE*. TXT files at `cs.bris.ac.uk` website: "AES-non-expanded.txt" SHA256: `0260ae86ddd882cb 44c86b6ef553056f 6793a0dec30ab504 a89a9555a9ea8d00`, "sha-256.txt" SHA256: `0741a0f58ff376d6 10f6cb4fc8aecb74 0437bd5bef07fc13 9eea7114ecba6976`, Accessed June 2013. (Cited on pages 128, 129, and 187.)

[BRSS10] J. Black, P. Rogaway, T. Shrimpton, and M. Stam. *An Analysis of the Block-cipher-Based Hash Functions from PGV*. Journal of Cryptology, 23(4):519–545, 2010. DOI:10.1007/s00145-010-9071-0. Previous version at Crypto 2002, DOI:10.1007/3-540-45708-9_21. (Cited on page 190.)

[BSBC13] E. Barker, M. Smid, D. Branstad, and S. Chokhani. *NIST Special Publication 800-*

130 — A Framework for Designing Cryptographic Key Management Systems. National Institute of Standards and Technology, U.S. Department of Commerce, August 2013. DOI:10.6028/NIST.SP.800-130. (Cited on page 182.)

[Can00] R. Canetti. *Security and Composition of Multiparty Cryptographic Protocols.* Journal of Cryptology, 13(1):143–202, 2000. DOI:10.1007/s001459910006. Also at eprint:ia.cr/1998/018. (Cited on pages 3, 30, and 33.)

[Can01] R. Canetti. *Universally composable security: a new paradigm for cryptographic protocols.* In Proc. 42nd Annual Symposium on Foundations of Computer Science — FOCS 2001, pages 136–145, 2001. DOI:10.1109/SFCS.2001.959888. Also at eprint:ia.cr/2000/067. (Cited on pages 3, 30, and 33.)

[CCL15] R. Canetti, A. Cohen, and Y. Lindell. *A Simpler Variant of Universally Composable Security for Standard Multiparty Computation.* In R. Gennaro and M. Robshaw (eds.), Advances in Cryptology — CRYPTO 2015, vol. 9216 of LNCS, pages 3–22. Springer, Heidelberg, 2015. DOI:10.1007/978-3-662-48000-7_1. Also at eprint:ia.cr/2014/553. (Cited on pages 34 and 35.)

[CDD+15] I. Cascudo, I. Damgård, B. David, I. Giacomelli, J. Nielsen, and R. Trifiletti. *Additively Homomorphic UC Commitments with Optimal Amortized Overhead.* In J. Katz (ed.), Public-Key Cryptography — PKC 2015, vol. 9020 of LNCS, pages 495–515. Springer Berlin Heidelberg, 2015. DOI:10.1007/978-3-662-46447-2_22. Also at eprint:ia.cr/2014/829. (Cited on pages 14, 153, and 202.)

[CDD+16] I. Cascudo, I. Damgård, B. David, N. Döttling, and J. B. Nielsen. *Rate-1, Linear Time and Additively Homomorphic UC Commitments.* In M. Robshaw and J. Katz (eds.), Advances in Cryptology — CRYPTO 2016, Part III, vol. 9816 of LNCS, pages 179–207. Springer, Heidelberg, 2016. DOI:10.1007/978-3-662-53015-3_7. Also at eprint:ia.cr/2016/137. (Cited on pages 154, 171, 202, 207, and 209.)

[CDPW07] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. *Universally Composable Security with Global Setup.* In S. P. Vadhan (ed.), TCC 2007, vol. 4392 of LNCS, pages 61–85. Springer, Heidelberg, 2007. DOI:10.1007/978-3-540-70936-7_4. Also at eprint:ia.cr/2006/432. (Cited on page 36.)

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols.* In Y. Desmedt (ed.), Advances in Cryptology — CRYPTO 1994, vol. 839 of LNCS, pages 174–187. Springer Berlin Heidelberg, 1994. DOI:10.1007/3-540-48658-5_19. (Cited on pages 235 and 254.)

[CES12] CESG and NTAIA and Cabinet Office. *Good Practice Guide No. 43 — Requirements for Secure Delivery of Online Public Services*, December 2012. PDF file (46 pages) at gov.uk website. SHA256: b765484870bac4e5 d3ecf12a6cc72923 69870bbac41b48fb 55eb8163006706ab. (Cited on page 180.)

[CES14] CESG and NTAIA and Cabinet Office. *Good Practice Guide No. 45 — Identity Proofing and Verification of an Individual*, July 2014. PDF file (32 pages) at gov.uk website. SHA256: 99870cebd50ab893 ad5bb25a96f9b33c f58a19e2472d63c0 71794b104fd15b5b. (Cited on page 175.)

[CEvdGP87] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta. *Demonstrating Possession of a Discrete Logarithm Without Revealing it.* In A. Odlyzko (ed.), Advances in Cryptology — CRYPTO 1986, vol. 263 of LNCS, pages 200–212. Springer Berlin Heidelberg, 1987. DOI:10.1007/3-540-47721-7_14. (Cited on page 251.)

[CF01] R. Canetti and M. Fischlin. *Universally Composable Commitments.* In J. Kilian (ed.), Advances in Cryptology — CRYPTO 2001, vol. 2139 of LNCS, pages 19–40. Springer, Heidelberg, 2001. DOI:10.1007/3-540-44647-8_2. Also at eprint:ia.cr/2001/055. (Cited on pages 35, 39, 150, 151, and 152.)

[CFN90] D. Chaum, A. Fiat, and M. Naor. *Untraceable Electronic Cash.* In S. Goldwasser (ed.), Advances in Cryptology — CRYPTO 1988, vol. 403 of LNCS, pages 319–327. Springer New York, 1990. DOI:10.1007/0-387-34799-2_25. (Cited on page 94.)

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. *The Random Oracle Methodology, Revisited.* Journal of

the ACM, 51(4):557–594, July 2004. DOI:10.1145/1008731.1008734. Previous version at STOC 1998, DOI:10.1145/276698.276741. Also at eprint:ia.cr/1998/011. (Cited on pages 22 and 234.)

[CGT95] C. Crépeau, J. v. d. Graaf, and A. Tapp. *Committed Oblivious Transfer and Private Multi-Party Computation*. In D. Coppersmith (ed.), Advances in Cryptology — CRYPTO 1995, vol. 963 of LNCS, pages 110–123. Springer, Heidelberg, 1995. DOI:10.1007/3-540-44750-4_9. (Cited on page 69.)

[Cha87] D. Chaum. *Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How*. In A. M. Odlyzko (ed.), Advances in Cryptology — CRYPTO 1986, pages 195–199. Springer, Heidelberg, 1987. DOI:10.1007/3-540-47721-7_13. (Cited on page 230.)

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. *Universally composable two-party and multi-party secure computation*. In Proc. 34th Annual ACM Symposium on Theory of Computing — STOC 2002, pages 494–503. ACM, New York, 2002. DOI:10.1145/509907.509980. Also at eprint:ia.cr/2002/140. (Cited on pages 34, 39, 42, 44, and 152.)

[CP92] D. Chaum and T. Pedersen. *Wallet Databases with Observers*. In E. Brickell (ed.), Advances in Cryptology — CRYPTO 1992, vol. 740 of LNCS, pages 89–105. Springer Berlin Heidelberg, 1992. DOI:10.1007/3-540-48071-4_7. (Cited on pages 252, 253, 254, and 264.)

[CR03] R. Canetti and T. Rabin. *Universal Composition with Joint State*. In D. Boneh (ed.), Advances in Cryptology — CRYPTO 2003, vol. 2729 of LNCS, pages 265–281. Springer, Heidelberg, 2003. DOI:10.1007/978-3-540-45146-4_16. Also at eprint:ia.cr/2002/047. (Cited on pages 13, 34, 137, 141, and 151.)

[Cra96] R. Cramer. *Modular Design of Secure, yet Practical Cryptographic Protocols*. Ph.D. thesis, University of Amsterdam, 1996. (Cited on page 231.)

[Cré88] C. Crépeau. *Equivalence Between Two Flavours of Oblivious Transfers*. In Advances in Cryptology — CRYPTO 1987, LNCS, pages 350–354. Springer, Heidelberg, 1988. DOI:10.1007/3-540-48184-2_30. (Cited on page 78.)

[CSV16] R. Canetti, D. Shahaf, and M. Vald. *Universally Composable Authentication and Key-Exchange with Global PKI*. In C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang (eds.), Public-Key Cryptography — PKC 2016, Part II, vol. 9615 of LNCS, pages 265–296. Springer, Heidelberg, 2016. DOI:10.1007/978-3-662-49387-8_11. (Cited on page 211.)

[Cyb11] Cyber-Auth DG Committee — Canada. *Cyber Authentication Technology Solutions Interface Architecture and Specification Version 2.0: Deployment Profile*, March 25, 2011. PDF file (53 pages) at `kantarainitiative.org` website. SHA256: `2faf71b0d92d83e3 18208d6676743ad1 e624025fc14efdb3 9f19ec750d1cf6ed`. (Cited on page 176.)

[Dam88a] I. B. Damgård. *The application of claw free functions in cryptography — Unconditional Protection in Cryptographic Protocols*. Ph.D. thesis, Aarhus University, Mathematical Institute, 1988. DOI:10.7146/dpb.v17i269.7622. (Cited on page 54.)

[Dam88b] I. B. Damgård. *Collision Free Hash Functions and Public Key Signature Schemes*. In D. Chaum and W. L. Price (eds.), Advances in Cryptology — EUROCRYPT 1987, vol. 304 of LNCS, pages 203–216. Springer, Heidelberg, 1988. DOI:10.1007/3-540-39118-5_19. (Cited on page 150.)

[Dam00] I. Damgård. *Efficient Concurrent Zero-Knowledge in the Auxiliary String Model*. In B. Preneel (ed.), Advances in Cryptology — EUROCRYPT 2000, vol. 1807 of LNCS, pages 418–430. Springer Berlin Heidelberg, 2000. DOI:10.1007/3-540-45539-6_30. (Cited on pages 72, 232, 233, and 254.)

[Dam07] I. Damgård. *A "proof-reading" of Some Issues in Cryptography*. In L. Arge, C. Cachin, T. Jurdziński, and A. Tarlecki (eds.), ICALP 2007, vol. 4596 of LNCS, pages 2–11. Springer, Heidelberg, 2007. DOI:10.1007/978-3-540-73420-8_2. (Cited on page 22.)

[Dam10] I. Damgård. *On Σ-protocols*, March 2010. PDF file (22 pages) at `cs.au.dk` website. SHA256: `a02410f1b3aebcc1 5557d3497eeb4428 3add67c2c2f956f4 da1b5e9de627319a`. (Cited on page 231.)

[DC03] G. Di Crescenzo. *Equivocable and Extractable Commitment Schemes*. In S. Cimato, G. Persiano,

and C. Galdi (eds.), SCN 2002, vol. 2576 of LNCS, pages 74–87. Springer, Heidelberg, 2003. DOI:10.1007/3-540-36413-7_6. (Cited on pages 150 and 152.)

[DCIO98] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. *Non-interactive and Non-malleable Commitment*. In Proc. 30th Annual ACM Symposium on Theory of Computing — STOC 1998, pages 141–150. ACM, New York, 1998. DOI:10.1145/276698.276722. (Cited on pages 45 and 152.)

[DCKOS01] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. *Efficient and Non-interactive Non-malleable Commitment*. In B. Pfitzmann (ed.), Advances in Cryptology — EUROCRYPT 2001, vol. 2045 of LNCS, pages 40–59. Springer, Heidelberg, 2001. DOI:10.1007/3-540-44987-6_4. Also at eprint:ia.cr/2001/032. (Cited on pages 45 and 151.)

[DCO99] G. Di Crescenzo and R. Ostrovsky. *On Concurrent Zero-Knowledge with Pre-processing*. In M. Wiener (ed.), Advances in Cryptology — CRYPTO 1999, vol. 1666 of LNCS, pages 485–502. Springer, Heidelberg, 1999. DOI:10.1007/3-540-48405-1_31. (Cited on page 150.)

[DDGN14] I. Damgård, B. David, I. Giacomelli, and J. B. Nielsen. *Compact VSS and Efficient Homomorphic UC Commitments*. In P. Sarkar and T. Iwata (eds.), Advances in Cryptology — ASIACRYPT 2014, vol. 8874 of LNCS, pages 213–232. Springer, Heidelberg, 2014. DOI:10.1007/978-3-662-45608-8_12. Also at eprint:ia.cr/2014/370. (Cited on pages 14, 153, and 202.)

[DDN00] D. Dolev, C. Dwork, and M. Naor. *Nonmalleable Cryptography*. SIAM Journal on Computing, 30(2):391–437, 2000. DOI:10.1137/S0097539795291562. Previous version (*Non-Malleable Cryptography*) at STOC 1991, DOI:10.1145/103418.103474. (Cited on page 45.)

[DL09] I. Damgård and C. Lunemann. *Quantum-Secure Coin-Flipping and Applications*. In M. Matsui (ed.), Advances in Cryptology — ASIACRYPT 2009, vol. 5912 of LNCS, pages 52–69. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-10366-7_4. Also at arXiv:0903.3118. (Cited on page 152.)

[DLT14] I. Damgård, R. Lauritsen, and T. Toft. *An Empirical Study and Some Improvements of the MiniMac Protocol for Secure Computation*. In M. Abdalla and R. De Prisco (eds.), Security and Cryptography for Networks, vol. 8642 of LNCS, pages 398–415. Springer International Publishing, 2014. DOI:10.1007/978-3-319-10879-7_23. (Cited on page 187.)

[DN02] I. Damgård and J. B. Nielsen. *Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor*. In M. Yung (ed.), Advances in Cryptology — CRYPTO 2002, vol. 2442 of LNCS, pages 581–596. Springer, Heidelberg, 2002. DOI:10.1007/3-540-45708-9_37. Also at eprint:ia.cr/2001/091. (Cited on pages 39, 42, 150, 152, and 361.)

[DNO10] I. Damgård, J. B. Nielsen, and C. Orlandi. *On the Necessary and Sufficient Assumptions for UC Computation*. In D. Micciancio (ed.), TCC 2010, vol. 5978 of LNCS, pages 109–127. Springer, Heidelberg, 2010. DOI:10.1007/978-3-642-11799-2_8. Also at eprint:ia.cr/2009/247. (Cited on pages 151 and 154.)

[DO10] I. Damgård and C. Orlandi. *Multiparty Computation for Dishonest Majority: From Passive to Active Security at Low Cost*. In T. Rabin (ed.), Advances in Cryptology — CRYPTO 2010, vol. 6223 of LNCS, pages 558–576. SBH, 2010. DOI:10.1007/978-3-642-14623-7_30. Also at eprint:ia.cr/2010/318. (Cited on page 152.)

[DSDCO+01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. *Robust Non-interactive Zero Knowledge*. In J. Kilian (ed.), Advances in Cryptology — CRYPTO 2001, pages 566–598. Springer, Heidelberg, 2001. DOI:10.1007/3-540-44647-8_33. (Cited on page 233.)

[DSW08] Y. Dodis, V. Shoup, and S. Walfish. *Efficient Constructions of Composable Commitments and Zero-Knowledge Proofs*. In D. Wagner (ed.), Advances in Cryptology — CRYPTO 2008, vol. 5157 of LNCS, pages 515–535. Springer, Heidelberg, 2008. DOI:10.1007/978-3-540-85174-5_29. (Cited on page 152.)

[Dwo06] C. Dwork. *Differential Privacy*. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener

(eds.), ICALP 2006, Part II, vol. 4052 of LNCS, pages 1–12. Springer Berlin Heidelberg, 2006. DOI:10.1007/11787006_1. (Cited on page 7.)

[EGL85] S. Even, O. Goldreich, and A. Lempel. *A randomized protocol for signing contracts*. Communications of the ACM, 28(6):637–647, June 1985. DOI:10.1145/3812.3818. (Cited on page 78.)

[ElG85] T. ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In G. Blakley and D. Chaum (eds.), Advances in Cryptology — CRYPTO 1984, vol. 196 of LNCS, pages 10–18. Springer, Heidelberg, 1985. DOI:10.1007/3-540-39568-7_2. (Cited on pages 60 and 66.)

[Eur10] European Court of Human Rights. *European Convention on Human Rights*, Entry into force on June 2010. (As amended by Protocols No. 11 and 14; supplemented by Protocols Nos. 1,4,6,7 and 13.) PDF file (30 pages) at `echr.coe.int` website. SHA256: `4c65e157638088a7 a0905352fbed358a ca0f79564a8dfc06 d3246ee2cd7becfa`. (Cited on page 185.)

[FF09] M. Fischlin and R. Fischlin. *Efficient Non-malleable Commitment Schemes*. Journal of Cryptology, 22(4):530–571, 2009. DOI:10.1007/s00145-009-9045-2. Also at Crypto 2000, DOI:10.1007/3-540-44598-6_26. (Cited on pages 45 and 360.)

[FF11] Federal Chief Information Officers Council and Federal Enterprise Architecture. *Federal Identity, Credential, and Access Management (FICAM) Roadmap and Implementation Guidance — Version 2.0*, December 2, 2011. PDF file (478 pages) at `idmanagement.gov` website. SHA256: `f4b91795175b2a73 1f3b29c32d5e5fef eaa1cb97d0cec3c2 522f4fe8c51a5bdc`. (Cited on page 175.)

[FFS88] U. Feige, A. Fiat, and A. Shamir. *Zero-knowledge proofs of identity*. Journal of Cryptology, 1(2):77–94, 1988. DOI:10.1007/BF02351717. (Cited on pages 231 and 249.)

[FJN+13] T. Frederiksen, T. Jakobsen, J. Nielsen, P. Nordholt, and C. Orlandi. *MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions*. In T. Johansson and P. Nguyen (eds.), Advances in Cryptology — EUROCRYPT 2013, vol. 7881 of LNCS, pages 537–556. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-38348-9_32. Also at eprint:ia.cr/2013/155. (Cited on pages 76 and 148.)

[FJN14a] T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen. *Faster Maliciously Secure Two-Party Computation Using the GPU*. In M. Abdalla and R. De Prisco (eds.), SCN 2014, vol. 8642 of LNCS, pages 358–379. Springer International Publishing, 2014. DOI:10.1007/978-3-319-10879-7_21. Also at eprint:ia.cr/2014/270. (Cited on pages 95, 147, and 201.)

[FJN14b] T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen. *Faster Maliciously Secure Two-Party Computation Using the GPU*. In M. Abdalla and R. De Prisco (eds.), Security and Cryptography for Networks, vol. 8642 of LNCS, pages 358–379. Springer International Publishing, 2014. DOI:10.1007/978-3-319-10879-7_21. (Cited on page 187.)

[FJNT15] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, and R. Trifiletti. *TinyLEGO: An Interactive Garbling Scheme for Maliciously Secure Two-Party Computation*. IACR Cryptology ePrint Archive, Report 2015/309, 2015. (Cited on page 148.)

[FJNT16] T. Frederiksen, T. Jakobsen, J. Nielsen, and R. Trifiletti. *On the Complexity of Additively Homomorphic UC Commitments*. In E. Kushilevitz and T. Malkin (eds.), Theory of Cryptography, vol. 9562 of LNCS, pages 542–565. Springer Berlin Heidelberg, 2016. DOI:10.1007/978-3-662-49096-9_23. Also at eprint:ia.cr/2015/694. (Cited on pages 153 and 202.)

[FLM11] M. Fischlin, B. Libert, and M. Manulis. *Non-interactive and Re-usable Universally Composable String Commitments with Adaptive Security*. In D. Lee and X. Wang (eds.), Advances in Cryptology — ASIACRYPT 2011, vol. 7073 of LNCS, pages 468–485. Springer, Heidelberg, 2011. DOI:10.1007/978-3-642-25385-0_25. (Cited on page 152.)

[FLS90] U. Feige, D. Lapidot, and A. Shamir. *Multiple Non-interactive Zero Knowledge Proofs Based on a Single Random String*. In Proc. 31st Annual Symposium on Foundations of Computer Science — FOCS 1990, vol. 1, pages 308–317. IEEE, 1990. DOI:10.1109/FSCS.1990.89549.

(Cited on page 233.)

[FN13] T. K. Frederiksen and J. B. Nielsen. *Fast and Maliciously Secure Two-Party Computation Using the GPU*. In M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini (eds.), Applied Cryptography and Network Security — ACNS 2013, vol. 7954 of LNCS, pages 339–356. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-38980-1_21. (Cited on page 4.)

[Fre15] T. K. Frederiksen. *The Hitchhiker's Guide to Garbled Circuits*. Ph.D. thesis, Aarhus University, 2015. (Cited on page 147.)

[FS87] A. Fiat and A. Shamir. *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*. In A. M. Odlyzko (ed.), Advances in Cryptology — CRYPTO 1986, vol. 263 of LNCS, pages 186–194. Springer Berlin Heidelberg, 1987. DOI:10.1007/3-540-47721-7_12. (Cited on pages 22, 233, and 254.)

[FS90a] U. Feige and A. Shamir. *Witness Indistinguishable and Witness Hiding Protocols*. In Proc. 2nd Annual ACM Symposium on Theory of Computing — STOC 1990, pages 416–426. ACM, New York, 1990. DOI:10.1145/100216.100272. (Cited on page 231.)

[FS90b] U. Feige and A. Shamir. *Zero Knowledge Proofs of Knowledge in Two Rounds*. In G. Brassard (ed.), Advances in Cryptology — CRYPTO 1989, vol. 435 of LNCS, pages 526–544. Springer New York, 1990. DOI:10.1007/0-387-34805-0_46. (Cited on page 150.)

[Fuj14] E. Fujisaki. *All-But-Many Encryption*. In P. Sarkar and T. Iwata (eds.), Advances in Cryptology — ASIACRYPT 2014, vol. 8874 of LNCS, pages 426–447. Springer, Heidelberg, 2014. DOI:10.1007/978-3-662-45608-8_23. Also at eprint:ia.cr/2012/379. (Cited on page 152.)

[Gen09] C. Gentry. *Fully Homomorphic Encryption Using Ideal Lattices*. In Proc. 41st Annual ACM Symposium on Theory of Computing — STOC 2009, pages 169–178. ACM, New York, 2009. DOI:10.1145/1536414.1536440. (Cited on page 77.)

[Gen14] General Services Administration. *Solicitation Number QTA0014AWA3005 — Identity Services Support*. Federal Business Opportunities (FedBizOpps), Updated on August 19, 2014. ZIP files at `fbo.gov` website: "Amendment_1.zip" (Aug 06, 2014) SHA256: 08a6a9f4b59f9ce2 466c1a90049b2498 1385386aa7df0a4f 8b2db74c50848881; "Amendment_2.zip" (Aug 19, 2014) SHA256: 0c078fe1266a1f3b 717292331a158c27 822fbd9a35c2128e f28af567642671fb. (Cited on pages 177 and 184.)

[GGH+13] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs. *Optimizing ORAM and Using It Efficiently for Secure Computation*. In E. De Cristofaro and M. Wright (eds.), PETS 2013, pages 1–18. Springer Berlin Heidelberg, 2013. DOI:10.1007/978-3-642-39077-7_1. Also at eprint:ia.cr/2013/239. (Cited on page 77.)

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. *Quadratic Span Programs and Succinct NIZKs without PCPs*. In T. Johansson and P. Q. Nguyen (eds.), Advances in Cryptology — EUROCRYPT 2013, vol. 7881 of LNCS, pages 626–645. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-38348-9_37. Also at eprint:ia.cr/2012/215. (Cited on page 171.)

[GIKW14] J. A. Garay, Y. Ishai, R. Kumaresan, and H. Wee. *On the Complexity of UC Commitments*. In P. Q. Nguyen and E. Oswald (eds.), Advances in Cryptology — EUROCRYPT 2014, vol. 8441 of LNCS, pages 677–694. Springer, Heidelberg, 2014. DOI:10.1007/978-3-642-55220-5_37. Also at eprint:ia.cr/2015/694. (Cited on pages 14, 153, 154, 170, 202, 367, and 368.)

[GKP+13] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. *Reusable Garbled Circuits and Succinct Functional Encryption*. In Proc. 45th Annual ACM Symposium on Theory of Computing — STOC 2013, pages 555–564. ACM, New York, 2013. DOI:10.1145/2488608.2488678. (Cited on page 77.)

[GLM15] P. Grassi, N. Lefkovitz, and K. Mangold. *Privacy-Enhanced Identity Brokers — Building Block White Paper*, October 19, 2015. PDF file (27 pages) at `nccoe.nist.gov` website. SHA256: 7f366f7d66cd4d65 c9d4363e1ffccbba 68b585caf96ff553 455b9cb515e3ed1a. (Cited on page 204.)

[GM84] S. Goldwasser and S. Micali. *Probabilistic encryption*. Journal of Computer and System Sciences, 28(2):270–299, 1984. DOI:10.1016/0022-0000(84)90070-9. (Cited on pages 55, 58, and 66.)

[GMPY06] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. *Resource Fairness and Composability of*

Cryptographic Protocols. In TCC 2006, vol. 3876 of LNCS, pages 404–428. Springer, Heidelberg, 2006. DOI:10.1007/11681878_21. Also at eprint:ia.cr/2005/370. (Cited on page 209.)

[GMR84] S. Goldwasser, S. Micali, and R. L. Rivest. *A "Paradoxical" Solution To The Signature Problem*. In Proc. 25th Annual Symposium on Foundations of Computer Science — FOCS 1984, pages 441–448. IEEE Computer Society, 1984. DOI:10.1109/SFCS.1984.715946. (Cited on page 54.)

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. *The Knowledge Complexity of Interactive Proof-systems*. In Proc. 17th Annual ACM Symposium on Theory of Computing — STOC 1985, pages 291–304, New York, NY, USA, 1985. ACM. DOI:10.1145/22145.22178. Extended version at the SIAM Journal on Computing 18(1):186-208, 1989, DOI:10.1137/0218012. (Cited on page 229.)

[GMS08] V. Goyal, P. Mohassel, and A. Smith. *Efficient Two Party and Multi Party Computation Against Covert Adversaries*. In N. Smart (ed.), Advances in Cryptology — EUROCRYPT 2008, vol. 4965 of LNCS, pages 289–306. Springer, Heidelberg, 2008. DOI:10.1007/978-3-540-78967-3_17. (Cited on pages 73 and 74.)

[GMW87a] O. Goldreich, S. Micali, and A. Wigderson. *How to play ANY mental game (or a Completeness Theorem for Protocols with Honest Majority)*. In Proc. 19th Annual ACM Symposium on Theory of Computing — STOC 1987, pages 218–229. ACM, New York, 1987. DOI:10.1145/28395.28420. (Cited on pages 68 and 77.)

[GMW87b] O. Goldreich, S. Micali, and A. Wigderson. *How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design (Extended Abstract)*. In A. M. Odlyzko (ed.), Advances in Cryptology — CRYPTO 1986, LNCS, pages 171–185. Springer, Heidelberg, 1987. DOI:10.1007/3-540-47721-7_11. (Cited on page 230.)

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. *Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-knowledge Proof Systems*. Journal of the ACM, 38(3):690–728, 1991. DOI:10.1145/116825.116852. (Cited on page 230.)

[GMY06] J. A. Garay, P. MacKenzie, and K. Yang. *Strengthening Zero-Knowledge Protocols Using Signatures*. Journal of Cryptology, 19(2):169–209, 2006. DOI:10.1007/s00145-005-0307-3. Also at eprint:ia.cr/2003/037. (Cited on page 235.)

[GO96] O. Goldreich and R. Ostrovsky. *Software Protection and Simulation on Oblivious RAMs*. Journal of the ACM, 43(3):431–473, 1996. DOI:10.1145/233551.233553. (Cited on page 77.)

[Gol04] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, 2004. DOI:10.1017/CBO9780511721656. (Cited on pages 1 and 12.)

[Gro10] J. Groth. *Short Pairing-Based Non-interactive Zero-Knowledge Arguments*. In M. Abe (ed.), Advances in Cryptology — ASIACRYPT 2010, vol. 6477 of LNCS, pages 321–340. Springer, Heidelberg, 2010. DOI:10.1007/978-3-642-17373-8_19. (Cited on pages 76 and 171.)

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. *Faster Secure Two-Party Computation Using Garbled Circuits*. In Proc. 20th USENIX Security Symposium, pages 539–554. USENIX Association, 2011. ISBN ISBN 978-1-931971-87-4. PDF file at `usenix.org` website. (Cited on page 348.)

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. *A Pseudorandom Generator from any One-way Function*. SIAM Journal on Computing, 28(4):1364–1396, 1999. DOI:10.1137/S0097539793244708. (Cited on page 150.)

[HKE12] Y. Huang, J. Katz, and D. Evans. *Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution*. In Proc. S&P 2012, pages 272–284. IEEE Computer Society, Washington, 2012. DOI:10.1109/SP.2012.43. (Cited on page 71.)

[HKE13] Y. Huang, J. Katz, and D. Evans. *Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose*. In R. Canetti and J. Garay (eds.), Advances in Cryptology — CRYPTO 2013, vol. 8043 of LNCS, pages 18–35. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-40084-1_2. Also at eprint:ia.cr/2013/081. (Cited on pages 11, 75, 148, and 164.)

[HKK+14] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. *Amortizing Garbled Circuits*. In J. A. Garay and R. Gennaro (eds.), Advances in Cryptology —

CRYPTO 2014, Part II, vol. 8617 of LNCS, pages 458–475. Springer, Heidelberg, 2014. DOI:10.1007/978-3-662-44381-1_26. Also at eprint:ia.cr/2015/081. (Cited on pages 187, 208, and 210.)

[HMQ04] D. Hofheinz and J. Müller-Quade. *Universally Composable Commitments Using Random Oracles*. In M. Naor (ed.), TCC 2004, vol. 2951 of LNCS, pages 58–76. Springer, Heidelberg, 2004. DOI:10.1007/978-3-540-24638-1_4. (Cited on page 152.)

[HMQU06] D. Hofheinz, J. Müller-Quade, and D. Unruh. *On the (Im-)Possibility of Extending Coin Toss*. In S. Vaudenay (ed.), Advances in Cryptology — EUROCRYPT 2006, vol. 4004 of lncs, pages 504–521. Springer, Heidelberg, 2006. DOI:10.1007/11761679_30. Also at eprint:ia.cr/2006/177. (Cited on pages 150 and 151.)

[IAC14] IACR Members Meeting held at Eurocrypt 2014. *IACR Statement On Mass Surveillance — "Copenhagen Resolution"*, May 14, 2014. Text at `iacr.org` website. (Cited on page 202.)

[Ide13] Identity Assurance Programme. *Identity Assurance Hub Service SAML 2.0 Profile v1.1a*, September 11, 2013. PDF file (36 pages) at `gov.uk` website. SHA256: `1b5770b030526414 c5cdf136355837b3 c64d7f55d6dd4340 29370aab1bcde35c`. (Cited on pages 15 and 177.)

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. *Extending Oblivious Transfers Efficiently*. In D. Boneh (ed.), Advances in Cryptology — CRYPTO 2003, vol. 2729 of LNCS, pages 145–161. Springer, Heidelberg, 2003. DOI:10.1007/978-3-540-45146-4_9. (Cited on page 69.)

[ISO13] ISO/IEC JTC 1/SC 27. *ISO/IEC 20008-1:2013 — Information technology — Security techniques — Anonymous Digital Signatures*. Joint Technical committee of the International Organization for Standardization and the International Electrotechnical Commission — Subcommittee 27, 2013. `iso.org` website. (Cited on page 179.)

[JKO13] M. Jawurek, F. Kerschbaum, and C. Orlandi. *Zero-knowledge Using Garbled Circuits: How to Prove Non-algebraic Statements Efficiently*. In Proc. 20th ACM Conference on Computer and Communications Security — CCS 2013, pages 955–966. ACM, New York, 2013. DOI:10.1145/2508859.2516662. (Cited on page 189.)

[JL13] M. Joye and B. Libert. *Efficient Cryptosystems from 2^k-th Power Residue Symbols*. In T. Johansson and P. Nguyen (eds.), Advances in Cryptology — EUROCRYPT 2013, vol. 7881 of LNCS, pages 76–92. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-38348-9_5. Also at eprint:ia.cr/2013/435. (Cited on pages 206 and 330.)

[Joh12] A. John. *Challenges in Operationalizing Privacy in Identity Federations*. IDMGOV Info — U.S. FICAM program, 2012. Three blog posts at `info.idmanagement.gov` website: Part 1; Part 2; Part 3. (Cited on page 176.)

[JS07] S. Jarecki and V. Shmatikov. *Efficient Two-Party Secure Computation on Committed Inputs*. In M. Naor (ed.), Advances in Cryptology — EUROCRYPT 2007, vol. 4515 of LNCS, pages 97–114. Springer, Heidelberg, 2007. DOI:10.1007/978-3-540-72540-4_6. (Cited on page 76.)

[KAF+10] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. Riele, A. Timofeev, and P. Zimmermann. *Factorization of a 768-Bit RSA Modulus*. In T. Rabin (ed.), Advances in Cryptology — CRYPTO 2010, vol. 6223 of LNCS, pages 333–350. Springer, Heidelberg, 2010. DOI:10.1007/978-3-642-14623-7_18. Also at eprint:ia.cr/2010/006. (Cited on page 29.)

[Kir08] M. S. Kiraz. *Secure and Fair Two-Party Computation*. Ph.D. thesis, Technische Universiteit Eindhoven, 2008. DOI:10.13140/RG.2.1.4458.2004. (Cited on page 70.)

[KK12] V. Kolesnikov and R. Kumaresan. *Improved Secure Two-Party Computation via Information-Theoretic Garbled Circuits*. In I. Visconti and R. De Prisco (eds.), SCN 2012, vol. 7485 of LNCS, pages 205–221. Springer, Heidelberg, 2012. DOI:10.1007/978-3-642-32928-9_12. (Cited on pages 69 and 76.)

[KK13] V. Kolesnikov and R. Kumaresan. *Improved OT Extension for Transferring Short Secrets*. In R. Canetti and J. A. Garay (eds.), Advances in Cryptology — CRYPTO 2013, Part II, vol.

8043 of LNCS, pages 54–70. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-40084-1_4. Also at eprint:ia.cr/2013/491. (Cited on page 206.)

[KKW16] W. S. Kennedy, V. Kolesnikov, and G. T. Wilfong. *Overlaying Circuit Clauses for Secure Computation*. IACR Cryptology ePrint Archive, Report 2016/685, 2016. (Cited on page 208.)

[KM16] A. Küpçü and P. Mohassel. *Fast Optimistically Fair Cut-and-Choose 2PC*. IACR Cryptology ePrint Archive, Report 2015/1209, 2016. (Cited on page 210.)

[KMRR15] V. Kolesnikov, P. Mohassel, B. Riva, and M. Rosulek. *Richer Efficiency/Security Trade-offs in 2PC*. In Y. Dodis and J. B. Nielsen (eds.), TCC 2015, pages 229–259. Springer, Heidelberg, 2015. DOI:10.1007/978-3-662-46494-6_11. Also at eprint:ia.cr/2015/055. (Cited on pages 71 and 148.)

[Kol09] V. Kolesnikov. *Advances and impact of secure function evaluation*. Bell Labs Technical Journal, 14(3):187–192, 2009. DOI:10.1002/bltj.20396. (Cited on page 1.)

[KOS15] M. Keller, E. Orsini, and P. Scholl. *Actively Secure OT Extension with Optimal Overhead*. In R. Gennaro and M. Robshaw (eds.), Advances in Cryptology — CRYPTO 2015, Part I, vol. 9215 of LNCS, pages 724–741. Springer, Heidelberg, 2015. DOI:10.1007/978-3-662-47989-6_35. Also at eprint:ia.cr/2015/546. (Cited on page 206.)

[Kra94] H. Krawczyk. *Secret Sharing Made Short*. In D. Stinson (ed.), Advances in Cryptology — CRYPTO 1993, vol. 773 of LNCS, pages 136–146. Springer, Heidelberg, 1994. DOI:10.1007/3-540-48329-2_12. (Cited on page 166.)

[KS06] M. S. Kiraz and B. Schoenmakers. *A protocol issue for the malicious case of Yao's garbled circuit construction*. In Proc. 27th Symp. Information Theory in the Benelux, pages 283–290, 2006. CiteSeerX:10.1.1.140.2627. (Cited on pages 69 and 70.)

[KS08a] M. S. Kiraz and B. Schoenmakers. *An Efficient Protocol for Fair Secure Two-Party Computation*. In T. Malkin (ed.), CT-RSA 2008, vol. 4964 of LNCS, pages 88–105. Springer, Heidelberg, 2008. DOI:10.1007/978-3-540-79263-5_6. (Cited on page 70.)

[KS08b] V. Kolesnikov and T. Schneider. *Improved Garbled Circuit: Free XOR Gates and Applications*. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz (eds.), ICALP 2008, vol. 5126 of LNCS, pages 486–498. Springer, Heidelberg, 2008. DOI:10.1007/978-3-540-70583-3_40. (Cited on pages 71, 121, and 151.)

[KS08c] V. Kolesnikov and T. Schneider. *A Practical Universal Circuit Construction and Secure Evaluation of Private Functions*. In G. Tsudik (ed.), Financial Cryptography and Data Security 2008, vol. 5143 of LNCS, pages 83–97. Springer Berlin Heidelberg, 2008. DOI:10.1007/978-3-540-85230-8_7. (Cited on page 7.)

[KS16] Á. Kiss and T. Schneider. *Valiant's Universal Circuit is Practical*. In M. Fischlin and J.-S. Coron (eds.), Advances in Cryptology — EUROCRYPT 2016, Part I, pages 699–728. Springer Berlin Heidelberg, 2016. DOI:10.1007/978-3-662-49890-3_27. Also at eprint:ia.cr/2016/093. (Cited on page 7.)

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. *Billion-gate secure computation with malicious adversaries*. In Proc. 21st USENIX Security Symposium — Security'12, pages 285–300. USENIX Association, 2012. ISBN 978-931971-95-9. PDF file at `usenix.org` website. Also at eprint:ia.cr/2012/179. (Cited on pages 4, 332, and 348.)

[Lin03] Y. Lindell. *Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation*. Journal of Cryptology, 16(3):143–184, 2003. DOI:10.1007/s00145-002-0143-7. Also at eprint:ia.cr/2001/107. (Cited on pages 150 and 368.)

[Lin11] Y. Lindell. *Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption*. In K. Paterson (ed.), Advances in Cryptology — EUROCRYPT 2011, vol. 6632 of LNCS, pages 446–466. Springer, Heidelberg, 2011. DOI:10.1007/978-3-642-20465-4_25. Also at eprint:ia.cr/2011/180. (Cited on page 152.)

[Lin13] Y. Lindell. *Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries*. In

R. Canetti and J. Garay (eds.), Advances in Cryptology — CRYPTO 2013, vol. 8043 of LNCS, pages 1–17. Springer, Heidelberg, 2013. DOI:10.1007/978-3-642-40084-1_1. Also at eprint:ia.cr/2013/079. (Cited on pages 11, 75, 87, 98, 147, 148, and 164.)

[Lin15] Y. Lindell. *An Efficient Transform from Sigma Protocols to NIZK with a CRS and Non-programmable Random Oracle.* In Y. Dodis and J. Nielsen (eds.), Theory of Cryptography, vol. 9014 of LNCS, pages 93–109. Springer Berlin Heidelberg, 2015. DOI:10.1007/978-3-662-46494-6_5. Also at eprint:ia.cr/2014/710. (Cited on pages 23, 72, 233, and 254.)

[LN11] C. Lunemann and J. B. Nielsen. *Fully Simulatable Quantum-Secure Coin-Flipping and Applications.* In A. Nitaj and D. Pointcheval (eds.), AFRICACRYPT 2011, vol. 6737 of LNCS, pages 21–40. Springer, Heidelberg, 2011. DOI:10.1007/978-3-642-21969-6_2. Also at eprint:ia.cr/2011/065. (Cited on pages 150 and 152.)

[LP02] Y. Lindell and B. Pinkas. *Privacy Preserving Data Mining.* Journal of Cryptology, 15(3):177–206, 2002. DOI:10.1007/s00145-001-0019-2. (Cited on pages 1, 142, and 209.)

[LP07] Y. Lindell and B. Pinkas. *An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries.* In M. Naor (ed.), Advances in Cryptology — EUROCRYPT 2007, vol. 4515 of LNCS, pages 52–78. Springer, Heidelberg, 2007. DOI:10.1007/978-3-540-72540-4_4. Also at eprint:ia.cr/2008/049. (Cited on page 70.)

[LP09] Y. Lindell and B. Pinkas. *A Proof of Security of Yao's Protocol for Two-Party Computation.* Journal of Cryptology, 22(2):161–188, 2009. DOI:10.1007/s00145-008-9036-8. (Cited on pages 4 and 68.)

[LP11] Y. Lindell and B. Pinkas. *Secure two-party computation via cut-and-choose oblivious transfer.* In Y. Ishai (ed.), TCC 2011, vol. 6597 of LNCS, pages 329–346. Springer, Heidelberg, 2011. DOI:10.1007/978-3-642-19571-6_20. Also at eprint:ia.cr/2010/284. (Cited on pages 69, 70, 73, and 133.)

[LR14] Y. Lindell and B. Riva. *Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings.* In J. Garay and R. Gennaro (eds.), Advances in Cryptology — CRYPTO 2014, vol. 8617 of LNCS, pages 476–494. Springer Berlin Heidelberg, 2014. DOI:10.1007/978-3-662-44381-1_27. (Cited on pages 187, 208, and 210.)

[LR15] Y. Lindell and B. Riva. *Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries.* In Proc. 22nd ACM Conference on Computer and Communications Security — CCS 2015, pages 579–590. ACM, New York, 2015. DOI:10.1145/2810103.2813666. Also at eprint:ia.cr/2015/987. (Cited on page 148.)

[Lub02] M. Luby. *LT codes.* In Proc. 43rd Annual Symposium on Foundations of Computer Science — FOCS 2002, pages 271–280. IEEE Computer Society, 2002. DOI:10.1109/SFCS.2002.1181950. (Cited on page 166.)

[Mau09] U. Maurer. *Unifying Zero-Knowledge Proofs of Knowledge.* In B. Preneel (ed.), Progress in Cryptology — AFRICACRYPT 2009, vol. 5580 of LNCS, pages 272–286. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-02384-2_17. (Cited on page 252.)

[MF06] P. Mohassel and M. Franklin. *Efficiency Tradeoffs for Malicious Two-Party Computation.* In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin (eds.), Public-Key Cryptography — PKC 2006, vol. 3958 of LNCS, pages 458–473. Springer, Heidelberg, 2006. DOI:10.1007/11745853_30. (Cited on pages 70 and 71.)

[MR13] P. Mohassel and B. Riva. *Garbled Circuits Checking Garbled Circuits: More Efficient and Secure Two-Party Computation.* In R. Canetti and J. A. Garay (eds.), Advances in Cryptology — CRYPTO 2013, Part II, vol. 8043 of LNCS, pages 36–53. SBH, 2013. DOI:10.1007/978-3-642-40084-1_3. Also at eprint:ia.cr/2013/051. (Cited on page 77.)

[MRH04] U. Maurer, R. Renner, and C. Holenstein. *Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology.* In M. Naor (ed.), TCC 2004,

vol. 2951 of LNCS, pages 21–39. Springer, Heidelberg, 2004. DOI:10.1007/978-3-540-24638-1_2. Also at eprint:ia.cr/2003/161. (Cited on page 234.)

[MSS14] P. Mohassel, S. Sadeghian, and N. P. Smart. *Actively Secure Private Function Evaluation*. In T. Sarkar, Palashand Iwata (ed.), Advances in Cryptology — ASIACRYPT 2014, Part II, vol. 8874 of LNCS, pages 486–505. Springer Berlin Heidelberg, 2014. DOI:10.1007/978-3-662-45608-8_26. Also at eprint:ia.cr/2014/102. (Cited on page 7.)

[Nao91] M. Naor. *Bit commitment using pseudorandomness*. Journal of Cryptology, 4(2):151–158, 1991. DOI:10.1007/BF00196774. (Cited on pages 39 and 150.)

[Nat15] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS Pub 202, NIST-ITL, U.S. Department of Commerce, August 2015. DOI:10.6028/NIST.FIPS.202. (Cited on page 13.)

[Nie02] J. B. Nielsen. *Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case*. In M. Yung (ed.), Advances in Cryptology — CRYPTO 2002, vol. 2442 of LNCS, pages 111–126. Springer, Heidelberg, 2002. DOI:10.1007/3-540-45708-9_8. (Cited on pages 22, 233, and 234.)

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. *A New Approach to Practical Active-Secure Two-Party Computation*. In R. Safavi-Naini and R. Canetti (eds.), Advances in Cryptology — CRYPTO 2012, vol. 7417 of LNCS, pages 681–700. Springer, Heidelberg, 2012. DOI:10.1007/978-3-642-32009-5_40. Also at eprint:ia.cr/2011/091. (Cited on pages 69 and 77.)

[NO09] J. B. Nielsen and C. Orlandi. *LEGO for Two-Party Secure Computation*. In O. Reingold (ed.), TCC 2009, vol. 5444 of LNCS, pages 368–386. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-00457-5_22. Also at eprint:ia.cr/2008/427. (Cited on pages 70, 76, and 148.)

[NP01] M. Naor and B. Pinkas. *Efficient oblivious transfer protocols*. In SODA 2001, pages 448–457. SIAM, Philadelphia, 2001. Indexed at ACM-DL:365411.365502. (Cited on page 78.)

[NPS99] M. Naor, B. Pinkas, and R. Sumner. *Privacy preserving auctions and mechanism design*. In Proc. EC 1999, pages 129–139. ACM, New York, 1999. DOI:10.1145/336992.337028. (Cited on pages 68 and 71.)

[NR16] J. B. Nielsen and S. Ranellucci. *Reactive Garbling: Foundation, Instantiation, Application*. In J. H. Cheon and T. Takagi (eds.), Advances in Cryptology — ASIACRYPT 2016, Part II, vol. 10032 of LNCS, pages 1022–1052. Springer, Heidelberg, 2016. DOI:10.1007/978-3-662-53890-6_34. Also at eprint:ia.cr/2015/693. (Cited on page 209.)

[NS14] J. B. Nielsen and M. Strefler. *Invisible Adaptive Attacks*. IACR Cryptology ePrint Archive, Report 2014/597, 2014. (Cited on page 39.)

[NST13] NSTIC National Program Office. *NSTIC Requirements Document*, September 10, 2013. Xlsx file at `idecosystem.org` website. SHA256: e359eea01c79b415 0048628273ec4f62 5faaa8c397c187aa 998b09128741dfd2. (Cited on pages 173, 182, and 185.)

[NST16] J. B. Nielsen, T. Schneider, and R. Trifiletti. *Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO*. The Network and Distributed System Security Symposium (NDSS) 2017, 2016. DOI:10.14722/ndss.2017.23075. Also at eprint:ia.cr/2016/1069. (Cited on page 148.)

[NY89] M. Naor and M. Yung. *Universal One-way Hash Functions and Their Cryptographic Applications*. In Proc. 21st Annual ACM Symposium on Theory of Computing — STOC 1989, pages 33–43. ACM, New York, 1989. DOI:10.1145/73007.73011. (Cited on page 150.)

[NZM91] I. M. Niven, H. S. Zuckerman, and H. L. Montgomery. *An introduction to the theory of numbers*. Wiley, fifth edition, 1991. ISBN: 9780471625469. (Cited on page 26.)

[OAS05] OASIS. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS Standard — March 15, 2005. PDF file (86 pages) at `oasis-open.org` website. SHA256: dc0890f88bfe862c b0bedb03491cc41c 9b39672bed76e2dc 161aea472b0468ab. (Cited on page 195.)

[Oka93] T. Okamoto. *Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes*. In E. Brickell (ed.), Advances in Cryptology — CRYPTO 1992, vol. 740 of LNCS, pages 31–53. Springer, Heidelberg, 1993. DOI:10.1007/3-540-48071-4_3. (Cited on page 252.)

[Ope] OpenID Foundation. *OpenID Connect*. Available at the `openid.net` website. (Cited on page 179.)

[Pai99] P. Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In J. Stern (ed.), Advances in Cryptology — EUROCRYPT 1999, vol. 1592 of LNCS, pages 223–238. Springer, Heidelberg, 1999. DOI:10.1007/3-540-48910-X_16. (Cited on pages 206 and 330.)

[Pas03] R. Pass. *On Deniability in the Common Reference String and Random Oracle Model*. In D. Boneh (ed.), Advances in Cryptology — CRYPTO 2003, vol. 2729 of LNCS, pages 316–337. Springer, Heidelberg, 2003. DOI:10.1007/978-3-540-45146-4_19. (Cited on pages 36 and 233.)

[Ped92] T. P. Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. In J. Feigenbaum (ed.), Advances in Cryptology — CRYPTO 1991, vol. 576 of LNCS, pages 129–140. Springer, Heidelberg, 1992. DOI:10.1007/3-540-46766-1_9. (Cited on pages 64 and 66.)

[PGV94] B. Preneel, R. Govaerts, and J. Vandewalle. *Hash functions based on block ciphers: a synthetic approach*. In D. R. Stinson (ed.), Advances in Cryptology — CRYPTO 1993, pages 368–378. Springer Berlin Heidelberg, 1994. DOI:10.1007/3-540-48329-2_31. (Cited on page 190.)

[Pin03] B. Pinkas. *Fair Secure Two-Party Computation*. In E. Biham (ed.), Advances in Cryptology — EUROCRYPT 2003, vol. 2656 of LNCS, pages 647–647. Springer, Heidelberg, 2003. DOI:10.1007/3-540-39200-9_6. (Cited on pages 8 and 69.)

[Pri14] Privacy and Consumer Advisory Group (PCAG). *Identity Assurance Principles — v3.1 (for publication)*, 2014. PDF file (12 pages) at `gov.uk` website. SHA256: `3f3a08a84e35f6e2 3ebc9f691f1816d1 0c303ce10ecd1805 3be2eb7a08727bd5`. (Cited on pages 15, 182, and 185.)

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams. *Secure Two-Party Computation Is Practical*. In M. Matsui (ed.), Advances in Cryptology — ASIACRYPT 2009, vol. 5912 of LNCS, pages 250–267. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-10366-7_15. Also at eprint:ia.cr/2009/314. (Cited on pages 1, 70, and 71.)

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. *A Framework for Efficient and Composable Oblivious Transfer*. In D. Wagner (ed.), Advances in Cryptology — CRYPTO 2008, vol. 5157 of LNCS, pages 554–571. Springer Berlin Heidelberg, 2008. DOI:10.1007/978-3-540-85174-5_31. Also at eprint:ia.cr/2007/348. (Cited on pages 78, 82, and 333.)

[PW09] R. Pass and H. Wee. *Black-Box Constructions of Two-Party Protocols from One-Way Functions*. In O. Reingold (ed.), TCC 2009, vol. 5444 of LNCS, pages 403–418. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-00457-5_24. (Cited on page 150.)

[Rab80] M. O. Rabin. *Probabilistic algorithm for testing primality*. Journal of Number Theory, 12(1):128–138, 1980. DOI:10.1016/0022-314X(80)90084-0. (Cited on page 25.)

[Rab81] M. O. Rabin. *How to exchange secrets with oblivious transfer*. Technical Report TR-81, Harvard University, Aiken Computation Lab, Cambridge, MA, 1981. See typeset version at the IACR Cryptology ePrint Archive, Report 2005/187. (Cited on page 78.)

[Rab89] M. O. Rabin. *Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance*. Journal of the ACM, 36(2):335–348, 1989. DOI:10.1145/62044.62050. (Cited on page 166.)

[Rom90] J. Rompel. *One-way Functions Are Necessary and Sufficient for Secure Signatures*. In Proc. 22nd Annual ACM Symposium on Theory of Computing — STOC 1990, pages 387–394. ACM, New York, 1990. DOI:10.1145/100216.100269. (Cited on page 150.)

[RR16] P. Rindal and M. Rosulek. *Faster malicious 2-party secure computation with online/offline dual execution*. In Proc. 25th USENIX Security Symposium, pages 297–314. USENIX Association, 2016. ISBN 978-1-931971-32-4. PDF file at `usenix.org` website. Also at eprint:ia.cr/2016/632. (Cited on page 148.)

[RS60] I. S. Reed and G. Solomon. *Polynomial codes over certain finite fields.* Journal of the SIAM, 8(2):300–304, 1960. DOI:10.1137/0108018. (Cited on page 166.)

[Rus95] A. Russell. *Necessary and sufficient conditions for collision-free hashing.* Journal of Cryptology, 8(2):87–99, 1995. DOI:10.1007/BF00190757. (Cited on page 150.)

[Sah99] A. Sahai. *Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security.* In Proc. 40th Annual Symposium on Foundations of Computer Science — FOCS 1999, pages 543–553, Washington, DC, USA, 1999. IEEE Computer Society. DOI:10.1109/SFFCS.1999.814628. (Cited on page 233.)

[SBC+12] N. Smart, S. Babbage, D. Catalano, C. Cid, B. de Weger, O. Dunkelman, C. Gehrmann, L. Granboulan, T. Güneysu, J. Hermans, T. Lange, A. Lenstra, C. Mitchell, M. Näslund, P. Nguyen, C. Paar, K. Paterson, J. Pelzl, T. Pornin, B. Preneel, C. Rechberger, V. Rijmen, M. Robshaw, A. Rupp, M. Schläffer, S. Vaudenay, F. Vercauteren, and M. Ward. *ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012).* Revision 1.0, ECRYPT II — European Network of Excellence in Cryptology II, September 2012. PDF file (124 pages) at `ecrypt.eu.org` website. SHA256: 057be7cbbb600df7 a18c87283b4a50dd b40100ee32343be2 e25dc1b697b6104c. (Cited on pages 25, 29, and 336.)

[Sch91] C. Schnorr. *Efficient signature generation by smart cards.* Journal of Cryptology, 4(3):161–174, 1991. DOI:10.1007/BF00196725. (Cited on pages 211, 231, 251, and 252.)

[SDCP00] A. Santis, G. Di Crescenzo, and G. Persiano. *Necessary and Sufficient Assumptions for Non-interactive Zero-Knowledge Proofs of Knowledge for All NP Relations.* In U. Montanari, J. Rolim, and E. Welzl (eds.), ICALP 2000, vol. 1853 of LNCS, pages 451–462. Springer, Heidelberg, 2000. DOI:10.1007/3-540-45022-X_38. (Cited on pages 12 and 43.)

[Sha79] A. Shamir. *How to Share a Secret.* Commun. ACM, 22(11):612–613, November 1979. DOI:10.1145/359168.359176. (Cited on page 166.)

[Sho01] V. Shoup. *OAEP Reconsidered.* In J. Kilian (ed.), Advances in Cryptology — CRYPTO 2001, vol. 2139 of LNCS, pages 239–259. Springer, Heidelberg, 2001. DOI:10.1007/3-540-44647-8_15. (Cited on page 339.)

[Sho06] A. Shokrollahi. *Raptor Codes.* IEEE Trans. Information Theory, 52:2551–2567, 2006. DOI:10.1109/TIT.2006.874390. (Cited on page 166.)

[Sim98] D. R. Simon. *Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?* In K. Nyberg (ed.), Advances in Cryptology — EUROCRYPT 1998, vol. 1403 of LNCS, pages 334–345. Springer, Heidelberg, 1998. DOI:10.1007/BFb0054137. (Cited on page 150.)

[SP92] A. D. Santis and G. Persiano. *Zero-knowledge proofs of knowledge without interaction.* In Proc. 33rd Annual Symposium on Foundations of Computer Science — FOCS 1992, pages 427–436. IEEE, 1992. DOI:10.1109/SFCS.1992.267809. (Cited on page 233.)

[SS11] A. Shelat and C.-h. Shen. *Two-Output Secure Computation with Malicious Adversaries.* In K. Paterson (ed.), Advances in Cryptology — EUROCRYPT 2011, vol. 6632 of LNCS, pages 386–405. Springer, Heidelberg, 2011. DOI:10.1007/978-3-642-20465-4_22. Also at eprint:ia.cr/2011/533. (Cited on pages 8, 70, and 96.)

[TBBC10] M. S. Turan, E. Barker, W. Burr, and L. Chen. *NIST Special Publication 800-132 — Recommendation for Password-Based Key Derivation (Part 1: Storage Applications).* National Institute of Standards and Technology, U.S. Department of Commerce, December 2010. DOI:10.6028/NIST.SP.800-132. (Cited on page 72.)

[The11] The White House. *National Strategy for Trusted Identities in Cyberspace — Enhancing Online Choice, Efficiency, Security, and Privacy*, April 2011. PDF file (52 pages) at `whitehouse.gov` website. SHA256: 2dfcb30eca1bf3e0 2b91addaaabde5b9 bfc463bfd270ba43 fc478af0bfa621e3. (Cited on pages 15 and 175.)

[TLL03] C. Tang, Z. Liu, and J. Liu. *The Statistical Zero-knowledge Proof for Blum Integer Based on Discrete Logarithm.* IACR Cryptology ePrint Archive, Report 2003/232, 2003. (Cited on page 238.)

[Uni13] United States Postal Service. *Solicitation Number 1B-13-A-0003 — Federal Cloud Credential*

Exchange (FCCX). Federal Business Opportunities (FedBizOpps), 2013. ZIP files at `fbo.gov` website: "RFP_Documents.zip" (January 10) SHA256: 72c68c9fe1be32da 2c586edab757d933 385811fada90c0b9 57ee9a6f3ecfb13a; "Amendment_5.zip" (January 28) SHA256: a61df077ee03fb3c 56d6ded65eb5058e 2a1db13deadb2ba7 961a237d81e6213b. (Cited on pages 15, 175, 183, 186, and 210.)

[USP14] USPS — Information Security and Privacy Advisory Board. *FCCX Briefing*, June 13, 2014. PDF file (24 slides) at `csrc.nist.gov` website. SHA256: 144135e4bb08419f d6904446fcb12e447 854620780e93a1fb 74b9afa6249c0c99. (Cited on page 176.)

[Val76] L. G. Valiant. *Universal Circuits (Preliminary Report)*. In Proc. 8th Annual ACM Symposium on Theory of Computing — STOC 1976, pages 196–203. ACM, New York, 1976. DOI:10.1145/800113.803649. (Cited on page 7.)

[vdGP88] J. van de Graaf and R. Peralta. *A Simple and Secure Way to Show the Validity of Your Public Key*. In C. Pomerance (ed.), Advances in Cryptology — CRYPTO 1987, vol. 293 of LNCS, pages 128–134. Springer, Heidelberg, 1988. DOI:10.1007/3-540-48184-2_9. (Cited on pages 26 and 238.)

[VNC03] P. Veríssimo, N. Neves, and M. Correia. *Intrusion-Tolerant Architectures: Concepts and Design*. In R. de Lemos, C. Gacek, and A. Romanovsky (eds.), Architecting Dependable Systems, vol. 2677 of LNCS, pages 3–36. Springer Berlin / Heidelberg, 2003. DOI:10.1007/3-540-45177-3_1. (Cited on page 2.)

[VZ12] S. Vadhan and C. J. Zheng. *Characterizing Pseudoentropy and Simplifying Pseudorandom Generator Constructions*. In Proc. 44th Annual ACM Symposium on Theory of Computing — STOC 2012, pages 817–836. ACM, New York, 2012. DOI:10.1145/2213977.2214051. (Cited on page 150.)

[Wee09] H. Wee. *Zero Knowledge in the Random Oracle Model, Revisited*. In M. Matsui (ed.), Advances in Cryptology — ASIACRYPT 2009, vol. 5912 of LNCS, pages 417–434. Springer, Heidelberg, 2009. DOI:10.1007/978-3-642-10366-7_25. (Cited on page 233.)

[Wik16] D. Wikström. *Simplified Universal Composability Framework*. In E. Kushilevitz and T. Malkin (eds.), Theory of Cryptography — TCC 2016, Part I, vol. 9562 of LNCS, pages 566–595. Springer, Heidelberg, 2016. DOI:10.1007/978-3-662-49096-9_24. (Cited on page 34.)

[WMK16] X. S. Wang, A. J. Malozemoff, and J. Katz. *Faster Two-Party Computation Secure Against Malicious Adversaries in the Single-Execution Setting*. IACR Cryptology ePrint Archive, Report 2016/762, 2016. (Cited on page 148.)

[Woo07] D. P. Woodruff. *Revisiting the Efficiency of Malicious Two-Party Computation*. In M. Naor (ed.), Advances in Cryptology — EUROCRYPT 2007, vol. 4515 of LNCS, pages 79–96. Springer, Heidelberg, 2007. DOI:10.1007/978-3-540-72540-4_5. Also at eprint:ia.cr/2006/397. (Cited on page 70.)

[Wre12] S. Wreyford. *Identity Assurance goes to Washington*. Government Digital Service Blog, May 29, 2012. Blog post at `gds.blog.gov.uk` website. (Cited on page 178.)

[Yao82] A. C. Yao. *Protocols for secure computations*. In Proc. 23rd Annual Symposium on Foundations of Computer Science — FOCS 1982, pages 160–164. IEEE Computer Society, 1982. DOI:10.1109/SFCS.1982.88. (Cited on page 1.)

[Yao86] A. C.-C. Yao. *How to generate and exchange secrets*. In Proc. 27th Annual Symposium on Foundations of Computer Science — FOCS 1986, vol. 0, pages 162–167. IEEE Computer Society, 1986. DOI:10.1109/SFCS.1986.25. (Cited on pages 1, 3, 4, and 68.)

[ZHKS16] R. Zhu, Y. Huang, J. Katz, and A. Shelat. *The Cut-and-Choose Game and its Application to Cryptographic Protocols*. In Proc. 25th USENIX Security Symposium, pages 1085–1100. USENIX Association, 2016. ISBN 978-1-931971-32-4. PDF file at `usenix.org` website. (Cited on page 98.)

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. *Two Halves Make a Whole*. In E. Oswald and M. Fischlin (eds.), Advances in Cryptology — EUROCRYPT 2015, vol. 9057 of LNCS, pages 220–250. Springer Berlin Heidelberg, 2015. DOI:10.1007/978-3-662-46803-6_8. Also at eprint:ia.cr/2014/756. (Cited on pages 71, 121, and 338.)

Appendix A

Non-interactive zero-knowledge sub-protocols

This chapter describes several non-interactive (NI) zero-knowledge (ZK) sub-protocols useful throughout the dissertation. Section A.1 starts with an overview of basic notions and general techniques, including transformations from interactive to non-interactive ZK sub-protocols. Section A.2, specific for IFC, discusses several security problems that may arise from the use of an incorrect Blum integer (i.e., if a ZKP of correctness is not performed) and then describes a respective NIZKP of correctness and a NIZKPoK of trapdoor. Section A.3 deals with NIZK sub-protocols for DLC, including a NIZKPoK of DL, a NIZK of of same DL and a NIZK of correct ElGamal BitComs. Section A.4 describes a NIZKP of same committed bits across different commitment schemes, based on XOR homomorphic properties.

A.1 Overview of types of ZK sub-protocols

This section reviews notions about zero-knowledge protocols, namely ZK proofs of membership (ZKPs) vs. ZK proofs of Knowledge (ZKPoKs), proofs vs. arguments (though then continuing to use "proof" as an umbrella term), Sigma (Σ) and honest-verifier (HV) ZK sub-protocols, transformations for non-interactive NIZKPs and NIZKPoK, and OR proofs.

A.1.1 Proofs and arguments of membership and of knowledge

Zero-knowledge proofs (of language membership). Zero-knowledge proofs (ZKPs) (of language membership) [GMR85] allow a Prover (P) to convince a Verifier (V) of the correctness

of an assertion (e.g., that a public element belongs to a certain language), but without revealing anything that would allow V to produce an element distinguishable from anything that it could have produced before the interaction. This kind of proof is possible for all statements in the non-deterministic polynomial time (NP) class, i.e., decidable in polynomial time when possessing the adequate auxiliary information [BC87, GMW87b, GMW91]. The original ZKP definition assumed provers with unbounded computational power, but variants are possible, e.g., unbounded verifiers and polynomially constrained provers [Cha87].

Needed properties. The needed properties of a ZKP are *completeness*, *soundness* and *zero-knowledge*. *Completeness* guarantees that if both parties are honest then V accepts the proof. The second and third properties deal with the cases where P (but not V) and V (but not P) are dishonest, respectively. More specifically: *soundness* ensures that, except with negligible probability, V only accepts the proof if the assertion is correct (i.e., if the public element belongs to the language), even if P maliciously deviates from the protocol execution; *zero-knowledge* ensures that the transcript made available to V (i.e., the view of the execution, including the internal states of V), when interacting with an honest P, has a distribution indistinguishable from one that V could produce by herself alone. The notion of indistinguishability is used here in the liberal *computational* sense (i.e., what a party restricted to probabilistic polynomial time is able to distinguish with noticeable probability), but more restricted notions of indistinguishability exist, namely *statistical* and *perfect*.

Arguments vs. proofs. In ZKPs the soundness property is statistical, i.e., except with negligible probability an unbounded prover is not able to cheat in the protocol. In contrast, ZK *arguments* only require computational soundness [BCC88], i.e., an unconditionally bounded prover may be successful even when the statement is false or when the claimed knowledge does not exist. For simplicity, "proof" is used in this dissertation as an umbrella term encompassing the *argument* and the (original) *proof* nuances, and always assuming that both prover and verifier are computationally bounded to probabilistic polynomial time.

Zero knowledge proofs of knowledge. In contrast to ZKPs of language membership, ZK proofs of knowledge (ZKPoK) allow a prover to prove knowledge of a secret. Here the *soundness* property guarantees that only a prover with the actual valid secret is able to make V accept the proof (except with negligible probability). By definition, a ZKPoK

requires that a simulator (a.k.a. knowledge extractor) impersonating V while playing against a black-box P is able to extract the secret knowledge of P, with a probability at least as large as (a value negligible close to) the probability of the prover succeeding in the proof [FFS88, FS90a, BG93]. For example, in the case of NP-languages a ZKPoK may be used to prove knowledge of a respective witness. In case of one-way functions, it may be used to prove knowledge of a pre-image of a public element, e.g., knowledge of a representation of a discrete log of a certain group element. Even in cases where V already know witness(es), ZKPoKs may make sense, e.g., to ensure non-malleability, and/or as a ZKP of correct behavior.

Specific protocols may combine a ZKPoK and a ZKP, e.g., a ZKPoK of a Blum integer trapdoor may simultaneously be a ZKP of the correctness of the Blum integer. Conversely, a NIZKP of correctness of a Blum integer does not necessarily have to allow extraction of the trapdoor, and a ZKPoK may become more efficient if correctness is already assumed.

A.1.2 Sigma and HVZK protocols

Sigma and HVZK protocols. The term "Sigma" (Σ) protocol (introduced in [Cra96]) denotes a three move protocol [Dam10], for proving language membership in a certain language, with the following structure and properties: in a first Σ-commit message, P sends a value that binds him to some value, e.g., by sending the image of some random value upon application of a one-way function (not necessarily hiding the pre-image in a semantic way); in a second message, V sends a Σ-challenge to P, uniformly selected from some known distribution; in a third message, P sends a Σ-response, whose value depends on the challenge, the initial commitment and the assertion being proven (and can be calculated based on an implicit secret witness of P); it ensures *completeness* (V always accepts if both parties are honest), *special soundness* (a pair of valid transcripts with same Σ-commit but different Σ-challenge and Σ-response allows extraction of the secret) and *special honest-verifier zero knowledge* (special HVZK) (transcripts of interactions by an honest V are simulatable, i.e., can be generated with an indistinguishable distribution even without knowing the secret of P, including when conditioning the Σ-challenge to be a specific value). This structure has been extensively used in ZK protocols, e.g., in Schnorr protocol [Sch91] and is itself similar to the usual cut-and-choose structure in diverse S2PC protocols.

A.1.3 Transformations from HVZK to interactive ZK

The HVZK property of Σ protocols does not guarantee ZK against a malicious V. The canonical example is V choosing the Σ-challenge as a one-way hash function of the public input and the Σ-commit message. Specifically, this breaks the non-transferability aspect of ZK — after a successful real protocol execution, V would be able to "convince" an external party that the Σ-challenge was selected after the Σ-commit message, which means that the subsequent valid Σ-response must have been produced by an actual P that knew a valid secret input. Nonetheless, a transformation from special HVZK to full ZK can be obtained. In an interactive setting, this can be based on commitment schemes, as exemplified hereafter.

In one approach, V may use an Ext-Com scheme to initially commit to the Σ-challenge, and open it only after receiving the Σ-commit message P. A simulator $\mathcal{S}$ impersonating P is able to extract the Σ-challenge built by the black-box V, before $\mathcal{S}$ decides on behalf of P the Σ-commit element. By the HVZK property, $\mathcal{S}$ is then able to produce respective Σ-commit and Σ-response elements that are consistent with the Σ-challenge, and using them in the respective subsequent parts of the protocol execution.

On a different approach [Dam00], P may start by using an Equiv-Com to commit to the Σ-commit element, then receive the Σ-challenge from V and only then open the Σ-commit element from the Equiv-Com and reveal the needed Σ-response. In a simulated execution, after learning the Σ-challenge the simulator is still able to produce a consistent of Σ-commit and Σ-response elements and then equivocate the calculated Σ-commit element, thus continuing a successful execution.

On a very related approach, the Σ-challenge could be obtained via a coin-flipping simulatable against a malicious V (e.g., P sends an Equiv-Com of a contribution, P sends a random contribution, P opens her contribution and both parties calculate the Σ-challenge as the XOR of both contributions). In this case the simulator starts by simulating a transcript with the three Σ elements (commit, challenge, response), then sends the Σ-commit and then induces the coin-flipping to yield the intended Σ-challenge element.

A.1.4 Transformations from HVZK to NIZK

Toward enabling non-interactive ZKPs, the Σ-challenge may be calculated non-interactively by P. However, to ensure soundness it must not be possible for P to learn the Σ-challenge

before it is bound to a Σ-commit element. There are several approaches in the literature.

Based on a CRS. Non-interactive zero-knowledge was originally introduced with a technique [BFM88] based on one-time-use common reference string (CRS) and specific number theoretical assumptions. Subsequent developments showed how to use a single CRS in multiple protocol executions and under general assumptions [FLS90]. Non-interactiveness can also be achieved for ZKPoKs [SP92], as non-malleability can also be achieved for both NIZKPs and NIZKPoKs [Sah99, DSDCO+01].

Based on a local programmable random-oracle. For a more efficient transformation, the Fiat-Shamir approach [FS87] generates a random Σ-challenge as the image of a random hash function applied to the concatenation of the public element (i.e., the instance being proven) and the Σ-commit element. In the simulation, after selecting a Σ-commit element the simulator impersonating P is able to "program" the random hash function to output the single Σ-challenge for which it can produce a valid Σ-response. The notion of random hash function was subsequently formalized as the *Random Oracle* model [BR93], though therein with a ZKP example where only the Σ-commit elements would be hashed. Subsequent work [BPW12] explored differences between hashing only the Σ-commit element vs. hashing also the public element, and showed that in general the later is preferable, to preclude some malleability attacks. If the random oracle is local (i.e., only accessible to P and V), then its programmability ensures ZK. However, if it is public/global then the procedure breaks the non-transferability of ZK [Pas03]. In fact, NIKZ cannot be obtained solely based on a *non-programmable random oracle* (NPRO) [Wee09], but this dissertation also considers the further use of a CRS and additional unique identifiers defining each execution context.

Based on a global non-programmable random-oracle (NPRO). Combining the CRS and NPRO ingredients it is possible to improve efficiency and applicability. Essentially, the techniques from Damgård [Dam00] and Fiat and Shamir [FS87] can be combined so that a simulator impersonating P is able to equivocate the Σ-commit message after determining the Σ-challenge obtained as a random hash of the Equiv-Com of the Σ-commit and the public element. This transformation has been proposed by Lindell [Lin15] showing that a non-programmable random oracle (NPRO) can be used [Nie02], because the ZK property no longer depends on the NPRO randomness, but instead on the equivocability of the Σ-commit

message. Since only soundness depends on the NPRO, the NPRO can be globally defined. The NIZKPs in this dissertation will use this Equiv-Com and NPRO-Hash combination.

It is worth emphasizing that simulatability based on a global NPRO is much more meaningful than one based on a *programmable random oracle*. In the later both the ZK and soundness properties are based on the random oracle, and there the ZK dependency directly implies that the random oracle cannot be replaced by any concrete hash function, lest it would break the non-transferability property. Any external party with access to a valid NIZK protocol transcript would be able to recompute hashes and verify that the inherent Σ-challenge was indeed obtained via the random oracle. There are other examples of applications where the replacement of a random oracle by any specific function (or one randomly selected from a set of efficiently computable functions) does not retain security [CGH04] (therein with some "unnatural" examples) and in fact a clear separation has been shown between the programmable and non-programmable uses of random oracles [Nie02]. For the use of a NPRO in this dissertation, where it only aids soundness (i.e., preventing the ability to cheat in a proof), there might be a cryptographic hash function replacement that enables simulatability for concrete NIZK protocols. While it remains an open problem proving the sufficiency of a general set of properties that could be required from a concrete function (or function family) in order to allow NIZKP, the converse (breaking soundness when using a concrete cryptographic hash function) would intuitively break some expected property of a cryptographic hash, e.g., indifferentiability [MRH04] or correlation intractability [CGH04] with respect to a particular type of transformation between Σ-commit and Σ-challenge elements, namely when also feeding to the hash pre-image an unpredictable component defined per session. This remains an open problem and so the proofs are given in the NPRO model.

A.1.5 Non-interactive ZKPoKs

If simulation with rewinding is allowed, the special soundness of Σ-protocols enables extraction of the witness whose knowledge is being proven. However, extraction is not possible in general without rewinding, and the same applies for its described transformations into NIZKPs.

Omega (Ω) protocols. A possible augmentation of Σ-protocols into ZKPoKs that allow extraction without rewinding involves a CRS with the public parameters of a public-key

encryption scheme. In an Ω-protocol [GMY06], besides a regular Σ-protocol the prover also sends to the verifier an encryption of the secret witness and a ZKP that the encrypted element is indeed the witness related to the Σ-protocol. Since soundness is guaranteed by the Σ-protocols, a simulator is then able to extract the witness from the ciphertext, by decryption using the secret-key associated with the CRS public-key.

Cut-and-choose leveraging of special soundness. The two main NIZKPoKs described in this chapter, namely a NIZKPoK of Blum integer trapdoor (§A.2.3), and NIZKPoK of discrete log (§A.3.3), use a related but different idea, which leverages the special soundness property of Σ-protocols. A CRS is also used to enable extraction (and equivocation), but the extraction is instead based on a cut-and-choose approach that enforces statistical soundness. Essentially, the CRS is used to describe the public parameters of an Ext-Com scheme and an Equiv-Com scheme (possibly both based on the same trapdoor). Then, for the same random Σ-commit message, P uses the Ext-Com to commit several Σ-responses, one of which is equal to the one obtained by the NPRO-hash application on top of the Equiv-Com. P then composes the NIZKPoK transcript as the concatenation of the Σ-commit, the Ext-Coms of the non-selected Σ-responses and the opening of the selected Σ-responses. In comparison with an Ω-protocol approach, this avoids a direct NIZKP of consistency between an encrypted value and the secret witness, which could require a protocol tailored to the structure of the encryption scheme, and instead enables any abstract Ext-Com and Equiv-Com; as a disadvantage, in terms of computation complexity the cut-and-choose approach requires committing elements in number proportional to the statistical parameter.

A.1.6 OR proof

If Σ-HVZK protocols exist for two types of assertions (A and B), then it is possible to construct a Σ-HVZK protocol for a respective disjunction, i.e., for an assertion of type "A OR B." A concrete application is proving that an ElGamal BitCom is correct (§A.3.2), i.e., that it commits to a 0 OR 1. The essential trick for such a "OR proof" [CDS94] is to let P choose the Σ-challenges in a correlated way, such that in practice it can fully determine one of the Σ-challenges — the one associated with the assertion that it does not know how to prove — but not control the other Σ-challenge. For example, this can be determined by V sending a random string and then giving flexibility for P to choose any two Σ-challenges (one

for each assertion) whose XOR is equal to the random string determined by V. The same applies in case of a transformation for NIZK, e.g., with the "random" string being instead defined as the NPRO-hash of an Equiv-Com of all the pair of assertions and the respective pair of Σ-commit elements. The, playing the two Σ-protocols in parallel, P simulates one, and executes the other as an honest P would, only having to additionally inform V about one of the Σ-challenges. Thus, this requires communication approximately equal to what two independent proofs would require (and an extra Σ-challenge in the non-interactive case).

A.2 NIZK sub-protocols for IFC

This section considers the problem of ensuring correctness of Blum integers, proving knowledge of its factors, and another proof about GM BitComs, when having access to a short trusted CRS and a global NPRO, in a simulation setting without rewinding, but with the simulator being having access to define the CRS. §A.2.1 discusses potential security problems of using moduli that are not Blum integers. §A.2.2 describes a NIZKP of Blum integer correctness, which is sound but does not allowing extraction of the trapdoor. §A.2.3 describes a NIZKPoK of Blum integer trapdoor of independent interest. For completeness, §A.2.3 briefly mentions the intuition for the NIZKP of GM BitComs of 0, derived directly from prior work.

A.2.1 Problems with non Blum integers

An IFC-based S2PC-with-Coms execution would be insecure if the used modulus was not a Blum integer. Since verification is trivial when knowing the factorization, the following paragraphs analyze the problem that may arise if a party in the role of selecting a Blum integer would be able to induce another party to use a non-Blum integer.

A Blum integer is congruent with 1 modulo 4 (and this ensures that −1 has class 0). Since this can be verified non-interactively, without knowing the factorization, the attention can be restricted to integers of this form. These are all odd integers that have an even number of (including none) prime factors congruent with 3 modulo 4 exponentiated to an odd exponent. In other words, −1 remains class 0 regardless of the number of prime factors that are congruent with 1 modulo 4, and of the prime powers that have an even exponent. The use of non-Blum integers of this form would raise security problems.

For example, things could go wrong with a modulus with exactly 3 prime factors, with at least one of them being congruent with 3 modulo 4 and such that the highest power of such a prime that would still divide the modulus would have an odd exponent. As toy examples, the integers 105 ($= 3 \times 5 \times 7$) and 1755 ($= 3^3 \times 5 \times 13$) have these properties. In the respective multiplicative groups, each quadratic residue has exactly eight square-roots, four of which in each possible *class*. Since the additive inverse of a square-root is also a square-root in the same class, the four square-roots in a certain class can be grouped into two pairs, each pair being a trivially correlated pair of square-roots. Any two square-roots not in the same pair are non-trivially correlated, i.e., the pair cannot be obtained except by being able to find some non-trivial factor of the modulus. This would raise at least two problems in the S2PC-with-Coms protocol:

- The 2-out-of-1 OT is not possible (though a 4-out-of-1 OT is), since for each square there are now 4 possible proper square-roots, 2 for each class.
- Assume that connectors of input wires of P_B would be correctly constructed, i.e., would be validated if selected for a *check* challenge. If they were selected for *evaluation*, a malicious P_A could still respond by disclosing multipliers with correct homomorphic properties that would nonetheless lead the known input BitComs into group-elements different from those that were used to derive the circuit input keys. The problem is that the squaring operation is no longer a permutation from the domain of proper square-roots in a certain class onto the set of squares, but rather a non-injective function.

Another problematic case is that of a modulus that once divided by the highest square factor does not retain any prime factor that is congruent with 3 modulo 4. Toy examples are integers 325 ($= 5^2 \times 13$) and 585 ($= 5 \times 3^2 \times 13$). In the respective multiplicative groups, each quadratic residue either has all square-roots with class 0 or all square-roots with class 1. This means that the respective 2-to-1 square BitCom scheme would become unconditionally binding, and a receiver knowing the factorization would be able (i.e., P_A), with the trapdoor, to decrypt the underlying bits — this would be a privacy problem.

Remark A.1 (General vs. specific Blum integers). From the point of view of correctness of a Blum integer, it does not matter which odd exponents are applied to each of the two prime factors. However, in terms of hiding the trapdoor, it is to the best interest of the party selecting the modulus that it makes the integer as hardest to factor as possible. Thus, it is preferable to construct it as a simple product of two primes (congruent with 3 modulo 4),

rather than having any prime power with exponent higher than 2. This makes more difficult the factorization of the integer by methods whose complexity increases with the size of the smallest factor. In other words, a proof for general Blum integers is sufficient and adequate, even though there are known proofs (more costly) to show that a certain integer is a Blum integer composed simply as the product of two primes [TLL03].

A.2.2 NIZKP of Blum integer correctness

If the factorization of a Blum integer modulus is known, then it is trivial to verify that it is a Blum integer. The challenge is in letting a prover (P), knowing the factorization of a modulus, convince a verifier (V) that the proposed modulus is a Blum integer, but without disclosing anything, i.e., in a zero-knowledge fashion. A ZKP of Blum integer correctness was devised by van de Graaf and Peralta [vdGP88], based on the observation that a modulus is a Blum integer if and only if it is not a square, not the power of a prime, is congruent with 1 modulo 4, and every residue in class 0 (i.e., with Jacobi Symbol 1) has a square-root or a pseudo square-root of each class. (A *pseudo square-root* of an integer is intended to mean a square-root of the additive inverse of the integer.) The protocol is also a ZKPoK of the trapdoor if the simulator is able to impersonate the trusted source of randomness, when also impersonating V playing with a black-box malicious P.

In the original description [vdGP88], a "trusted source of randomness" provides a random vector of group elements in class 0 and a corresponding vector of challenge bits, i.e., the equivalent to Σ-commit and Σ-challenge elements. However, this dissertation gives preference for a setting where, even if there is a trusted source of randomness (e.g., a local CRS), it should be reusable across several sub-protocols. While a larger random string could be obtained via a simulatable two-party coin-flipping, this would require additional interaction, which is also undesirable. The protocol in Figure A.1 considers a different adaptation, for the purpose of a NIZKP of Blum integer correctness (it is no longer a NIZKPoK of the Blum integer trapdoor). A different adaptation into a NIZKPoK of trapdoor is included in A.2.3.

Intuition. The vector of random group elements that would be provided by a trusted source of randomness is instead obtained via an internally simulated two-party coin-flipping, with the help of a NPRO and an Equiv-Com scheme. The challenge bits that would also be provided by the trusted source of randomness are instead also obtained via the same NPRO

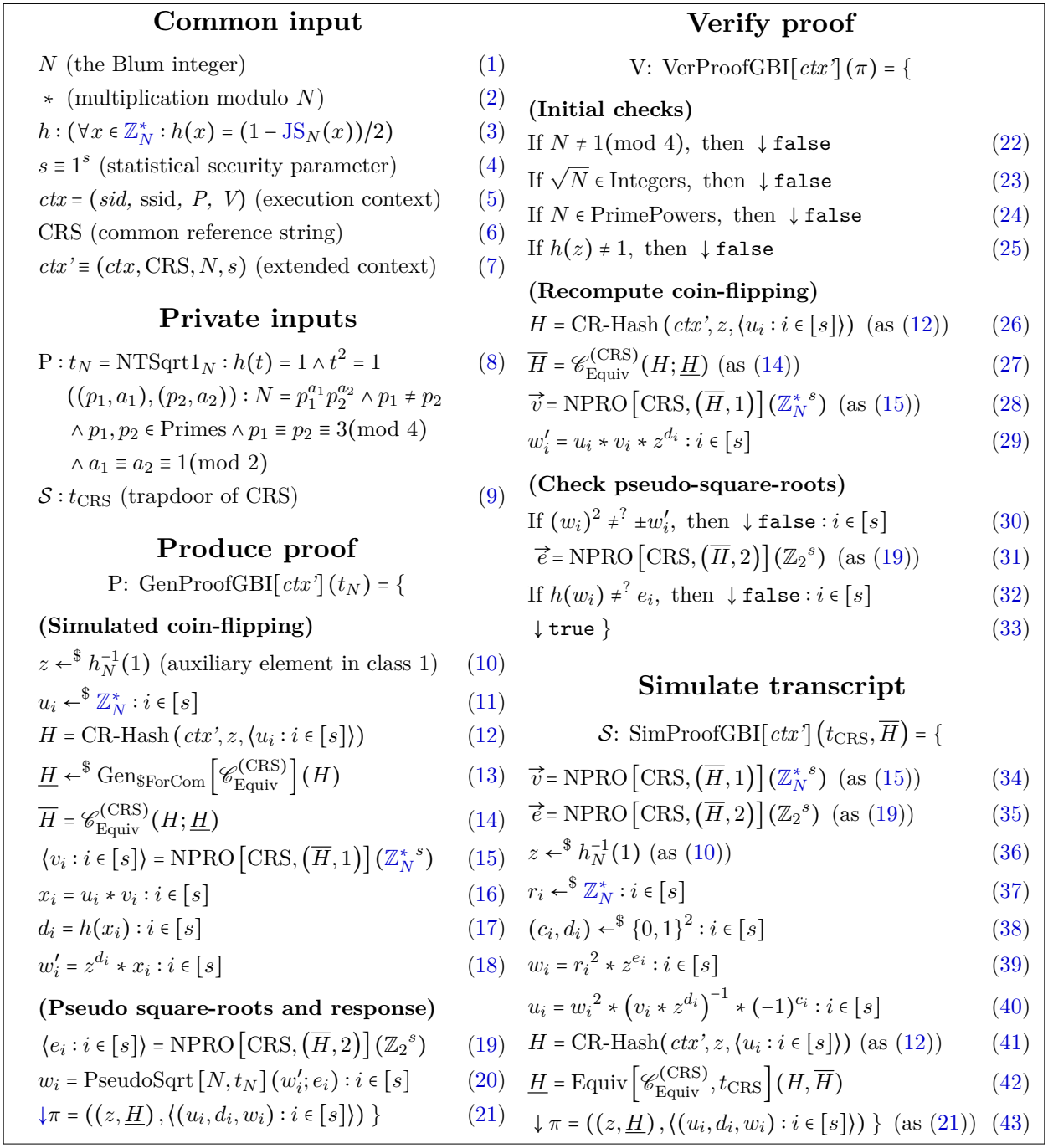
hashing, in time for P to equivocate the coin-flipping result to a vector of group elements for which it can provide the vector of challenged pseudo-square roots.

Setup.

- **Common input.** The generation and verification of proofs make sense in a context where the prover (P) and verifier (V) have a common input. Specifically, the proof of knowledge (of a trapdoor) is in respect to a known integer modulus (the Blum integer) (1). The actual prime factors and exponents are not known by V. The integer modulus defines a respective group operation, which is modular multiplication (2), and a respective Jacobi-Symbol-based XOR homomorphism (3). Each proof is performed in respect to a statistical security (soundness) parameter (4), and in a shared context composed of a session identifier, a sub-session identifier, and the identity of prover and verifier (5). The parties also have access to the same CRS, which must be local in order to ensure non-transferability of the proof (6). By definition, the extended context of the execution is obtained by concatenating to the original context the CRS, the Blum integer and the statistical security parameter (7).
- **Private inputs.** As private input, P knows the trapdoor of the Blum integer, defined as a non-trivially correlated square-root of 1 (8). In practice, the trapdoor is a knowledge equivalent of the integer factorization of the modulus, i.e., two primes with remainder 3 upon division by 4, and respective odd exponents. In respect to simulations, a simulator $\mathcal{S}$ impersonating a prover against a malicious V is able to define the CRS in a way that it knows a respective trapdoor t_{CRS} that will be used as an equivocation trapdoor for an Equiv-Com scheme (9)

Proof generation. The prover invokes an algorithm GenProofGBI, where "GBI" stands for "good Blum Integer," using as input the extended context *ctx'*, and the trapdoor t_N of the Blum integer N. The algorithm executes as follows:

- **Simulated coin-flipping.**
  P selects an auxiliary random group element z with class 1 (10). Actually, this could also be pre-computed as the shortest such element, if it can be obtained after a linear number of trials (e.g., 40) (e.g., trying consecutive numbers and computing its class) and later sent as part of the proof transcript. It cannot however be an element that would give any knowledge to V, e.g., it cannot be the non-trivially square-root of 1.



Common input

N (the Blum integer) (1)
$*$ (multiplication modulo N) (2)
$h : (\forall x \in \mathbb{Z}_N^* : h(x) = (1 - \mathrm{JS}_N(x))/2)$ (3)
$s \equiv 1^s$ (statistical security parameter) (4)
$ctx = (sid, \mathrm{ssid}, P, V)$ (execution context) (5)
CRS (common reference string) (6)
$ctx' \equiv (ctx, \mathrm{CRS}, N, s)$ (extended context) (7)

Private inputs

P : $t_N = \mathrm{NTSqrt1}_N : h(t) = 1 \wedge t^2 = 1$ (8)
$((p_1, a_1), (p_2, a_2)) : N = p_1^{a_1} p_2^{a_2} \wedge p_1 \neq p_2$
$\wedge\, p_1, p_2 \in \mathrm{Primes} \wedge p_1 \equiv p_2 \equiv 3 (\mathrm{mod}\ 4)$
$\wedge\, a_1 \equiv a_2 \equiv 1 (\mathrm{mod}\ 2)$
$\mathcal{S} : t_{\mathrm{CRS}}$ (trapdoor of CRS) (9)

Produce proof

P: $\mathrm{GenProofGBI}[ctx'](t_N) = \{$

(Simulated coin-flipping)

$z \leftarrow^{\$} h_N^{-1}(1)$ (auxiliary element in class 1) (10)
$u_i \leftarrow^{\$} \mathbb{Z}_N^* : i \in [s]$ (11)
$H = \text{CR-Hash}\,(ctx', z, \langle u_i : i \in [s] \rangle)$ (12)
$\underline{H} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}\left[\mathscr{C}_{\mathrm{Equiv}}^{(\mathrm{CRS})}\right](H)$ (13)
$\overline{H} = \mathscr{C}_{\mathrm{Equiv}}^{(\mathrm{CRS})}(H; \underline{H})$ (14)
$\langle v_i : i \in [s] \rangle = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 1\right)\right](\mathbb{Z}_N^{*\,s})$ (15)
$x_i = u_i * v_i : i \in [s]$ (16)
$d_i = h(x_i) : i \in [s]$ (17)
$w_i' = z^{d_i} * x_i : i \in [s]$ (18)

(Pseudo square-roots and response)

$\langle e_i : i \in [s] \rangle = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 2\right)\right](\mathbb{Z}_2{}^s)$ (19)
$w_i = \mathrm{PseudoSqrt}\,[N, t_N]\,(w_i'; e_i) : i \in [s]$ (20)
$\downarrow \pi = ((z, \underline{H}), \langle (u_i, d_i, w_i) : i \in [s] \rangle)\ \}$ (21)

Verify proof

V: $\mathrm{VerProofGBI}[ctx'](\pi) = \{$

(Initial checks)

If $N \neq 1 (\mathrm{mod}\ 4)$, then $\downarrow$ `false` (22)
If $\sqrt{N} \in$ Integers, then $\downarrow$ `false` (23)
If $N \in$ PrimePowers, then $\downarrow$ `false` (24)
If $h(z) \neq 1$, then $\downarrow$ `false` (25)

(Recompute coin-flipping)

$H = \text{CR-Hash}\,(ctx', z, \langle u_i : i \in [s] \rangle)$ (as (12)) (26)
$\overline{H} = \mathscr{C}_{\mathrm{Equiv}}^{(\mathrm{CRS})}(H; \underline{H})$ (as (14)) (27)
$\vec{v} = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 1\right)\right](\mathbb{Z}_N^{*\,s})$ (as (15)) (28)
$w_i' = u_i * v_i * z^{d_i} : i \in [s]$ (29)

(Check pseudo-square-roots)

If $(w_i)^2 \neq^{?} \pm w_i'$, then $\downarrow$ `false` $: i \in [s]$ (30)
$\vec{e} = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 2\right)\right](\mathbb{Z}_2{}^s)$ (as (19)) (31)
If $h(w_i) \neq^{?} e_i$, then $\downarrow$ `false` $: i \in [s]$ (32)
$\downarrow$ `true` $\}$ (33)

Simulate transcript

$\mathcal{S}$: $\mathrm{SimProofGBI}[ctx']\left(t_{\mathrm{CRS}}, \overline{H}\right) = \{$

$\vec{v} = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 1\right)\right](\mathbb{Z}_N^{*\,s})$ (as (15)) (34)
$\vec{e} = \mathrm{NPRO}\left[\mathrm{CRS}, \left(\overline{H}, 2\right)\right](\mathbb{Z}_2{}^s)$ (as (19)) (35)
$z \leftarrow^{\$} h_N^{-1}(1)$ (as (10)) (36)
$r_i \leftarrow^{\$} \mathbb{Z}_N^* : i \in [s]$ (37)
$(c_i, d_i) \leftarrow^{\$} \{0,1\}^2 : i \in [s]$ (38)
$w_i = {r_i}^2 * z^{e_i} : i \in [s]$ (39)
$u_i = {w_i}^2 * \left(v_i * z^{d_i}\right)^{-1} * (-1)^{c_i} : i \in [s]$ (40)
$H = \text{CR-Hash}(ctx', z, \langle u_i : i \in [s] \rangle)$ (as (12)) (41)
$\underline{H} = \mathrm{Equiv}\left[\mathscr{C}_{\mathrm{Equiv}}^{(\mathrm{CRS})}, t_{\mathrm{CRS}}\right](H, \overline{H})$ (42)
$\downarrow \pi = ((z, \underline{H}), \langle (u_i, d_i, w_i) : i \in [s] \rangle)\ \}$ (as (21)) (43)

Figure A.1: **NIZKP of Blum integer correctness ($\mathbf{NIZKP_{GBI}}$).**

P selects random group elements u_i (hereafter called offset elements) (11) in number equal to the statistical parameter. Then, P computes a CR-Hash H (denoted global hash) of the concatenation of the extended context ctx', the auxiliary element z and the vector of offsets u_i (12). P selects randomness $\underline{H}$ necessary for an Equiv-Com of the hash (13), and produces a respective commitment $\overline{H}$ (14). The public parameters of the Equiv-Com are

suitably based on the CRS, which defines a public parameter of the commitment scheme, whose respective equivocation-trapdoor can be influenced by the simulator.

P then uses the global NPRO, using as input the extended context *ctx* and the Equiv-Com of the hash, along with a suffix "1" (to distinguish from a subsequent use), to produce a new vector of complementary group elements v_i (15). For each position of the vector, P multiplies the initial offset element u_i with the complementary group element v_i (16), calculates the class d_i of the resulting product x_i (17), and if necessary applies a correction to obtain an element in class 0, by multiplying the auxiliary element z of class 1 (18). This procedure completes the internal coin-flipping simulation of group elements w_i of class 0.

- **Pseudo square-roots and response** P reuses the global NPRO, using as input the extended context *ctx* and the Equiv-Com of the hash, along with a new suffix "2" (to distinguish from the previous use), to produce a vector of challenge bits e_i (19). For each of the coin-flipped group elements w'_i, P computes a pseudo-square root with class equal to the respective obtained challenge bit e_i (20).

  As output, the proof transcript π (its version stripped away from contextual information) contains a pair composed of the auxiliary element z of class 1, the randomness $\underline{H}$ needed to reproduce the Equiv-Com $\overline{H}$ of the global hash H, and contains a vector of triplets, with each triplet containing an initial offset element u_i, the class d_i of the respective element obtained before corrections (this bit is not essential, but is helpful for V), and the pseudo square-root w_i of the respective final coin-flipped element w'_i (21).

Remark A.2 (On the use of a malleable Equiv-Com scheme). It is not a problem using a malleable Com-scheme, because the overall non-malleability of the NIZKP is already ensured by the NPRO input containing the extended context, and because the non-interactive nature of the NIZKP ensures that the Equiv-Com is fully disclosed (commitment, and opening) at once in the transcript, i.e., without any interleaved commitment or protocol taking place that could take advantage of the commitment before knowing the opening.

Remark A.3 (On more general notation for the Equiv-Com). For simplicity, the description assumes that the opening of the Equiv-Com scheme can be performed by simply revealing the committed element and the original randomness used to produce the commitment. This is applicable for example to Pederesen Coms and Blum BitComs, without jeopardizing the Equiv property, and allows the proof transcript to only have the randomness $\underline{H}$ as added

element in comparison with the remaining relevant group element. More generally, there may be Equiv schemes whose opening is performed in a different manner, namely sending the committed value H and then a probabilistic element $\underline{\underline{H}}$ (e.g., a NIZKP that it was the correct committed element) that is different from the original randomness $\underline{H}$ used to produce the commitment $\overline{H}$, and which cannot on its own be used to reproduce the commitment value. In that case the notation would change to additionally include in the transcript also the commitment value $\overline{H}$.

Proof verification.

- **Initial checks.** As initial checks, V directly checks the received modulus, namely that it is 1 modulo 4 (22), that it is not a square (23), and that it is not the power of a prime (24). V also checks that the received auxiliary element z has class 1 (25). If any of the previous checks fails, then V immediately rejects the transcript, outputting `false`.
- **Recompute coin-flipping.**
  Then, using the remaining information received in the proof transcript, V recomputes the global hash (26), recomputes the Equiv-Com c produced by P (27), uses the NPRO to obtain the vector of complementary group elements v_i (28), and uses them to calculate the final vector w'_i of coin-flipped elements w'_i with class 0 (29). Since the auxiliary element z has been verified as having class 1, it is sufficient to recompute the coin-flipped elements w'_i directly trusting the auxiliary bits d_i informed in the proof transcript — if one such bit was incorrect, then the resulting coin-flipped group element would have class 1 and would necessarily not have any pseudo-square-root, and thus the next step of the verification would necessarily fail.
- **Recompute coin-flipping.** V checks that the vector of random group elements is consistent with the received vector of pseudo-square-roots w_i, i.e., they the square of each received pseudo-square-root is equal to the random group element w_i or to its additive inverse (30). Then, V uses the NPRO to recompute the vector of challenge bits e_i (31), and uses them to check whether they correspond to the respective classes of the received vector of pseudo square-roots. (32). If any check fails, then V rejects the proof, outputting `false`; otherwise it accepts the proof, outputting `true` (33).

Proof simulation. A simulator can produce a proof transcript associated with any NPRO input, i.e., any pre-image that V would propose as PRNG seed in case this would be an

interactive Σ-protocol. Simulator starts by recomputing the vector of complementary group elements (i.e., the second contribution to the simulated coin-flipping) (34), and also the vector of challenge bit classes (35).

Then, $\mathcal{S}$ computes a random auxiliary element z (36), as a regular prover would. Then, $\mathcal{S}$ selects a vector of auxiliary random group elements r_i (37) and two vectors of random bits, one related to random classes d_i, the other related to random additive inverses c_i (38). Then, $\mathcal{S}$ uses the random auxiliary elements r_i and the fixed auxiliary element z to generate simulated random pseudo-square-roots w_i of the respective challenged classes e_i (39). From these elements, $\mathcal{S}$ can then calculate what need to be the respective offset elements u_i (40). Essentially, they are elements that once multiplied with the complementary group-elements v_i obtained from the NPRO, and eventually correcting with the auxiliary group element z, result in either the square or the additive inverse (depending on the randomly chose bit c_i) of the previously computed square root w_i. Then, $\mathcal{S}$ computes the Hash of the vector of offset group elements, also using as input prefix the extended context and the auxiliary element z (41). Finally, $\mathcal{S}$ uses its equivocation power, based on a trapdoor $trap_{\text{CRS}}$ associated with the Equiv-Com scheme whose public parameters are defined by the CRS, to generate an equivocated randomness $\underline{H}$ that when used in the Equiv-Com scheme to commit the global hash H would produce the commitment $\overline{H}$ that the simulator has as input of the simulation procedure (42). Finally, $\mathcal{S}$ outputs the transcript, based on the elements it calculated (43).

Analysis.

- **Security.** Completeness follows from the ability to produce pseudo-square-roots of any group element class 0 (modulo a Blum integer), when knowing the trapdoor (20). Soundness follows from P having to produce pseudo-square-roots of "random" class e_i for a sufficient large number s of "random" group elements. ZK follows from the equivocability of the Com scheme used to commit the global hash, which allows the simulator to induce any random-group elements and classes, and thus produce a proof transcript even if not knowing the trapdoor of the Blum integer, as detailed in the proof simulation procedure. Essentially, the simulator would equivocate each offset element u_i
- **Communication complexity.** See cell E6 in Table B.2. The transcript requires only two group elements (one square, and one square-root) and one bit (class) for each bit of statistical security, one auxiliary group element z, one "randomness" used to produce an equivocable commitment. For example, for Blum integers with 3,248 bits and for 96 bits

of statistical security (a short-term computational parameter in case the proof transcript is built during a protocol execution, after a new unpredictable extended context is known), the proof transcript would require about 79 KB, whereas for 128 bits of statistical security this would be adjusted to about 104 KB.

- **Computational complexity.** The most relevant group operations are: in GenProofGBI, 1 Equiv-Com (equivalent to one exponentiation), s square-roots (each roughly equivalent to one exponentiation modulo each prime factor); about $s+2$ Jacobi symbols (and about $1.5s$ multiplications); in VerProofGBI, 1 Equiv-Com (corresponding to one exponentiation), $s+1$ Jacobi symbols (and about $2.5s$ multiplications, besides checking that N is not a square and not a prime power).

A.2.3 NIZKPoK of Blum integer trapdoor

This subsection describes a new NIZKPoK of Blum integer trapdoor, i.e., non-interactive and not requiring rewinding to allow the simulator extract the trapdoor from an accepted transcript produced by a possibly malicious P. The main intuition is that, besides proving that an integer is correct, all that is required for extraction is to allow the simulator to learn two non-trivially correlated square-roots. Intuitively, this can be achieved by producing several squares, then using an Ext-Com scheme to commit individually one square-root of each class for each square, and then use a NPRO to choose which square-root to open. Based on the extractable properties, the simulator is able to open the pairs and then obtain the trapdoor from any pair of non-trivially correlated square-roots. Instead of appending this procedure to a NIZKP of correctness, it is more efficient to integrate both in the same protocol. In comparison with the previously described NIZKP-GBI, the main difference of the new NIZKPoK-BI-trapdoor is the computation of both square-roots of each square and making the respective Ext-Coms, and then making the challenge bits depend also on the Ext-Coms. The description is given in Figure A.2 and in the text below.

Setup. The setup and inputs are similar to the NIZKP of Blum integer correctness (44).

Proof generation.

- **Simulate coin-flipping.** P internally simulates a coin-flipping of a vector of random group elements in class 0, in number equal to the number σ of bits of intended statistical

Context and inputs

As in Fig. A.1 (steps 1–8) (44)

Produce proof

P: GenPokBItrap $[ctx']\,(t_N) = \{$

(Simulated coin-flipping)

$z \leftarrow^{\$} h_N^{-1}(1)$ (auxiliary element in class 1) (45)

$u_i \leftarrow^{\$} \mathbb{Z}_N^* : i \in [s]$ (46)

$H_1 = \text{CR-Hash}\,(ctx', z, \langle u_i : i \in [s]\rangle)$ (47)

$\underline{H}_1 \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}\right](H_1)$ (48)

$\overline{H}_1 = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H_1; \underline{H}_1)$ (49)

$\langle v_i : i \in [s]\rangle = \text{NPRO}\left[\text{CRS}, \overline{H}_1\right](\mathbb{Z}_N^{*\,s})$ (50)

$x_i = u_i * v_i : i \in [s]$ (51)

$d_i = h(x_i) : i \in [s]$ (52)

$w'_i = z^{d_i} * x_i : i \in [s]$ (53)

(Ext-Coms of pseudo-square-roots)

$w_{i,0} = \text{PseudoSqrt}\,[N, t_N]\,(w'_i; 0) : i \in [s]$ (54)

$w_{i,1} = w_{i,0} * t_N : i \in [s]$ (55)

$\underline{w}_{i,b} = \text{Gen}_{\$\text{ForCom}}[\mathscr{C}_{\text{Ext}}^{(\text{CRS})}](w_{i,b}) : i \in [s], b \in \mathbb{Z}_2$ (56)

$\overline{w}_{i,b} = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}\left(w_{i,b};\ \underline{w}_{i,b}\right) : i \in [s], b \in \mathbb{Z}_2$ (57)

$\vec{\overline{w}} = \langle(\overline{w}_{i,0}, \overline{w}_{i,1}) : i \in [s]\rangle$ (58)

(Bit-challenges and replies)

$H_2 = \text{CR-Hash}(H_1, \vec{\overline{w}})$ (59)

$\underline{H}_2 \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}](H_2)$ (60)

$\overline{H}_2 = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H_2; \underline{H}_2)$ (61)

$\vec{e} \equiv \langle e_i : i \in [s]\rangle = \text{NPRO}\left[\text{CRS}, \overline{H}_2\right](\mathbb{Z}_2{}^s)$ (62)

$w\vec{\overline{w}}w = \left\langle(w_{i,e_i}, \underline{w}_{i,e_i}, \overline{w}_{i,1-e_i}) : i \in [s]\right\rangle$ (63)

$\downarrow \pi = \left((z, \underline{H}_1, \underline{H}_2), \langle u_i : i \in [s]\rangle, \vec{e}, w\vec{\overline{w}}w\right)\ \}$ (64)

Verify proof

V: VerPokBItrap $[ctx']\,(\pi) = \{$

(Initial checks)

If $N \neq 1 (\text{mod } 4)$, then $\downarrow$ `false` (65)

If $\sqrt{N} \in$ Integers, then $\downarrow$ `false` (66)

If $N \in$ PrimePowers, then $\downarrow$ `false` (67)

If $h(z) \neq 1$, then $\downarrow$ `false` (68)

(Recompute coin-flipping)

$H_1 = \text{CR-Hash}\,(ctx', z, \langle u_i : i \in [s]\rangle)$ (as (47)) (69)

$\overline{H}_1 = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H_1; \underline{H}_1)$ (as (49)) (70)

$\langle v_i : i \in [s]\rangle = \text{NPRO}[\text{CRS}, \overline{H}_1](\mathbb{Z}_N^{*\,s})$ (as (50)) (71)

$w'_i = u_i * v_i * z^{d_i} : i \in [s]$ (72)

(Recompute and check challenge bits)

$\overline{w}_{i,e_i} = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}\left(w_{i,e_i}, \underline{w}_{i,e_i}\right) : i \in [s]$ (73)

$\vec{\overline{w}} = \left\langle \overline{w}_{i,b} : i \in [s], b \in \{0,1\}\right\rangle$ (74)

$H_2 = \text{CR-Hash}\left(H_1, \vec{\overline{w}}\right)$ (as (59)) (75)

$\overline{H}_2 = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H_2; \underline{H}_2)$ (as (61)) (76)

If $\vec{e} \neq \text{NPRO}[\text{CRS}, \overline{H}_2](\mathbb{Z}_2{}^s)$, then $\downarrow$ `false` (77)

(Check pseudo-square-roots)

If $(w_{i,e_i})^2 \neq \pm w'_i$, then $\downarrow$ `false` $: i \in [s]$ (78)

If $h(w_{i,e_i}) \neq e_i$, then $\downarrow$ `false` $: i \in [s]$ (79)

$\downarrow$ `true` $\}$ (80)

Simulate transcript

$\mathcal{S}$: SimPokBItrap$[ctx']\left(t_{\text{CRS}}, \overline{H}_1, \overline{H}_2\right) = \{$

(Offsets and $ for 1st Equiv-Com)

$\vec{v} = \text{NPRO}\left[\text{CRS}, \overline{H}_1\right](\mathbb{Z}_N^{*\,s})$ (as (50)) (81)

$\vec{e} = \text{NPRO}\left[\text{CRS}, \overline{H}_2\right](\mathbb{Z}_2{}^s)$ (as (62)) (82)

$z \leftarrow^{\$} h_N^{-1}(1)$ (as (45)) (83)

$r_i \leftarrow^{\$} \mathbb{Z}_N^* : i \in [s]$ (84)

$(c_i, d_i) \leftarrow^{\$} \{0,1\}^2 : i \in [s]$ (85)

$w_i \equiv w_{i,0} = w_{i,1} = {r_i}^2 * z^{e_i} : i \in [s]$ (86)

$u_i = {w_i}^2 * \left(v_i * z^{d_i}\right)^{-1} * (-1)^{c_i} : i \in [s]$ (87)

$H_1 = \text{CR-Hash}(ctx', z, \langle u_i : i \in [s]\rangle)$ (as (47)) (88)

$\underline{H}_1 = \text{Equiv}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}, t_{\text{CRS}}\right](H_1, \overline{H}_1)$ (89)

(Ext-Coms and $ for 2nd Equiv-Com)

$\underline{w}_{i,b} = \text{Gen}_{\$\text{ForCom}}[\mathscr{C}_{\text{Ext}}^{(\text{CRS})}](w_{i,b}) : i \in [s], b \in \mathbb{Z}_2$ (90)

$\overline{w}_{i,b} = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}\left(w_{i,b};\ \underline{w}_{i,b}\right) : i \in [s], b \in \mathbb{Z}_2$ (91)

$\vec{\overline{w}} = \langle(\overline{w}_{i,0}, \overline{w}_{i,1}) : i \in [s]\rangle$ (92)

$H_2 = \text{CR-Hash}(H_1, \vec{\overline{w}})$ (93)

$\underline{H}_2 = \text{Equiv}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}, t_{\text{CRS}}\right](H_2, \overline{H}_2)$ (94)

$\downarrow \pi = \left((z, \underline{H}_1, \underline{H}_2), \vec{u}, \vec{e}, w\vec{\overline{w}}w\right)\ \}$ (as (64)) (95)

Figure A.2: **NIZKPoK of Blum integer trapdoor (NIZKPoK$_{\text{BI-trap}}$).**

security, as follows: P samples a random auxiliary element z in class 1 (45), then samples a random vector of random group elements u_i in class 0 (hereafter denotes offsets) (46), then computes a collision resistant hash H_1 of the concatenation of the extended context *ctx'*, the auxiliary element z, and the vector of random offset elements u_i (47). P samples randomness $\underline{H}_1$ suitable to produce an Equiv-Com (48) and then uses it to produce a respective commitment $\overline{H}_1$ of the hash (49). Then, using as input the CRS and the Equiv-Com $\overline{H}_1$, P uses the NPRO to generate a vector of random complementary group element v_i (the second contribution to the coin-flipping) (50). P calculates the component-wise multiplication of the two vectors of group elements, thus obtaining in each component a new group element x_i (51). P computes the class d_i of each resulting elements (52), and if the class is 1 then it further multiplies it by the auxiliary group element z of class 1 (53), thus obtaining a new element of class 0, here denoted as a coin-flipped group element, being a square or the additive inverse of a square.

- **Ext-Coms of pseudo square-roots.** For each coin-flipped group element, P computes a pseudo-square-root $w_{i,0}$ of class 0 (54). Then, P multiplies the result by the trapdoor (a non-trivial square-root of 1) to obtain a pseudo square-root $w_{i,1}$ of class 1 (55). For each pseudo square-root $w_{i,b}$, P samples randomness $\underline{w}_{i,b}$ suitable for an Ext-Com scheme $\mathscr{C}_{\text{Ext}}$ (56), and then uses it to produce a Ext-Com $\overline{w}_{i,b}$ of the square-root (57). The Ext-Coms can thus be organized in a vector $\vec{\overline{w}}$ of pairs of Ext-Coms (58).
- **Bit-challenges and replies.** P computes a collision-resistant hash H_2 of the concatenation of the initial CR hash H_1 and the vector $\vec{\overline{w}}$ of pairs of Ext-Coms (59), then samples randomness $\underline{H}_2$ suitable to produce an Equiv-Com of the hash (60) and then uses it to indeed produce a respective Equiv-Com $\overline{H}_2$ of the hash (61). P makes another call to the NPRO, using as input the CRS and the Equiv-Com of the hash, to obtain a vector of bit challenges e_i (62). As reply to the challenges, P produces a vector $w\vec{w}w$ that contains, for each index of the challenge (i.e., for each bit of statistical security), the pseudo-square-root w_{i,e_i} and respective randomness $\underline{w}_{i,e_i}$ associated with the challenged bit e_i, and the Ext-Com $\overline{w}_{i,1-e_i}$ associated with the complementary bit (63).

  Finally, the procedure outputs the proof transcript π, as a tuple containing: the auxiliary element z of class 1, the two randomnesses $(\underline{H}_1, \underline{H}_2)$ used as input in the NPROs, the vector $\vec{u}$ of offset elements, the vector $\vec{e}$ of challenge bits, and the vector $w\vec{w}w$ of triplets that containing one Ext-Com and one opening for each challenge index. (64).

Proof verification.

- **Initial NI checks by V.** As initial checks, V checks that the integer N is equal to 1 modulo 4 (65), that it is not a square (66), that it is not a power of a prime (67), and that the auxiliary element z is indeed of class 1 (68).
- **Recompute coin-flipping.** V uses the received elements to reproduce the first collision-resistant hash H_1 (69). Then, using the received randomness $\underline{H}_1$, V recomputes the same Equiv-Com $\overline{H}_1$ of the hash (70). V then uses the NPRO, using as input the CRS and the Equiv-Com of the hash, to recompute the vector of complementary group elements v_i (71), by directly multiplying the offset elements u_i contained in the proof transcript π, the complementary elements v_i obtained from the NPRO, and the class correction by means of multiplying by the auxiliary element z whenever the informed class d_i of the previous product was 1 (72).
- **Recompute and check challenge bits.** V uses the receive coin-flipped elements w_{i,e_i} (just one per challenge index) and respective randomnesses $\underline{w}_{i,e_i}$ of Ext-Coms to recompute the respective Ext-Coms $\overline{w}_{i,e_i}$ (73). At this point, V is able to reconstruct the vector $\vec{\overline{w}}$ of pairs of Ext-Coms of pseudo-square-roots of the coin-flipped group-elements (74). V uses said vector to recompute the second collision-resistant hash H_2, using as prefix of the pre-image also the previous CR hash H_1 (75). Using the respective received randomness $\underline{H}_2$, V then recomputes the respective Equiv-Com $\overline{H}_2$ of the second hash (76). V then verifies that the received vector $\vec{e_i}$ of bit challenges e_i is equal to the one that would be obtained from the NPRO applied to the concatenation of the CRS and the Equiv-Com of the second hash (77).
- **Check pseudo square-roots.** For each challenge bit e_i, V checks that the square of the respective tentative square-root w_{i,e_i} is equal to the respective coin-flipped element w'_i or its additive square (78), and checks that its class is equal to the challenge bit (79). If some verification fails, then the procedure returns `false`, otherwise it outputs `true` (80).

Proof simulation. A simulator can produce a proof transcript associated with any NPRO inputs, i.e., any pair of pre-images (Equiv-Coms) that V would propose as PRNG seed in case this would be an interactive Σ-protocol.

- **Offsets and randomness for first Equiv-Com.** The first part of the simulation is very similar to the simulation of the NIZKP of Blum integer correctness (steps (34–42)). From the first Equiv-Com $\overline{H}_1$, $\mathcal{S}$ uses the NPRO to regenerate the vector of comple-

mentary group-elements v_i (81). From the second Equiv-Com $\overline{H}_2$, $\mathcal{S}$ regenerates the vector of challenge bits e_i (82). (In the NIZKP-GBI there was instead a single Equiv-Com, and the differentiation in the input of the NPRO was made via a character suffix.)

$\mathcal{S}$ the samples a random auxiliary element z of class 1 (83), then samples a vector of random auxiliary group elements r_i (84), as well as a vector of random class bits d_i and sign bits c_i (85). Then, as in the NIZKP, $\mathcal{S}$ uses the sampled randomness to produce in advance the needed pseudo-square roots w_i (86) However, as an augmentation to the NIZKP casae, in this NZKPoK simulation the value calculated for each challenge index is duplicated to fill the variable associated with each possible bit challenge $w_{i,b}$. (Actually, any syntactically valid value could be used for the non-selected class.) Then, the calculated pseudo suqare-root for each challenge index is used to calculate the respective offset group element u_i that is needed for the coin-flipping to induce the needed group elements w'_i (87). $\mathcal{S}$ computes the CR-Hash H_1 of the concatenation of the extended context *ctx'* with the auxiliary element z class 1 and the vector $\vec{u}$ of offset group elements (88) Finally, $\mathcal{S}$ uses the trapdoor t_{CRS} of the CRS to calculate the randomness $\underline{H}_1$ needed to equivocate the first Equiv-Com $\overline{H}_1$ to open to the calculated Hash H_1 (89).

- **Ext-Coms and randomness for second Equiv-Com.** The second part of the simulation, specific to this NIZKPoK, concerns the regeneration of the Ext-Coms of pseudo square-roots. For each (of two) coin-flipped elements $w_{i,b}$ associated with each challenge index, $\mathcal{S}$ generates randomness $\underline{w}_{i,b}$ suitable for an Ex-Com (90) and then uses it to produce a respective Ext-Com $\overline{w}_{i,b}$ (91). $\mathcal{S}$ produces a vector $\vec{\overline{w}}$ of pairs of Ext-Coms (92). Then, $\mathcal{S}$ concatenates the CRS and the vector and calculates its CR-Hash (denoted the second hash H_2) (93). Finally, $\mathcal{S}$ uses the trapdoor t_{CRS} of the CRS to calculate the randomness $\underline{H}_2$ needed to produce from the second hash the second Equiv-Com $\overline{H}_2$ used as input of the simulation algorithm. (94). The algorithm then outputs as simulated proof transcript π the appropriate tuple of elements that have been calculated (95), with the same structure as in an honest proof.

Analysis.

- **Security.** Completeness follows again from the ability to produce pseudo square-roots of any class from any group element class 0 (modulo a Blum integer), when knowing the trapdoor. Knowledge-extraction follows from the extractability of the pairs of Ext-Coms of pseudo-square roots of different class, with the respective statistical soundness being as

in the NIZKP. Computational zero-knowledge is reducible to the hiding property of the Ext-Coms (i.e., the ones that would actually not be committing to a pseudo-square-root) — all the rest is reducible to the equivocability of the Equiv-Coms that enable a simulator to induce the needed result of the coin-flipping of group elements.

- **Communication complexity.** See cell E2 in Table B.2. The transcript is composed on an initial triplet composed of one auxiliary group element and two randomnesses of an Equiv-Com, followed by a vector of group elements, a vector of bit challenges and a vector of triplets, where each triplet has one group element, one randomness for an Ext-Com and one Ext-Com. If using the defined Blum-based BitString Equiv-Com scheme (§B.4.1.2), each respective randomness requires one group element (a 2^{256}-th root) (see row 10 of Table B.1). If using OAEP-RSA as Ext-Com scheme, each respective randomness requires 256 bits, and each Ext-Com of a group element requires two group elements (see row 9 of Table B.1). Overall, for 3,248-bit Blum integers and statistical security of 96 bits the NIZKPoK transcript requires about 160 KB, whereas for a statistical security of 128 bits it requires about 212 KB.
- **Computational complexity.** The most relevant group operations are: in GenPokBItrap, s square-roots, 2 Equiv-Coms (each costing 1 exponentiation), $2s$ Ext-Coms (each costing 1 exponentiation), $s+2$ Jacobi Symbols (and about $2.5s$ multiplications); in VerPokBItrap, 2 Equiv-Coms, s Ext-Coms and $s+1$ Jacobi symbols (and about 2.5 multiplications, besides checking that N is not a square and not a prime power).

A.2.4 NIZKP of GM BitComs of 0

Informal description. Proving that a vector of BitComs is a vector of commitments to 0 corresponds to proving in ZK that it is a vector of squares (i.e., quadratic residues). Here it is sufficient to consider a NIZK-transformation of the original Feige-Fiat-Shamir ZKPoK (with rewinding) of a vector of square-roots [FFS88], along with a minor adaptation concerning additive inverses. The protocol takes advantage of the homomorphic properties of GM BitComs, allowing a statistical verification of square-roots of products of random subsets of the squares (and masked with an additional random square factor). The NIZKP is produced as follows: P produces new random auxiliary GM BitComs of 0, in number equal to the intended number of bits of statistical security; then, P calculates the CR-Hash of the concatenation of the triplet composed of the extended execution context, the original vector

of BitComs and the auxiliary vector of BitComs, and uses it as input of a NPRO to generate for each auxiliary square a pseudo-random subset of positions of the original vector; then, P calculates the products of the respective subsets of known square-roots, multiplying also the square-root of the respective auxiliary BitCom of 0; the proof transcript is the triplet composed of the vector of auxiliary BitComs of 0, the randomness of the Equiv-Com of the hash (but not the hash), and the vector of square-roots of masked products of random subsets. If there is at least one original BitCom for which P does not know a respective square-root (or one square-root does not exist), then it will fail to provide a square-root with probability overwhelming in the number of auxiliary challenges.

Communication. As described, the protocol requires group elements in number twice the statistical parameter, and one randomness of an Equiv-Com (e.g., also one group element, if using the generalized Blum BitString Com scheme).

A.3 NIZK sub-protocols for DLC

This section describes DLC-related NIZKPs and NIZKPoKs. For initial intuition, §A.3.1 describes the classic (interactive) HVZKPoK of discrete log (DL), of same DL, of Pedersen representation (Rep) and of same Rep. §A.3.2 describes a NIZKP for proving correctness of an ElGamal BitCom, which is essentially an OR of "same DL" NIZKP. §A.3.3 describes a full-fledged NIZKPoK of DL (simulatale without rewinding). §A.3.4 describes a NIZKPoK of ElGamal opening. §A.3.5 describes a NIZKP of vectors of ElGamal Coms of 0.

Initial definitions. The protocols in this section relate to proofs about elements from a cyclic-group $(\mathbb{G}, *)$ represented in multiplicative notation, with non-interactively and efficiently computable order $q = \#(\mathbb{G})$, where the DDH assumption holds, and for which two random generators (g_0, g_1) (i.e., such that $\langle g_0 \rangle = \langle g_1 \rangle = \mathbb{G}$) can be encoded in a common reference string (CRS) provided by a trusted setup, such that no party knows the discrete log of one generator base the other.

For ΣHVZK protocols (§A.3.1, §A.3.2), a standalone setting is considered, e.g., without concern for session and subsession identifiers. Then, when considering transformations of Σ-protocols to a non-interactive and possibly concurrent execution setting §A.3.3, the

determination of the Σ-challenge makes use of the context *ctx* information and the common reference string (CRS), in order to ensure non-malleability.

A.3.1 Sigma HVZKPoKs (DL, same DL, Rep, same Rep)

A classical problem in the area of ZK protocols is that of proving knowledge of a discrete log α of an element g_1 base some fixed generator g_0, in a cyclic group of prime order q. This has traditionally been considered in a setting of simulation with rewinding. An early protocol based on a cut-and-choose approach [CEvdGP87] was devised by Chaum et al. There, the parties engage in several Sigma-type interactions, possibly in parallel, with each challenge being binary, thus achieving soundness overwhelming in the number of challenges, namely exponentially small error probability. A much more efficient protocol, also of Sigma type, was later devised by Schnorr [Sch91]. There, a single Sigma-challenge (a random exponent) is sufficient to ensure overwhelming soundness, and so the needed communication (in the HVZK version) is reduced to a single group element (for the Sigma-commit) and two exponents (one for Sigma-challenge and one for Sigma-response).

Several (stand-alone) Sigma HV-ZKPoK protocols are described with succinct notation in Figure A.3, for proving knowledge of a DL and several generalizations. In Schnorr protocol (Figure A.3a), the proof relates to a *principal* element A that can be obtained by raising the base generator g to the power of a secret exponent α (96). The base generator and the principal value are public, i.e., known by both P and V (97), but the secret exponent is known only by P (98). The protocol starts with a Σ-commit phase, where the prover selects a random exponent k (99), and uses it to send to V a respective random exponentiation K of the base generator g (100). Then, V selects a random challenge c (in the exponent space) and sends it to P (101). P then applies a linear transformation to the secret exponent α, using the challenge c and the random exponent k as coefficients of first and zeroth order, respectively, and sending the result z to V (102). Finally, V accepts the proof as correct if and only if it successfully verifies that the Σ-commit value K multiplied by the principal value A raised to the power of the challenge c is equal to the base generator g raised to the power of the response value z (103).

Generalizations to Schnorr protocol can be obtained for example for: proving knowledge of a same DL α of two principal values (A_1, A_2) with respect to two respective base generators (g_1, g_2) (Figure A.3b); proving knowledge of a representation (α_0, α_1) of a principal value

Elements relation: $A = g^{\alpha}$ (96)
Initial input:
$\mathrm{P}, \mathrm{V} : (g, A)$ (common input) (97)
$\mathrm{P} : \alpha = \mathrm{DL}_g(A)$ (private) (98)
Sigma interaction:
$\mathrm{P} : k \leftarrow^{\$} \mathbb{Z}_q$ (rand for Σ-commit) (99)
$\mathrm{P} \rightarrow \mathrm{V} : K = g^k$ (Σ-commit) (100)
$\mathrm{V} \rightarrow \mathrm{P} : c \leftarrow^{\$} \mathbb{Z}_q$ (Σ-challenge) (101)
$\mathrm{P} \rightarrow \mathrm{V} : z = \alpha \cdot c + k \pmod q$ (Σ-response) (102)
$\mathrm{V} : A^c * K =^{?} g^z$ (Σ-verify) (103)

(a) Sigma-HVZKPoK of DL (Schnorr [Sch91])

Elements relation: $A_1 = {g_1}^{\alpha} \wedge A_2 = {g_2}^{\alpha}$ (104)
Initial input:
$\mathrm{P}, \mathrm{V} : (g_1, A_1), (g_2, A_2)$ (common input) (105)
$P : \alpha = \mathrm{DL}_{g_1}(A_1) = \mathrm{DL}_{g_2}(A_2)$ (private) (106)
Sigma interaction:
$\mathrm{P} : k \leftarrow^{\$} [q]$ (rand for Σ-commit) (107)
$\mathrm{P} \rightarrow \mathrm{V} : (K_1, K_2) = ({g_1}^k, {g_2}^k)$ (Σ-commit) (108)
$\mathrm{V} \rightarrow \mathrm{P} : c \leftarrow^{\$} [q]$ (Σ-challenge) (109)
$\mathrm{P} \rightarrow \mathrm{V} : z = \alpha \cdot c + k \pmod q$ (Σ-response) (110)
$\mathrm{V} : \wedge_{i \in \{1,2\}} A_i^c * K_i =^{?} {g_i}^z$ (Σ-verify) (111)

(b) Sigma-HVZKPoK of same DL ([CP92])

Elements relation: $A = g_0^{\alpha_0} g_1^{\alpha_1}$ (112)
Initial input:
$\mathrm{P}, \mathrm{V} : (g_0, g_1), A$ (common input) (113)
$\mathrm{P} : (\alpha_0, \alpha_1) \left(\in \mathrm{Rep}_{g_0, g_1}(A)\right)$ (private input) (114)
Sigma interaction:
$\mathrm{P} : k_0, k_1 \leftarrow^{\$} \mathbb{Z}_q$ (rand for Σ-commit) (115)
$\mathrm{P} \rightarrow \mathrm{V} : K = {g_0}^{k_0} {g_1}^{k_1}$ (Σ-commit) (116)
$\mathrm{V} \rightarrow \mathrm{P} : c \leftarrow^{\$} \mathbb{Z}_q$ (Σ-challenge) (117)
$\mathrm{P} : z_0 = \alpha_0 \cdot c + k_0 \pmod q$ (118)
$\mathrm{P} : z_1 = \alpha_1 \cdot c + k_1 \pmod q$ (119)
$\mathrm{P} \rightarrow \mathrm{V} : (z_0, z_1)$ (Σ-response) (120)
$\mathrm{V} : A^c * K =^{?} {g_0}^{z_0} {g_1}^{z_1}$ (Σ-verify) (121)

(c) Sigma-HVZKPoK of Representation ([Oka93])

Elements relation: $A_i = {g_{i,0}}^{\alpha_0} {g_{i,1}}^{\alpha_1}$ (122)
Initial input:
$\mathrm{P}, \mathrm{V} : ((g_{i,0}, g_{i,1}), A_i : i \in \{1,2\})$ (common) (123)
$P : (\alpha_0, \alpha_1) \left(\in \mathrm{Rep}_{g_{i,0}, g_{i,1}}(A_i), \forall i \in \{1,2\}\right)$ (124)
Sigma interaction:
$\mathrm{P} : k_0, k_1 \leftarrow^{\$} [q]$ (rand for Σ-commit) (125)
$\mathrm{P} \rightarrow \mathrm{V} : \left(K_i \equiv {g_{i,0}}^{k_0} {g_{i,1}}^{k_1} : i \in \{1,2\}\right)$ (126)
$\mathrm{V} \rightarrow \mathrm{P} : c \leftarrow^{\$} [q]$ (Σ-challenge) (127)
$\mathrm{P} \rightarrow \mathrm{V} : z_0 = \alpha_0 \cdot c + k_0 \pmod q$ (128)
$\mathrm{P} \rightarrow \mathrm{V} : z_1 = \alpha_1 \cdot c + k_1 \pmod q$ (129)
$\mathrm{P} \rightarrow \mathrm{V} : (z_0, z_1)$ (Σ-response) (130)
$\mathrm{V} : \wedge_{i \in \{1,2\}} {A_i}^c * K_i =^{?} {g_{i,0}}^{z_0} {g_{i,1}}^{z_1}$ (Σ-verify) (131)

(d) Sigma-HVZKPok of same Representation

Figure A.3: **Several Sigma-HVZKPoKs (DL, same DL, Rep, same Rep).** Each protocol allows extraction if rewinding is allowed in the simulation, but not otherwise.

A, with respect to two base generators (g_0, g_1) (Figure A.3c); proving knowledge of a same representation (α_0, α_1) of two principal values (A_1, A_2) with respect to two pairs $((g_{1,0}, g_{1,1}), (g_{2,0}, g_{2,1}))$ of base generators (Figure A.3d). A unified analysis of this type of ZKPoKs of a preimage of a group homomorphism can be found for example in [Mau09].

In a setting of simulation-with-rewinding the protocols are indeed ZKPoKs, as the simulator is able to use rewinding to obtain two valid Σ-responses (z, z') to two different Σ-challenges (c, c') for the same Σ-commit message K and from there extract the secret discrete logarithms. If rewinding is not allowed, a more sophisticated protocol is required for extraction (e.g., see §A.3.3 for a (NI)ZKPoK of DL).

A.3.2 NIZKP of good ElGamal BitCom

This subsection describes a NIZKP of good ElGamal BitCom ($\text{NIZKP}_{\text{GEB}}$) (i.e., of correctly constructed BitCom), based on the OR-proof technique (§A.1.6) applied to a HVZKP of Same DL [CP92], and using a transformation into NIZKP based on an Equiv-Com and a NPRO. Per DDH assumption, an ElGamal Com semantically hides the committed value, to anyone not knowing the DL between the two generators. Thus, the goal of the NIZKP is to prove that the committed value is either a 0 or a 1. The prover (P) may (but does not need to) know the secret DL (between the two generators) because the commitment is unconditionally binding (i.e., considering an opening phase corresponding to reveal the randomness). The NIZKP procedures are described with succinct notation in Figure A.4. A textual description follows.

Common input.

- **Proof instance.** As common input, both parties know two generators $(g_{\text{A}}, g_{\text{B}})$ (132) of the same group of known order q (133). The NIZKP instance is in respect to a known ElGamal Com C, which is a pair $(G_A, G_{B,0})$ of group elements (134). The first component is the first generator g_{A} raised to the power of a secret "randomness" r, and the second element is a multi-exponentiation composed of the product of two powers, the first being the first generator g_{A} raised to the power of the committed value b, and the other being the second generator g_{B} raised to the power of the "randomness" r. From the original BitCom it is possible to produce an adjusted pair composed of the original first group element G_A and an adjusted second group element $G_{B,1}$ obtained after dividing the second original component $G_{B,0}$ by the first generator g_{A} (135). The assertion that the original pair is an ElGamal BitCom is equivalent to the assertion that either the original pair is a BitCom of 0 OR the adjusted pair is a BitCom of 0. Since a BitCom of 0 is a pair of powers with the same exponent, base the respective pair of generators, the overall NIZKP is reducible to an OR disjunction of assertions of same DL.
- **Context and auxiliary input.** The NIZKP is contextualized by session and sub-session identifiers, and a specific prover and verifier (136) (i.e., when intending non-malleability). The parties have access to a common reference string (CRS), assumed to be local when intending non-transferability of proofs (137). The *extended context* is defined as a tuple containing the session context *ctx*, the CRS and the proof instance, i.e., the pair $(g_{\text{A}}, g_{\text{B}})$ of generators and the triplet of principal group elements $(G_A, G_{B,0}, G_{B,1})$ (138).

Common input

(Proof instance)

(g_A, g_B) (pair of generators) (132)

$q = \#(\langle g_A \rangle) = \#(\langle g_B \rangle)$ (group order) (133)

$C = (G_A, G_{B,0})$ (ElGamal BitCom) (134)

$(A = {g_A}^r, G_{B,0} = {g_A}^b {g_B}^r)$

$G_{B,1} \equiv G_{B,0}/g_A$ (135)

Context and auxiliary input

$ctx = (sid,$ ssid, $P,\ V)$ (context) (136)

CRS (common reference string) (137)

$ctx' \equiv (ctx, \text{CRS}, (g_A, g_B), C)$ (138)

Private inputs

$\text{P} : x = \text{DL}_{g_A}(G_A)$ (139)

$\text{P} : b \in \{0,1\} : r = \text{DL}_{g_B}(G_{B,b})$ (140)

$\mathcal{S} : t_{\text{CRS}}$ (trapdoor of CRS) (141)

Produce proof

P: GenProofGEB $[ctx'](x, b) = \{$

(Simulate incorrect case ($b' \equiv 1 - b$))

$c_{b'} \leftarrow^{\$} \mathbb{Z}_q$ (Σ-challenge-b') (142)

$z_{b'} \leftarrow^{\$} \mathbb{Z}_q$ (Σ-response-b') (143)

For $j \in \{A, B\}$: (Σ-commit-b')

$K_{b',j} = {G_{j,b'}}^{-c_{b'}} * {g_j}^{z_{b'}}$ (144)

(Prepare correct case (b))

$k_b \leftarrow^{\$} \mathbb{Z}_q$ (rand for Σ-commit-b) (145)

$K_{b,j} = g_j^{k_b} : j \in \{A, B\}$ (Σ-commit-b) (146)

(Challenges)

$\vec{K} \equiv ((K_{0,A}, K_{0,B}), (K_{1,A}, K_{1,B}))$ (147)

$H = \text{CR-Hash}(ctx', \vec{K})$ (148)

$\underline{H} \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}\right](H)$ (149)

$\overline{H} = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H; \underline{H})$ (150)

$c = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q)$ (151)

$c_b = c - c_{b'} \pmod q$ (Σ-challenge-b) (152)

(Responses)

$z_b = r \cdot c_b + k_b \pmod q$ (Σ-response–b) (153)

$\pi = (\underline{H}, \vec{K}, c_0, (z_0, z_1))\ \}$ (154)

Verify proof

P: VerProofGEB $[ctx'](\pi) = \{$

$H = \text{CR-Hash}(ctx', \vec{K})$ (155)

$\overline{H} = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(H; \underline{H})$ (156)

$c = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q)$ (157)

$c_1 = c - c_0 \pmod q$ (158)

If $\bigvee_{i \in \{0,1\}, j \in \{A,B\}} K_{i,j} * {G_{B,i}}^{c_i} \neq {g_j}^{z_i}$, (159)

then $\downarrow$`false`, else `true` $\}$ (160)

Simulate proof

P: SimProofGEB $[ctx'](t_{\text{CRS}}, \overline{H}) = \{$

(Simulate both cases)

$c = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q)$ (161)

$c_0 \leftarrow^{\$} \mathbb{Z}_q$ (Σ-challenge-0) (162)

$c_1 = c - c_0 \pmod q$ (Σ-challenge-1) (163)

$z_b \leftarrow^{\$} \mathbb{Z}_q : b \in \mathbb{Z}_2$ (Σ-response-b) (164)

For $b \in \mathbb{Z}_2, j \in \{A, B\}$: (Σ-commit)

$K_{b,j} = {B_b}^{-c_b} * {g_j}^{z_b} : b \in \mathbb{Z}_2$ (165)

(Equivocate randomness)

$\vec{K} \equiv ((K_{0,A}, K_{0,B}), (K_{1,A}, K_{1,B}))$ (166)

$H = \text{CR-Hash}(ctx', \vec{K})$ (167)

$\underline{H} = \text{Equiv}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}, t_{\text{CRS}}\right](H, \overline{H})$ (168)

$\pi = (\underline{H}, \vec{K}, c_0, (z_0, z_1))\ \}$ (169)

Figure A.4: **NIZKP of Good ElGamal BitCom ($\text{NIZKP}_{\text{GEB}}$).** It is a NIZKP of the inclusive disjunction (OR) of two "Same DL" assertions sharing the first component. The construction derives from the HVZKP of same DL [CP92], the OR proof technique [CDS94] and the transformation to NIZK based on an Equiv-Com and random oracle [Dam00, FS87, Lin15].

Private inputs. As private input, P knows the "randomness" x (i.e., the DL) of the first ElGamal component G_A (139), and knows which second component (the original $G_{B,0}$ or the adjusted $G_{B,1}$) has the same DL base the second generator g_B (140). A proof simulator has access to the CRS trapdoor, so that it is able to equivocate the opening of Equiv-Coms (141).

Proof generation. The proof generation algorithm uses as input the extended context *ctx'* and the private DL α and bit b known by P.

- **Simulate incorrect case ($b' = 1 - b$).** The protocol starts with P locally simulating the three elements of a "Σ-HVZKP of same DL" transcript, in respect to the incorrect instance of the disjunction. The transcript is simulated by selecting a random Σ-challenge $c_{b'}$ (142) and a random Σ-response $z_{b'}$ (143) and then using them to solve the HVZKPoK verification equation to yield the consistent pair $(K_{A,b'}, K_{B,b'})$ of Σ-commit values (144). (The elements of an actual honestly generated transcript would be generated in different order, starting with the Σ-commit value).
- **Prepare correct case (b).** Then, for the correct instance the algorithm proceeds as in a regular Σ-protocol, sampling a random exponent k_b (145) and then producing a respective pair $(K_{b,A}, K_{b,B})$ of Σ-commit elements as the result of raising each generator to the power of the random exponent (146).
- **Challenges.** At this point, the four Σ-commit elements (two for each instance of the disjunction) are organized into a respective vector $\vec{k}$ (147). P calculates the CR-Hash H of the tuple composed of the extended context *ctx'* and the vector $\vec{K}$ of Σ-commit elements (148). Then, P samples randomness $\underline{H}$ suitable to produce an Equiv-Com of the hash (149) and uses it to produce a respective Equiv-Com $\overline{H}$ (150). The Σ-challenge c (what in an interactive HVZKP would be decided by V) is obtained as the NPRO output when using as input the CRS and the Equiv-Com $\overline{H}$ (151). Since this is an OR proof, P uses the received challenge as the offset between the two needed Σ-challenges (one for each instance of the OR disjunction). Since this is an OR proof, the Σ-challenge c_b for the correct instance is defined as the difference between the global challenge c and the challenge $c_{b'}$ already fixed for the incorrect instance (152).
- **Responses.** P calculates the Σ-response for the correct instance as it would for a HVZKPoK of same DL, namely as a linear function of the private randomness exponent x, using the Σ-challenge c_b and the Σ-commit pre-image k_b as first and zero-th order coefficients, respectively (153). Finally, the algorithm outputs the proof transcript as

the tuple π containing the randomness $\underline{H}$ of the Equiv-Com of the hash (but not the hash), the vector $\vec{K}$ of Σ-commit elements the σ-challenge c_0 of the first (but not of the second) instance (regardless of whether it was the correct or the incorrect one), and the pair (z_0, z_1) of Σ-responses (154).

Proof verification. V recomputes the CR-Hash H of the extended context *ctx'* followed by the vector $\vec{K}$ of Σ-commit elements (155). Then, using the randomness element $\underline{H}$ present in the proof transcript π, V recomputes the respective Equiv-Com $\overline{H}$ of the hash (156). V recomputes the global challenge c, when using as input the CRS and the Equiv-Com $\overline{H}$ of the hash (157). Based on the Σ-challenge c_0 present in the proof transcript π, V recomputes the Σ-challenge c_1 of the second instance (158). Finally, V executes for both instances the Σ verification procedure of the HVZKP of Same DL, i.e., for each instance it checks whether each generator g_j raised to the power of the Σ-response element z_i associated with each instance equals the respective Σ-commit element $K_{j,i}$ multiplied by second group element $G_{j,i}$ of the instance raised to the power of the respective Σ-challenge c_i (159) If any verification fails, then the algorithm outputs `false`; otherwise it outputs `true` (160).

Proof simulation. A simulator $\mathcal{S}$ knowing the CRS trapdoor t_{CRS} is able to simulate proof transcripts for any Equiv-Com $\overline{H}$ that would be used as NPRO seed by V in an eventual interactive version of the protocol.

- **Simulate both cases.** $\mathcal{S}$ starts by generating the global challenge c as the NPRO output when using as input the CRS and the Equiv-Com of the hash (161). Then, $\mathcal{S}$ samples a random Σ-challenge c_0 for the first instance (162) and then calculates the corresponding Σ-challenge c_1 for the second instance, as the difference between the global challenge c and the first challenge c_0 (163). Then, for both instances, $\mathcal{S}$ samples a random Σ-response z_b (164) and then calculates the corresponding Σ-commit pair $K_{b,j}$ of group elements (165).
- **Equivocate randomness.** $\mathcal{S}$ organizes the two pairs of Σ-commit values into a respective vector $\vec{K}$ (166), then computes the CR-Hash of tuple composed of the extended context *ctx'* and the vector $\vec{K}$ of Σ-commit elements (167), and then uses the CRS trapdoor to calculate the randomness $\underline{H}$ suitable for opening the computed hash H from the Equiv-Com present in the input of the simulation procedure (168). Finally, the algorithm outputs as proof transcript the tuple composed of the calculated randomness $\underline{H}$, the vector $\vec{K}$ of Σ-commit elements, the Σ-challenge c_0 of the first instance, and the pair (z_0, z_1) of Σ-responses (169).

Analysis.

- **Security.** Completeness follows from the ability of an honest prover (i.e., knowing a consistent opening of the ElGamal BitCom) to simulate the incorrect case and to answer any challenge for the correct case. Soundness follows directly from the soundness of the HVZKP of Same DL and of the OR technique — a malicious prover not knowing a suitable DL is able to construct a valid Σ-response for at most one instance, but not for both. Zero knowledge follows from the ability of a simulator, knowledgeable of the CRS trapdoor, to produce false transcripts indistinguishable from correct ones, by equivocating the Equiv-Com used as pre-image of the NPRO (168).
- **Communication complexity.** See cell E8 in Table B.2. The transcript contains 1 Equiv-Com randomness, 4 group elements and 3 exponents. For 128 bits of computational security, using an exponent domain with 256 bits, and group elements with 264 bits in ECC and 3,248 bits in FFC, the transcript is about 0.26 kB in ECC and 1.75 kB in FFC.
- **Computational complexity.** In terms of multiplicative operations, it requires: 1 multiplication in the exponent space, and 6 exponentiations and 2 group-multiplication of group elements from P; 8 exponentiations and 4 group-multiplications of group elements from V.

A.3.3 NIZKPoK of discrete log

Intuition. A direct transformation of the Σ-challenge of Schnorr protocol into a non-interactive protocol would remove the ability of extraction without rewinding. Also the described idea of leveraging the special soundness with a cut-and-choose does not apply directly to the Schnorr Σ-challenge because of exponential size of challenges. Instead, the description below uses a cut-and-choose approach suited to binary challenges, related to the ability of extracting a DL from any two distinct Pedersen representations of the same group element. The NIZKPoK procedures are defined in Figure A.5. A textual description follows.

Common input. The generation and verification of proofs make sense in a context where the prover (P) and verifier (V) share some common input. The NIZKPoK of a DL is in respect to a pair (g_0, g_1) of generators of the same cyclic group (170). The group order q is assumed known and verifiable (171). There is a computational security parameter 1^κ (172) used in respect to PRNGs, and a statistical security parameter σ that defines the soundness goal of each proof (173). There are cut-and-choose parameters (C&C) defining a minimum

and maximum number of challenges of type *evaluation* and of type *check* (174), consistent with the statistical security goal, in respect to the probability of guessing in advance an exact cut-and-choose partition selected at random (see Table 3.1 and Table 3.2). These numbers define the overall number s of challenges (175), as well as the set Π of all possible partitions of challenge indices (176). Each proof is generated and verified within the scope of an execution context, composed of a session identifier, a sub-session identifier, and the identity of prover and verifier (177). It is also based on a CRS, which must be local in order to ensure non-transferability of the proof (178). By definition, the extended context of the execution is the tuple that includes the original context, the CRS, the pair of generators, the specification of the set of possible partitions, and the security parameters (179).

Private inputs. As private input, P knows the discrete log of the second generator g_1 base the first generator g_0 (here denoted the global generator) (180). In respect to simulations, a simulator $\mathcal{S}$ is able to define the CRS in a way that it knows a respective trapdoor t_{CRS} (181). When impersonating a prover playing against a malicious V, the trapdoor allows equivocaing the opening of Equiv-Coms. When impersonating a verifier playing against a malicious P, the trapdoor allows extraction from Ext-Coms.

Proof generation. The prover invokes an algorithm GenPokDL, using as input the extended context *ctx'* and the trapdoor t. The algorithm executes as follows:

- **Group elements and Ext-Coms.** For each challenge index $j \in [n]$, P selects a random PRNG seed λ_j (e.g., 256 bits) (182), then uses it to pseudo-randomly generate an exponent u_j (183), then calculated a group element u'_j by raising the second generator g_1 (whose knowledge of DL base the global generator g_0 is being proven) to the power of exponent u_j (184). Then, P uses the random seed λ_j to pseudo-randomly generate "randomness" $\underline{u'_j}$ suitable for an Ext-Com of the exponent (185), and then uses it to produce a respective Ext-Com $\overline{u}'_j$ (186). P defines the complementary exponent v_j to be the modular sum (in the space of exponents) of the pseudo-random exponent u_j and the secret DL t (187).
- **Challenges and replies.** P organizes the group elements u'_j into a respective vector $\vec{u'}$ and the Ext-Coms $\overline{u}_j$ into a respective vector $\vec{\overline{u}}$ (188). P calculates a CR-Hash of the triplet composed of the extended context *ctx'* and the two vectors (189). Then, P generates random randomness $\underline{H}$ suitable for an Equiv-Com of the hash (190), and then produces a respective Equiv-Com $\overline{H}$ (191). P then determines the cut-and-choose partition

Common input

g_0, g_1 (two generators) (170)

$q = \#(\langle g_0 \rangle) = \#(\langle g_1 \rangle)$ (group order) (171)

$\kappa \equiv 1^\kappa$ (computational security parameter) (172)

$\sigma \equiv 1^\sigma$ (statistical security parameter) (173)

$\text{C\&C} = ((v_{\min}, v_{\max}), (e_{\min}, e_{\max}))$ (174)

$s \equiv v_{\min} + e_{\max} = e_{\min} + v_{\max}$ (175)

$(\sum_{j \in \{e_{\min}, e_{\max}\}} (s!/(j!(s-j)!)) \geq 2^s)$

$\Pi \equiv \text{PARTITIONS}[\text{C\&C}]([s])$ (176)

$ctx = (sid, \text{ssid}, P, V)$ (context) (177)

CRS (common reference string) (178)

$ctx' \equiv (ctx, \text{CRS}, (g_0, g_1), \text{C\&C}, (1^\sigma, 1^\kappa))$ (179)

Private inputs

$\text{P} : t = DL_{g_0}(g_1) : g_1 = {g_0}^t$ (180)

$\mathcal{S} : t_{\text{CRS}}$ (trapdoor of CRS) (181)

Produce proof

P: GenPokDL $[ctx'](t) = \{$

(Group elements and Ext-Coms)

For $j \in [s]$:

$\lambda_j \leftarrow^{\$} \text{Gen}_{\text{Seed}}(1^\kappa)$ (182)

$u_j = \text{PRGen}_{\text{Exp}}[\lambda_j](\mathbb{Z}_q)$ (183)

$u'_j = {g_1}^{u_j}$ (184)

$\underline{u}_j = \text{PRGen}_{\$\text{ForCom}}[\lambda_j]\left(\mathscr{C}^{(\text{CRS})}_{\text{Ext}}, u_j\right)$ (185)

$\overline{u}_j = \mathscr{C}^{(\text{CRS})}_{\text{Ext}}\left(u_j; \underline{u}_j\right)$ (186)

$v_j = u_j + t \pmod q$ (187)

(Challenges and replies)

$\vec{u'} = \left\langle u'_j : j \in [s] \right\rangle ; \ \vec{\overline{u}} = \langle \overline{u}_j : j \in [s] \rangle$ (188)

$H = \text{CR-Hash}\left(ctx', \vec{u'}, \vec{\overline{u}}\right)$ (189)

$\underline{H} \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}\left[\mathscr{C}^{(\text{CRS})}_{\text{Equiv}}\right](H)$ (190)

$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H})$ (191)

$(J_V, J_E) = \text{NPRO}\left[\text{CRS}, \overline{H}\right](\Pi)$ (192)

$R_V = \langle \lambda_j : j \in J_V \rangle$ (193)

$R_E = \langle (v_j, \overline{u}_j) : j \in J_E \rangle$ (194)

$\downarrow \pi = (\underline{H}, (J_V, J_E), (R_V, R_E)) \ \}$ (195)

Verify proof

V: VerPokDL $[ctx'](\pi) = \{$

(Group elements and Ext-Coms)

For $j \in J_V$:

$u_j = \text{PRGen}_{\text{Exp}}[\lambda_j](\mathbb{Z}_q)$ (196)

$u'_j = {g_1}^{u_j}$ (197)

$\underline{u}_j = \text{PRGen}_{\$\text{ForCom}}[\lambda_j]\left(\mathscr{C}^{(\text{CRS})}_{\text{Ext}}, u_j\right)$ (198)

$\overline{u}_j = \mathscr{C}^{(\text{CRS})}_{\text{Ext}}\left(u_j; \underline{u}_j\right)$ (199)

For $j \in J_E : u'_j = {g_0}^{v_j}$ (200)

(Challenges)

$\vec{u'} = \left\langle u'_j : j \in [s] \right\rangle ; \ \vec{\overline{u}} = \langle \overline{u}_j : j \in [s] \rangle$ (201)

$H = \text{CR-Hash}\left(ctx', \vec{u'}, \vec{\overline{u}}\right)$ (202)

$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H})$ (203)

If $(J_V, JE) \neq \text{NPRO}\left[\text{CRS}, \overline{H}\right](\Pi)$, (204)

then $\downarrow$ `false`, else $\downarrow$ `true` $\}$ (205)

Figure A.5: **NIZKPoK of discrete logarithm (NIZKPoK$_{\text{DL}}$).** (Simulator in Fig. A.6)

(J_V, J_E) (subsets of check and evaluation challenges) as the NPRO output, when having as input seed the CRS and the Equiv-Com of the global hash and requesting generation of a random possible partition (192). For the subset J_V (also interpreted as a vector) of check challenges, P prepares the vector R_V of respective seeds (193). For the subset J_E (also interpreted as a vector) of evaluation challenges, P prepares a respective vector R_E of pairs, each composed of the complementary exponent v_j and the Ext-Com $\overline{u}_j$ of

the pseudo-random exponent u_j (194). As proof transcript, P outputs the Equiv-Com $\overline{H}$ of the hash, the pair (J_V, J_E) of vectors of challenge indices and the respective pair (R_V, R_E) of vectors with responses (195).

Proof verification.

- **Group elements and Ext-Coms.** Upon receiving a proof transcript π, V invokes the algorithm VerPokDL, using as input the extended context *ctx'* and the proof transcript. Several obvious syntactical checks are left implicit (e.g., that the cut-and-choose partition has the correct parameters, that elements are in the expected domains). Based on the vector J_V of check indices and the respective vector R_V of replies, V pseudo-randomly recomputes the respective elements as in the proof-generation mechanism, namely: it uses the PRNG seed λ_j to regenerate an exponent u_j (196), then calculates the respective group elements u'_j, by raising the second generator g_1 to the power of the exponents (197), then pseudo-randomly generates randomness $\underline{u}_j$ suitable for an Ext-Com of the exponent (198), and recalculates the respective Ext-Com $\overline{u}_j$ (199).

  Based on the vector J_E of evaluation indices and the respective vector R_E of replies, V computes the respective group elements u'_j by raising the global generator g_0 to the power of the complementary exponent v_j (200).
- **Challenges.** V organizes the group elements u'_j and Ext-Coms $\overline{u}_j$ into respective vectors ($\vec{u'}$ and $\vec{\overline{u}}$) (201). V then computes the CR-Hash of the triplet composed of the extended context *ctx'* and the two vectors (202), similarly to how P had done in the proof generation. V then uses the Equiv-Com randomness $\underline{H}$ received in the proof transcript to produce a respective Equiv-Com $\overline{H}$ of the hash (203). Finally, V checks that the partition of challenges described in the proof transcript is equal to the one that can be obtained by the NPRO applied to the CRS and the Equiv-Com of the hash. If the check fails, then the algorithm outputs `false` (204); otherwise it outputs `true` (205).

Proof simulation. A simulator knowing the CRS trapdoor is able to generate an undetectably fake transcript for any Equiv-Com $\overline{H}$ value that in an interactive version of the protocol a possibly malicious verifier could send as a PRNG seed of the vector of challenges. The simulator is described with succinct notation in Figure A.6.

$\mathcal{S}$ invokes the algorithm SimPokDL, which starts by using the NPRO to compute the cut-and-choose partition (J_V, J_E) of challenges (206).

Simulate transcript

$\mathcal{S}$: SimPokDL$[ctx']\left(t_{\text{CRS}}, \overline{H}\right) = \{$

$(J_V, J_E) = \text{NPRO}\left[\text{CRS}, \overline{H}\right](\Pi)$ (206)

(Group elements and Ext-Coms)

For $j \in J_V$:

$\lambda_j \leftarrow^{\$} \text{Gen}_{\text{Seed}}(1^{2\kappa})$ (207)

$u_j = \text{PRGen}_{\text{Exp}}[\lambda_j](\mathbb{Z}_q)$ (208)

$u'_j = {g_1}^{u_j}$ (209)

$\underline{u}_j = \text{PRGen}_{\$\text{ForCom}}[\lambda_j]\left(\mathscr{C}_{\text{Ext}}^{(\text{CRS})}, u_j\right)$ (210)

$\overline{u}_j = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}\left(u_j; \underline{u}_j\right)$ (211)

$v_j \leftarrow^{\$} \mathbb{Z}_q : i \in J_E$ (212)

$u'_j = {g_0}^{v_j} : j \in J_E$ (213)

$\underline{u}_j = \text{Gen}_{\$\text{ForCom}}\left(\mathscr{C}_{\text{Ext}}^{(\text{CRS})}, 0\right) : j \in J_E$ (214)

$\overline{u}_j = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}\left(0; \underline{u}_j\right) : j \in J_E$ (215)

(Challenges and replies)

$\vec{u'} = \langle u'_j : j \in [s] \rangle$; $\vec{\overline{u}} = \langle \overline{u}_j : j \in [s] \rangle$ (216)

$H = \text{CR-Hash}\left(ctx', \vec{u'}, \vec{\overline{u}}\right)$ (217)

$\underline{H} = \text{Equiv}\left[\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}, t_{\text{CRS}}\right](H, \overline{H})$ (218)

$R_V = \langle \lambda_j : j \in J_V \rangle$ (219)

$R_E = \langle (v_j, \overline{u}_j) : j \in J_E \rangle$ (220)

$\downarrow \pi = (\underline{H}, (J_V, J_E), (R_V, R_E))$ } (221)

Figure A.6: **Simulator of NIZKPoK$_{\text{DL}}$ transcripts.** (Compare with Fig. A.5)

- **Challenges and replies.** For each check index $j \in J_V$, $\mathcal{S}$ builds elements as an honest prover would, namely generates a random seed λ_j (207), then uses it to pseudo-randomly generate an exponent u_j (208), and compute a respective group element u'_j by raising the second generator g_1 to the power of the pseudo-random exponent u_j (209), then uses again the random seed λ_j to pseudo-randomly generate "randomness" $\underline{u}_j$ suitable to produce an Ext-Com of the exponent (210) and then produces the respective Ext-Com $\overline{u}_j$ (211). For each evaluation index $j \in J_E$, $\mathcal{S}$ randomly generates a complementary exponent v_j (212), and calculates a respective group element u'_j by raising the global generator g_0 to the power of the exponent (213). $\mathcal{S}$ samples "randomness" $\underline{v}_j$ suitable for a respective Ext-Com of zero (214) and then produces a respective Ext-Com $\overline{v}_j$ (215).
- **Cut-and-choose challenges.** $\mathcal{S}$ organizes all group elements u'_j and all Ext-Coms $\overline{u}_j$ into respective vectors ($\vec{u'}$ and $\vec{\overline{u}}$) (216). $\mathcal{S}$ computes the CR-Hash of the triplet composed of the extended context ctx' and the two vectors (217). Finally, $\mathcal{S}$ makes use of the CRS trapdoor t_{CRS} to calculate the randomness $\underline{H}$ needed to equivocate the opening of the Equiv-Com $\overline{H}$ into the hash H just calculated (218). $\mathcal{S}$ produces the vector R_V of check instances as a vector with the random seeds λ_j of check instances (219), and organizes the vector R_E of evaluation instances as a vector with pairs composed of the complementary exponent v_j and the Ext-Com $\overline{u}_j$ of zero of each evaluation reply (220). The algorithm outputs as proof transcript π the triplet composed of the randomness $\underline{H}$ of the Equiv-Com, the cut-and-choose partition and the respective two vectors of replies (221).

Analysis.

- **Security.** Completeness follows from the ability to answer to any type of challenge, for each instance, when knowing the secret DL t. Specifically, this is equivalent to producing different Pedersen representations of the same group element (20). Soundness follows from the inability, when not knowing the secret DL t, to answer to different challenges for the same produced group element u'_j. Thus, a successful cheating would involve guessing the exact cut-and-choose partition in advance. Knowledge extraction follows from the ability of a simulator, knowing the CRS extraction trapdoor, to extract the committed pairs of exponents of each challenge index, and thus with overwhelming probability (same as soundness argument) obtaining at least one good pair of distinct representations of the same group element, which enables extracting the secret DL. ZK follows from: (i) the semantic hiding of the Ext-Coms, which prevents an observer from distinguishing the correct Ext-Coms of a real proof from the unopened Ext-Coms of 0 in the simulated transcript; and the (ii) equivocability of the Equiv-Com, which prevents distinguishing an equivocated randomness from a real randomness of the global hash.
- **Communication complexity.** See cell E4 in Table B.2. The transcript is composed of one randomness of an Equiv-Com, a cut-and-choose partition, a vector of random seeds, and a vector of pairs, each composed of a random exponent and an Ext-Com of an exponent. Assuming the random seed (e.g., 256 bits) is smaller then the pair of exponent and Ext-Com of exponent, it follows that the communication is maximum when the number e of evaluation instances is maximal and the number v of check instances is minimal. Consider Pedersen Equiv-Coms with randomness of 256 bits. Consider hybrid Ext-Coms based on ElGamal encryption, with each Ext-Com requiring two group elements plus the size of the committed elements. Consider FFC with 3,248-bit group elements, and ECC based on 264-bit group elements, both with 256-bit exponents. For 128 bits of statistical security, with $(s, e_{\min}, e_{\max}) = (129, 1, 64)$, the proof transcript requires about 58.2 kB for FFC and 10.5 kB for ECC. At the cost of more computation, the communication can decrease by appropriately increasing $v_{\min}$ and decreasing $e_{\max}$, e.g., $(s, v_{\min}, e_{\max}) = (220, 189, 32)$ would lead to NIZKPoK transcript sizes of 34.1 kB for FFC and 10.3 kB ECC.
- **Computational complexity.** In terms of computational complexity, the most relevant group operations are: in GenPokDL, s exponentiations, s Ext-Coms (each roughly equivalent to two exponentiations); 1 Equiv-Com (equivalent to two exponentiations), in VerPokDL, s exponentiations v Ext-Coms; 1 Equiv-Com.

Common input

Same as in Fig. A.5 (steps (170–179)) (222)

$C \equiv (C_1, C_2)$ (ElGamal Com) (223)

Private inputs

$\mathrm{P} : (\alpha_0, \alpha_1) : C = ({g_0}^{\alpha_0}, {g_0}^{\alpha_1} {g_1}^{\alpha_0})$ (224)

$\mathcal{S} : t_{\mathrm{CRS}}$ (trapdoor of CRS) (225)

Produce proof

P: GenPokElgOpen $[ctx'](t) = \{$

(Group elements and Ext-Coms)

For $j \in [s]$:

$\lambda_j \leftarrow^{\$} \mathrm{Gen}_{\mathrm{Seed}}(1^{2\kappa})$ (226)

$u_j \equiv (u_{j,0}, u_{j,1}) = \mathrm{PRGen}_{\mathrm{Exp}}[\lambda_j](\mathbb{Z}_q{}^2)$ (227)

$u'_j = (C_1 {g_0}^{u_0}, C_2 {g_0}^{u_1} {g_1}^{u_0})$ (228)

$\underline{u}_j = \mathrm{PRGen}_{\$\mathrm{ForCom}}[\lambda_j]\left(\mathscr{C}_{\mathrm{Ext}}^{(\mathrm{CRS})}, u_j\right)$ (229)

$\overline{u}_j = \mathscr{C}_{\mathrm{Ext}}^{(\mathrm{CRS})}\left(u_j; \underline{u}_j\right)$ (230)

$v_j = (\alpha_0 + u_{j,0}, \alpha_1 + u_{j,1}) \pmod q$ (231)

$v_j \equiv (v_{j,0}, v_{j,1})$ (232)

(Challenges and replies)

Steps (188–195) (233)

Verify proof

V: VerPokElgOpen $[ctx'](\pi) = \{$

(Group elements and Ext-Coms)

For $j \in J_V$: Steps (227–232) (234)

For $j \in J_E : u'_j = ({g_0}^{v_{j,0}}, {g_0}^{v_{j,1}} {g_1}^{v_{j,0}})$ (235)

(Challenges and replies)

Steps (201–205) (236)

Figure A.7: **NIZKPoK of ElGamal opening ($\mathbf{NIZKPoK_{ElgOpen}}$).**

A.3.4 NIZKPoK of ElGamal opening

A NIZKPoK of an ElGamal opening needs to allow extraction of both exponents — the "randomness" and the "committed value" — while ensuring that the extracted randomness is indeed used in both sides (i.e., it is not enough to perform a NIZKPoK of Pedersen representation of the second component). A protocol is described in Figure A.7. Given the similarity and same high-level structure, the description of the NIZKPoK of ElGamal opening makes reference to blocks of steps of the NIZKPoK of DL. The overall communication is less that the double of the NIZKPoK of DL; the number of exponentiations is triples.

Analysis.

- **Security.** The analysis is similar to the case of the NIZKPoK of DL, except that now there are two exponents, instead of just one, being committed with an Ext-Com. For any correctly generated evaluation instance, the knowledge-extractor simulator extracts the committed pair u_j of exponents and combines it with the complementary pair v_j of exponents and obtains the secret exponents. The ZK simulator is able to generate fake transcripts due to its ability to equivocate the opening of the Equiv-Com that served as pre-image to determine the challenges.

- **Communication complexity.** See cell E5 in Table B.2. The transcript is composed of one randomness of an Equiv-Com (e.g., 256 bits using Pedersen Coms), a cut-and-choose partition (e.g., s bits), a vector with about v PRNG seeds (e.g., $v_{\min}$ times 256 bits), and a vector with e pairs (e.g., $e_{\max}$), each composed of a pair of random exponents (e.g., 512 bits per pair) and an Ext-Com of a pair of exponents (e.g., $2 \times |\mathrm{GE}| + 512$ per pair, with $|\mathrm{GE}|$ equal to 3,248 bits for FFC and 264 for ECC). For 128 bits of statistical security, using cut-and-choose parameters $(s, e_{\min}, e_{\max})$ equal to $(129, 65, 64)$ this yields at most about 62,3 kB for FFC and 14.5 kB for ECC. In summary, in comparison with the NIZKPoK of DL, the communication less than doubled.
- **Computational complexity.** In terms of computational complexity, the most relevant group operations are: in GenPokElgOpen, $3s$ exponentiations, s Ext-Coms (each roughly equivalent to two exponentiations); 1 Equiv-Com (equivalent to two exponentiations), in VerPokElgOpen, $3s$ exponentiations v Ext-Coms; 1 Equiv-Com. In summary, in comparison with the NIZKPoK of DL, the number of exponentiations tripled.

A.3.5 NIZKP of vector of ElGamal BitComs of 0

This subsection considers how to produce a NIZK that a vector of ElGamal commitments is a vector of BitComs of 0. The prover is assumed to know the ElGamal openings. Interestingly, the proof can be statistically parallelized into the size of the NIZKP of a single ElGamal BitCom of 0, involving communication of only a unitary number of group elements.

The protocol is described with succinct notation in Figure A.8. An ElGamal BitCom C_i of 0 is a pair of group elements $(G_{A,i}, G_{B,i})$ with the same DL r_i, base the respective pair $(g_{\mathrm{A}}, g_{\mathrm{B}})$ of generators that parametrizes the ElGamal Com scheme.

Intuition. A NIZKP protocol for the Same DL language can be directly defined by a NI transformation of the HVZKPoK (with rewinding) of same DL [CP92]. An efficient parallelization can take advantage of the additive homomorphic properties of the scheme. The key observation is that the inner product of a vector of zeros with any vector of arbitrary coefficients is zero, whereas the probability that the inner product of any non-zero vector with a random vector of random coefficients is negligible small if the space of coefficients is exponentially large. In the Com space, the analogous observation is that the component-wise multiplication of two ElGamal BitComs of 0 is still a BitCom of 0. Upon defining the

Common input

(Proof instance)

(g_A, g_B) (pair of generators) (237)

$q = \#(\langle g_A \rangle) = \#(\langle g_B \rangle)$ (group order) (238)

$\vec{C} \equiv \langle C_i : i \in [n] \rangle$ (ElGamal Coms of 0) (239)

$C_{i,1} \equiv G_{A,i} (= {g_A}^{r_i}) : i \in [n]$ (240)

$C_{i,2} \equiv G_{B,i} (= {g_B}^{r_i}) : i \in [n]$ (241)

Context and auxiliary input

$ctx = (sid,$ ssid, $P,\ V)$ (context) (242)

CRS (common reference string) (243)

$ctx' \equiv (ctx, \text{CRS}, (g_A, g_B), \vec{C})$ (244)

Private inputs

$\text{P} : \vec{r} \equiv \langle r_i : i \in [n] \rangle : (g_A, g_B)^{\vec{r}} = \vec{C}$ (245)

$\mathcal{S} : t_{\text{CRS}}$ (trapdoor of CRS) (246)

Produce proof

P: GenProofGEB $[ctx'](\vec{r})$ = {

(Prepare Σ-commit)

$k \leftarrow^{\$} \mathbb{Z}_q$ (rand for Σ-commit) (247)

$K_j = {g_A}^k : j \in \{A, B\}$ (Σ-commit) (248)

(Challenges)

$H = \text{CR-Hash}(ctx', (K_A, K_B))$ (249)

$\underline{H} \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}\left[\mathscr{C}^{(\text{CRS})}_{\text{Equiv}}\right](H)$ (250)

$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H})$ (251)

$(\vec{\alpha}, c) = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q{}^n \times \mathbb{Z}_q)$ (252)

(Responses)

$r_0 = \sum_{i \in [n]} r_i \cdot \alpha_i \pmod q$ (253)

$z = r_0 \cdot c + k \pmod q$ (Σ-response) (254)

$\pi = (\underline{H}, (K_A, K_B), c, z)$ } (255)

Verify proof

P: VerProofGEB $[ctx'](\pi)$ = {

$H = \text{CR-Hash}(ctx', (K_A, K_B))$ (256)

$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H})$ (257)

$(\vec{\alpha}, c) = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q)$ (258)

$G_{j,0} = \prod_{i \in [n]} {G_{j,i}}^{\alpha_i}$ (259)

If $\vee_{j \in \{A,B\}}\ K_j * Gj, 0^c \neq {g_j}^z$, (260)

then $\downarrow$`false`, else `true` } (261)

Simulate proof

P: SimProofGEB $[ctx'](t_{\text{CRS}}, \overline{H})$ = {

(Simulate two instances)

$(\vec{\alpha}, c) = \text{NPRO}[\text{CRS}, \overline{H}](\mathbb{Z}_q)$ (262)

$c \leftarrow^{\$} \mathbb{Z}_q$ (Σ-challenge) (263)

$z \leftarrow^{\$} \mathbb{Z}_q$ (Σ-response) (264)

For $j \in \{A, B\}$:

$G_{j,0} = \prod_{i \in [n]} {G_{j,}}^{\alpha_i}$ (265)

$K_j = {G_{j,0}}^{-c} * {g_A}^z$ (Σ-commit) (266)

(Equivocate randomness)

$H = \text{CR-Hash}(ctx', (K_A, K_B))$ (267)

$\underline{H} = \text{Equiv}\left[\mathscr{C}^{(\text{CRS})}_{\text{Equiv}}, t_{\text{CRS}}\right](H, \overline{H})$ (268)

$\pi = (\underline{H}, (K_A, K_B), c, z)$ } (269)

Figure A.8: **NIZKP of vector of ElGamal BitComs of 0 ($\text{NIZKP}_{\text{EG-All-0s}}$).**

coefficients of such vector via a NPRO, the corresponding homomorphic operation between Coms corresponds to computing in the exponent space the weighed sum of exponents, getting the same exponent in both sides if all initial Coms were also pairs of elements with the same DL, or otherwise only with negligible probability if the coefficients were a solution to a linear equation.

Concrete complexity. The communication complexity is: one randomness for an Equiv-Com (e.g., one exponent), one ElGamal Com of 0 (i.e., two group elements) and two exponents.

A.4 NIZK sub-protocols for XOR-homomorphic BitComs

A.4.1 NIZKP of same committed bits across XOR-hom. BitCom schemes

A NIZKP of same committed bits across vectors of BitComs from different XOR-homomorphic schemes is described in Figure A.9. It may be useful for example when P_B uses a 2-out-of-1 OT scheme, where it needs to prove that his intermediate OT BitComs (using $\mathscr{B}_{OT}$), based on a Blum BitCom scheme with trapdoor known by P_A, are committing to the same bits as the outer BitComs (using $\mathscr{B}_B$), based on a GM BitCom scheme. The protocol is neither (and does not intend to be) a NIZKPoK of the randomness, nor of the committed bits (i.e., when rewinding is not allowed).

Setup.

- **BitComs.** The common input is composed of: a vector with public parameters of n XOR-homomorphic BitCom schemes (270), each defining a commitment function $\mathscr{B}_k$, and a group group operation $*_k$ (multiplicative notation). For simplicity all group operations are denoted with the same symbol ($*$), but in practice the operation may vary across schemes. Also for simplicity, the same symbol is used for the operation in randomness space. As common input, the proof is performed in respect to m vectors, each with ℓ BitComs $x'_{k,j}$, which can be overall organized in a matrix X' of BitComs (271). The BitCom scheme may change across vectors, but in each vector all BitComs are based on the same BitCom scheme (the homomorphisms will be performed within each vector).
- **Other parameters.** Each NIZKP is produced in respect to a statistical security parameter σ (272). The proof has an underlying cut-and-choose structure defined by parameters that define the minimum and maximum number of check and evaluation instances (273). The overall number s of challenges is equal to the sum of the number v check and number e of evaluation instances (274). The parameters are assumed consistent with the statistical security goal, and they define the set Π of possible cut-and-choose partitions (275). Each NIZP transcript is also produced for a particular execution context, considering a session identifier, a sub-session identifier, and the identity of the prover and verifier (276), and

Common input

(Parameters of m BitCom schemes)

$$\vec{\mathscr{B}} = \left\langle (\mathscr{B}_k, *^{(k)} \equiv *) : k \in [m] \right\rangle \quad (270)$$

(m vectors of ℓ BitComs)

$$X' = \left\langle x'_{k,i} = \mathscr{B}_k(b_i; x_{k,i}) : i \in [\ell], k \in [m] \right\rangle \quad (271)$$

(Other parameters)

$$\sigma = 1^\sigma \text{ (statistical security parameter)} \quad (272)$$

$$\text{C\&C} = ((v_{\min}, v_{\max}), (e_{\min}, e_{\max})) \quad (273)$$

$$s \equiv v_{\min} + e_{\max} = e_{\min} + v_{\max} \quad (274)$$

$$\left(\sum_{j \in \{e_{\min}, e_{\max}\}} (s!/(j!(s-j)!)) \geq 2^s\right)$$

$$\Pi \equiv \text{PARTITIONS}[\text{C\&C}]([s]) \quad (275)$$

$$ctx = (sid, \text{ssid}, P, V) \text{ (context)} \quad (276)$$

$$\text{CRS (common reference string)} \quad (277)$$

$$ctx' \equiv (ctx, \text{CRS}, \vec{\mathscr{B}}, \vec{x}) \quad (278)$$

Private inputs

$$\text{P} : \vec{b} \equiv \langle b_i \in \{0,1\} : i \in [\ell] \rangle \text{ (bits)} \quad (279)$$

$$\text{P} : X \equiv \left\langle x_{k,i} : i \in [\ell], k \in [m] \right\rangle \text{ (rands)} \quad (280)$$

$$\mathcal{S} : t_{\text{CRS}} \text{ (trapdoor of CRS)} \quad (281)$$

Produce proof

$$\text{P: GenProofSameComBits}[ctx']\left(\vec{b}, X\right) = \{$$

(Commit bit-masks)

For $j \in [s]$:

$$\lambda_j \leftarrow^{\$} \text{Gen}_{\text{Seed}}(1^\kappa) \quad (282)$$

$$d_j = \text{PRGen}_{\text{BitString}}[\lambda_j](s) \text{ (mask)} \quad (283)$$

For $j \in [s], k \in [m]$:

$$y_{k,j} = \text{PRGen}_{\$\text{ForCom}}[\lambda_j](\mathscr{B}_k, d_j) \quad (284)$$

$$y'_{k,j} = \mathscr{B}_k[d_j, y_{k,j}] \quad (285)$$

$$Y' = \left\langle y'_{k,j} : k \in [m], j \in [s] \right\rangle \quad (286)$$

(Challenges)

$$H = \text{CR-Hash}\left(ctx', Y'\right) \quad (287)$$

$$\underline{H} \leftarrow^{\$} \text{Gen}_{\$\text{ForCom}}\left[\mathscr{C}^{(\text{CRS})}_{\text{Equiv}}\right](H) \quad (288)$$

$$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H}) \quad (289)$$

$$(\lambda_0, (J_V, J_E)) = \text{NPRO}[\text{CRS}, \overline{H}](\{0,1\}^{\kappa_{\text{PRG}}}, \Pi) \quad (290)$$

(Evaluation responses)

For $j \in J_E$:

$$\vec{a_j} = \text{PRGen}_{\text{BitString}}[\lambda_0][j]\left(\mathbb{Z}_2{}^\ell\right) \quad (291)$$

$$e_j = d_j \oplus \left(\oplus_{i \in [\ell]} b_i \cdot a_{j,i}\right) \quad (292)$$

$$z_{k,j} = y_j * \left(*_{i \in [\ell]} x_i{}^{a_{j,i}}\right) : k \in [m] \quad (293)$$

$$\vec{z_j} = \left\langle z_{j,k} : k \in [m] \right\rangle \quad (294)$$

(Check responses)

$$R_V = \langle \lambda_j : j \in J_V \rangle \quad (295)$$

$$R_E = \left\langle (e_j, \vec{z_j}) : j \in J_E \right\rangle \quad (296)$$

$$\downarrow \pi = (\underline{H}, \lambda_0, (J_V, J_E), (R_V, R_E)) \; \} \quad (297)$$

Verify proof

$$\text{P: VerProofSameComBits}\,[ctx'](\pi) = \{$$

For $j \in J_V$:

$$d_j = \text{PRGen}_{\text{BitString}}[\lambda_j](s) \text{ (mask)} \quad (298)$$

For $j \in J_V, k \in [m]$:

$$y_{k,j} = \text{PRGen}_{\$\text{ForCom}}[\lambda_j](\mathscr{B}_k, d_j) \quad (299)$$

$$y'_{k,j} = \mathscr{B}_k[d_j, y_{k,j}] \quad (300)$$

For $j \in J_E$:

$$z'_{k,j} = \mathscr{B}_k(e_j; z_{k,j}) : k \in [m] \quad (301)$$

$$\vec{a'_j} = \text{PRGen}_{\text{BitString}}[\lambda_0][j](\ell) \quad (302)$$

$$y'_{k,j} = z'_{k,j} * \left(*_{i \in [\ell]} x'_i{}^{a_{j,i}}\right)^{-1} : k \in [m] \quad (303)$$

(Recompute challenges)

$$Y' = \left\langle y'_{k,j} : k \in [m], j \in [s] \right\rangle \quad (304)$$

$$H = \text{CR-Hash}\left(ctx', Y'\right) \quad (305)$$

$$\overline{H} = \mathscr{C}^{(\text{CRS})}_{\text{Equiv}}(H; \underline{H}) \quad (306)$$

$$\text{If } (\lambda_0, (J_V, J_E)) \neq \text{NPRO}[\text{CRS}, \overline{H}](\{0,1\}^{\kappa_{\text{PRG}}}, \Pi) \quad (307)$$

$$\text{then } \downarrow \texttt{false}, \text{ else } \downarrow \texttt{true} \; \} \quad (308)$$

Figure A.9: **NIZKP of same committed bits across XOR-hom. BitCom schemes.**

the parties have access to a common reference string (CRS) (277). The extended context of the execution is defined to be the tuple composed of the context, the CRS, the vector of BitCom scheme parameters and the matrix of BitComs (278).

Private inputs. As private input, P knows the vector of committed bits b_i (equal across all vectors of BitComs) (279) and the randomnesses $x_{k,i}$ needed to open them from the BitComs $x'_{k,i}$ presented as common input (280). It is irrelevant whether or not the verifier (V) knows a trapdoor of one or several schemes. A simulator of transcripts has access to the CRS trapdoor t_{CRS}, which allows equivocation of a respective Equiv-Com (281).

Proof generation. P invokes algorithms GenProofSameComBits, using as input the extended context, the vector of committed bits and the matrix of randomnesses associated with the matrix of BitComs.

- **Commit bit-masks.** For each instance, P selects a random PRNG seed λ_j (282), then uses it to generate a random bit-mask d_j (283), to pseudo-randomly generate randomness suitable to commit the bit with each BitCom scheme (284), and then uses each randomness to produce each respective BitCom $y'_{k,j}$ of the bit mask (285). All the BitComs of bit-masks can now be organized into a matrix Y' of new BitComs (286), where the dimension of the number of BitComs in each original vector as now been replaced for the number of instances in the cut-and-choose.
- **Calculate challenges.** P computes the CR-Hash H of the pair composed of the extended context *ctx'* and the matrix Y' of bitmask BitComs (287). Then, prover generates randomness $\underline{H}$ suitable for an Equiv-Com of the hash (288) and then uses it to produce the Equiv-Com (289). With the CRS and the Equiv-Com of the hash as input, P obtains from the NPRO a new pseudo-random seed λ_0 and a pseudo-random cut-and-choose partition (J_V, JE) (290).
- **Evaluation responses.** For each evaluation instance, P uses the new random seed λ_0 (invariant across evaluation instances) to compute a random bit-string $\vec{a_j}$ that defines a pseudo-random subset of positions of components to combine in the original vectors of BitComs (291). Then, P computes the result e_j of XORing together the subset of committed bits, and then also XORing the bit mask d_i (292). Correspondingly, for each BitCom scheme, P performs in the randomness space the corresponding homomorphic group operations, obtaining a new randomness $z_{k,j}$ as the product of the subset of selected

randomnesses and then multiplies by the randomness $y_{k,j}$ associated with the bit mask (293) Each instance j has thus a corresponding vector $\vec{z_j}$ of randomnesses, with one component per BitCom scheme $\mathscr{B}_k$ (294).

- **Check responses.** P organizes the check responses as a vector R_V of the initial random seeds λ_j of check instances (295). It organizes the evaluation responses as a vector of pairs, each composed of the masked bit and the vector of masked randomnesses (296). Finally, the algorithm outputs a proof transcript π as a tuple containing the randomness of the Equiv-Com of the hash, the PRNG seed λ_0 and the cut-and-choose partition (J_V, J_E) determined by the NPRO, and the vector R_V of check responses and the vector R_E of evaluation responses (297).

Proof verification.

- **Check instances.** For each check instance ($j \in J_V$), V uses the random seed λ_j to pseudo-randomly regenerate the check instances (298), then uses it again to pseudo-randomly regenerate the "randomness" suitable to create a BitCom for each BitCom scheme (299), and then computes the respective BitComs (300).
- **Evaluation instances.** For each evaluation instance ($j \in J_E$), V uses the received group element $z_{k,j}$ (combined randomness) to generate a respective response BitCom $z'_{k,j}$ (301). Then, V uses the auxiliary seed λ_0 received in the proof transcript (invariant across instances), to pseudo-randomly regenerate the sub-set a_j of elements that should be combined in each vector (302). V uses the subset to recompute what should have been the masking BitComs $y'_{k,j}$ (303), basically by dividing each response BitCom $z'_{k,j}$ by the respective subset product of original BitComs.
- **Recompute challenges.** V organizes all known BitComs of subset combinations into a matrix Y' (304). Then, V recomputes the CR-hash H of the pair composed of the extended context and the matrix of BitCom maskings (305). V uses the randomness $\underline{H}$ received in the proof transcript π to produce the same Equiv-Com of hash that an honest P would have generated (306).

  Finally, V calls the NPRO, using as input the CRS and the Equiv-Com of the hash, and checks whether or not the output is the expected one, namely a pair composed of the same auxiliary PRG seed λ_0 and the same cut-and-choose partition (J_V, JE) as included in the proof transcript (307). If the check fails, then the algorithm outputs `false`; otherwise it outputs `true` (308).

Analysis.

- **Security.** Completeness follows directly from the ability of P to open any original BitCom, and from all BitComs committing to the same bit. Soundness follows from the cut-and-choose approach — if there is at least one BitCom that V cannot open, or at least on vector position for which P only knows different openings for two BitComs, then P cannot generate a correct transcript, except if it "guesses" in advance the cut-and-choose partition and the random subsets. Zero knowledge follows from the ability of generating fake transcripts when possessing the CRS trapdoor. $\mathcal{S}$ is able to learn the cut-and-choose partition and the subsets still in time to equivocate the BitComs of maskings. Thus, for check instances it simply produces correct elements (which do not require knowing any randomness of the original BitComs), for evaluation instances it produces as incorrect auxiliary BitCom $y'_{k,j}$ in a way that it enables providing a respective well formed answer $z_{k,j}$. (Note: extractability of the committed bits is trivial if at least one of the BitCom schemes is extractable — in such case this NIZKP becomes also a NIZKPoK of the committed bits.)
- **Communication complexity and optimizations.** The NIZKP transcript is a tuple composed of one randomness of an Equiv-Com, $v+1$ PRNG seeds, one cut-and-choose partition, e bits, $m \times e$ randomnesses of BitComs. For GM BitComs with randomnesses (square-roots) with 3,248 bits each, the transcript requires about 54 kB for 128 bits of statistical security using $(s, e_{\min}, e_{\max}) = (129, 1, 64)$, and about 40.6 kB for 96 bits of statistical security, using $(s, e_{\min}, e_{\max}) = (97, 1, 48)$. Due to the statistical verification, the communication complexity does not depend of the number ℓ of BitComs in each vector.

Appendix B

Low-level description of S2PC-with-Coms

This appendix provides a more low level description of the S2PC-with-Coms protocol. Section B.1 describes notation for Com schemes, for homomorphisms and for the parameters of the protocol. Section B.2 recalls the ideal protocol execution (using succinct notation) and gives a lower-level specification of the real protocol. Section B.3 describes the respective simulators, and analyzes how they induce in the ideal world a joint output distribution indistinguishable from the one in real executions. Section B.4 describes optimizations to connectors. Section B.5 provides further details about communication complexity.

B.1 Notation for protocol descriptions

The section starts with a description in Figure B.1 of notation useful to describe real Com schemes (§B.1.1). Then it describes in Figure B.2 the notation used for additive homomorphisms and pseudo-homomorphisms in the space of committed values and the space of randomness (§B.1.2). Finally, it describes in Figure B.3 the implicit primitives and some symbols (e.g., labels of Com schemes) used in the overall S2PC-with-Coms protocol (§B.1.3).

B.1.1 Notation of commitment schemes

Given the diverse use of commitment schemes throughout the dissertation, Table B.1 describes notation for the commit and open phases of real commitment schemes, for different types of interactivity and with variations suited to different properties, but leaving implicit the aspects

Commit phase:	**Open phase:**
$\mathrm{P_S} \leftrightarrow \mathrm{P_R}:$	$\mathrm{P_S} \leftrightarrow \mathrm{P_R}:$
$\mathscr{C}\left[\mathrm{P_S}(x) \leftarrow^{\$} (\underline{x}_\mathrm{S}, \overline{x}_\mathrm{S}); \mathrm{P_R} \leftarrow^{\$} (\underline{x}_\mathrm{R}, \overline{x}_\mathrm{R})\right]$ (309)	$\mathcal{O}\left[\mathrm{P_S}(x, \underline{x}_\mathrm{S}, \overline{x}_\mathrm{S}); \mathrm{P_R}(\underline{x}_\mathrm{R}, \overline{x}_\mathrm{R}) \leftarrow^{\$} (x, \underline{\underline{x}})\right]$ (310)

(a) **Interactive Com scheme**

Commit phase:	**Open phase:**
$\mathrm{P_S}: (\underline{x}, \overline{x}) \leftarrow^{\$} \mathscr{C}[x]$ (311)	$\mathrm{P_S} \to \mathrm{P_R}: \underline{\underline{x}} \leftarrow^{\$} \mathcal{O}(x, \underline{x}, \overline{x})$ (313)
$\mathrm{P_S} \to \mathrm{P_R}: \overline{x}$ (312)	$\mathrm{P_R}: x \leftarrow \mathrm{Ver\&Ext}[\mathscr{C}](\underline{\underline{x}}, \overline{x})$ (314)

(b) **Non-interactive Com scheme**

Commit phase:	**Open phase (type 1):**	**Open phase (type 2):**
$\mathrm{P_S}: \underline{x} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{C}](x)$ (315)	$\mathrm{P_S} \to \mathrm{P_R}: (\underline{x}, x)$ (317)	$\mathrm{P_S} \to \mathrm{P_R}: \underline{x}$ (319)
$\mathrm{P_S} \to \mathrm{P_R}: \overline{x} = \mathscr{C}(x; \underline{x})$ (316)	$\mathrm{P_R}: \mathscr{C}(x; \underline{x}) \stackrel{?}{=} \overline{x}$ (318)	$\mathrm{P_R}: x \leftarrow \mathrm{Ver\&Ext}[\mathscr{C}](\underline{x}; \overline{x})$ (320)

(c) **Simpler NI Com scheme**

Figure B.1: **Diverse notation for real Com Schemes.**

of non-malleability (which often require use of identifiers of the intervening entities and of the execution session). For example, sometimes the opening may correspond to sending the original randomness selected by the sender, but other times the receiver should not learn it. In some cases it is enough to reveal the randomness (e.g., an encryption key) and the receiver uses it to extract the committed value. In other cases the opening needs to explicitly contain the committed value.

Interactive case (Figure B.1a). While this dissertation is mostly focused on non-interactive commitment schemes, it remains implicit that the commitment schemes may in most cases be replaced by interactive versions. Thus, for the sake of generality a notation for interactive commitments is also described. (An application in §C.1.4 exceptionally requires a non-interactive open phase, in order to be able to commit in advance to the randomness needed to open another commitment.)

- **Interactive commit phase.** $\mathrm{P_S}$ and $\mathrm{P_R}$ interact (309), with $\mathrm{P_S}$ having as private input the value x that it wants to commit. As a result of the probabilistic interaction, $\mathrm{P_R}$ obtains the "commitment" $\overline{x}_\mathrm{R}$ and $\mathrm{P_S}$ obtains the "randomness" (a.k.a. "encoding") $\underline{x}_\mathrm{S}$ needed to later open the committed value x, as well as a "commitment" value $\overline{x}_\mathrm{S}$. $\mathrm{P_R}$ also learns a

commitment value $\overline{x}_{\mathrm{R}}$, which typically is the same as that learned by $\mathrm{P_S}$, but in general it need not be so, and for that reason the respective symbols values may have distinct subscripts. In an interactive protocol, $\mathrm{P_R}$ may also need to retain private randomness $\underline{x}_{\mathrm{R}}$, which may be useful for the later open phase. The hiding property ensures that in the commit phase the output of (even a malicious) $\mathrm{P_R}$ does not reveal anything (in a computational complexity sense) about the committed value.

- **Interactive open phase.** Later, the parties interact in an open phase (310). $\mathrm{P_S}$ uses as input the committed value x, the respective private randomness $\underline{x}_{\mathrm{S}}$ and the (possibly private) commitment value $\overline{x}_{\mathrm{S}}$ known to her. $\mathrm{P_R}$ uses as input his (possibly private) randomness $\underline{x}_{\mathrm{R}}$ and the commitment value $\overline{x}_{\mathrm{R}}$ known to him. As a result of the interaction, $\mathrm{P_R}$ learns the value x that had been previously committed by $\mathrm{P_S}$, and possibly some additional randomness $\underline{\underline{x}}$ (in some cases it may be equal to the initial private randomness of $\mathrm{P_S}$), e.g., possibly allowing transferability of the predicate verification that the committed value x and the initially received commitment $\overline{x}_{\mathrm{R}}$ are consistent). The binding property ensures that $\mathrm{P_R}$ obtains the correct committed value x (or it aborts if it detects malicious behavior from $\mathrm{P_S}$), except perhaps with at most a negligible probability in the security parameter(s).
- **Application example.** In the example alluded in Remark 4.3 an Equiv-Com from $\mathrm{P_S}$ to $\mathrm{P_R}$ is based on an inherently interactive protocol, where $\mathrm{P_R}$ builds in the commit phase an Ext-Com whose randomness is only revealed to $\mathrm{P_S}$ in the open phase. Also, the randomness used by $\mathrm{P_S}$ in the commit phase is not all revealed to $\mathrm{P_R}$ in the open phase, lest it would compromise the equivocability property.

Non-interactive case (Figure B.1b). In a non-interactive scheme the notation can be changed to enable a clear distinction between local computations and communications.

- **Non-interactive commit phase.** The *commit* phase starts with a local probabilistic computation by $\mathrm{P_S}$, who alone computes a pair composed of the public commitment $\overline{x}$ and the private "randomness" $\underline{x}$ needed to later open it (311). Then, $\mathrm{P_S}$ simply sends the commitment to $\mathrm{P_R}$ (312). In this non-interactive setting there is no longer a need to distinguish the commitment values ($\overline{x}_{\mathrm{S}}$, $\overline{x}_{\mathrm{R}}$).
- **Non-interactive open phase.** In the *open* phase, $\mathrm{P_S}$ computes (possibly probabilistically) an element $\underline{\underline{x}}$ that enables verification and extraction of the committed value, and sends it to $\mathrm{P_R}$ (313). Depending on the scheme, this might be the initial randomness

$\underline{x}$, or the randomness and the committed value, or something different (e.g., a NIZKP transcript) that reveals the committed value but hides the original randomness. Then, $\mathrm{P_R}$ uses an appropriate procedure to extract the committed value x and verify its correctness with respect to the commitment $\overline{x}$ (314).

- **Application example.** This notation fits a scheme whose open phase corresponds to sending the committed value x and a NIZKP $\underline{\underline{x}}$ that it is the value that had been committed, without revealing the original randomness $\underline{x}$. In another example considered in §C.1.4, the actual commitment contains several sub-commitments, of which some but not all are opened (either in the commit or the overall open phase), thus implying that not all original randomness $\underline{x}$ is not revealed to $\mathrm{P_R}$.

Simpler non-interactive case (Figure B.1c).

- **Simpler non-interactive commit phase.** In a more simplified (and usual) case, the "randomness" in the *commit* phase is obtained as a first step (315) and then the commitment is obtained by simply applying a commitment "function" ($\mathscr{C}$) to the value to commit and to the prior obtained "randomness" (316).
- **Simpler non-interactive open phase.** In a simple open phase, $\mathrm{P_S}$ sends the committed value and the respective randomness (317), letting $\mathrm{P_R}$ simply check that the commitment "function" applied to the received value and randomness yields the previously obtained "commitment value" (318).
- **Alternative opening.** As an alternative, typically considered for extractable commitment schemes based on hybrid encryption schemes (i.e., encrypt a random value and uses its pseudo-random expansion as a one-time pad to more efficiently encrypt the committed value), the message of the open phase may correspond to simply sending the randomness (319); then, V verifies that the randomness is consistent with the commitment and then uses it to extract the committed value (320).

Adaptive notation. Throughout the dissertation, the use of notation for the simpler non-interactive case is preferred, leaving implicit that an interactive scheme could also be used with a corresponding notational adjustment. The notation may be augmented with more information about the commitment scheme properties. Specifically, a subscript may be used to denote an intended simulatability property (e.g., Ext or Equiv), indicating that the respective action (e.g., Ext or Equiv) is achievable by a simulator in the respective commit

$$\chi_i = \mathscr{C}(b_i; r_i) : i \in \{1,2\} \quad (321)$$
$$\chi_3 = \chi_2 *' \chi_1 \quad (322)$$
$$b_3 = b_2 + b_1 (\mathrm{mod}\ \#(\mathbb{G})) \quad (323)$$
$$r_3 = r_2 \ddagger_* r_1 \quad (324)$$
$$\mathscr{C}(b_3; r_3) = \chi_3 \quad (325)$$

(a) **Regular additive-homomorphism**

$$\chi_i = \mathscr{C}(b_i; r_1) : i \in \{1,2\} \quad (326)$$
$$\chi_4 = \chi_2 *'_{(b_2)} \chi_1 \equiv \chi_2 * \chi_1^{(1-2b_2)} \quad (327)$$
$$b_4 = b_2 \oplus b_1 \quad (328)$$
$$r_4 = r_2 \ddagger_{*(b_2)} r_1 \equiv r_2 \ddagger_* inv(r_1)^b \ddagger_* r_1^{1-b} \quad (329)$$
$$\mathscr{C}(b_4; r_4) = \chi_4 \quad (330)$$

(b) **Pseudo XOR-homomorphism**

Figure B.2: **From an additive homomorphism to a pseudo XOR-homomorphism.** r_i denotes the randomness used to commit to bit i. $\ddagger_*$ denotes the group operation in the space of randomness, which is either $*$ in multiplicative notation or $+$ in additive notation. $*'_{(b_2)}$ and $\ddagger_{*(b_2)}$ depend on b_2 (the bit associated with the left element of the sum), being $*'$ and $\ddagger_*$, respectively, if $b_2 = 0$, or being the inverse of $*'$ and $\ddagger_*$, respectively, if $b_2 = 1$.

or open phase. When explicitly ensuring non-malleability, a superscript may be added with some of the additional context (e.g., session and sub-session identifiers, names of the parties) used by the commitment and/or opening algorithms. A common reference string may also be used in the superscript, whenever intended that the parameters are defined by the CRS, with a trapdoor known to the simulator.

B.1.2 Notation for homomorphisms and pseudo-homomorphisms

The homomorphic and pseudo-homomorphic operations are described in Figure B.2.

Homomorphisms. A commitment scheme is additively homomorphic (for some additive group operation +, e.g., modular sum, in the space of committable values) if any pair (χ_1, χ_2) of commitments, of a respective pair (b_1, b_2) of values (321), can be combined into a new commitment χ_3 (322) of the value resulting from applying a group addition to the two originally committed values (323), and if the same can be done with the respective randomnesses (324), such that the new commitment χ_3 is what would be obtained from directly committing the resulting bit b_3 using the respective homomorphically obtained randomness (325). The operations in the space of commitments and in the space of randomnesses are also appropriate group operations $(*', \ddagger_*)$. The commitment space is here always considered with multiplicative notation, whereas the randomness space may vary depending on the instantiation.

Pseudo-homomorphisms. In a baseline scenario, P_A knows a pair (χ_1, χ_2) of commitments an a respective pair (r_1, r_2) of randomnesses used to commit two a respective pair (b_1, b_2) of bits (326), and from which it "somewhat-homomorphically" computes a new commitment χ_4 (327), of the XOR (instead of a regular addition $+$) of the two underlying bits (b_1, b_2) (328). The randomness r_4 needed for opening can also be obtained "somewhat-homomorphically" from the two initial randomnesses (329), such that the new commitment χ_4 is what would be obtained from committing the resulting bit b_4 using the respective "somewhat-homomorphically" obtained randomness (330).

In the intended context, the other party (P_B) will initially know only the first and third commitments (χ_1, χ_4), but not the respective randomnesses. Then, it will be asked to make one (and only one) of the following two conditional verifications: if it receives the opening (b_2, r_2) of the second commitment, then it verifies that said opening can be used to produce a commitment χ_2 whose respective "somewhat-homomorphic" combination with χ_1 would indeed be the third commitment χ_4; if it receives the opening (b_4, χ_4) of the third commitment χ_4, then it verifies that it is indeed a consistent opening.

The crucial observation is that the needed properties can be achieved if the "somewhat-homomorphic" operations ($*'^{(b_2)}$ and $*^{(b_2)}$, respectively in the space of commitments and randomness) depend explicitly on the value of the second committed value b_2, as follows (in multiplicative notation): if $b_2 = 0$, then they respectively are $*'$ and $*$ (i.e., regular group-multiplication); if $b_2 = 1$, then they respectively are the inverse of $*'$ and $*$ (i.e., multiplication by the multiplicative inverse of the second argument, a.k.a. group division), respectively, if $b_2 = 1$. (Where $*'$ and $*$ are respectively the operations of the original additively homomorphic Com scheme.) It is assumed that multiplicative inverses are easy to compute in the commitment and randomness spaces.

B.1.3 Symbols and implicit primitives

The protocol specification depends on several primitives (e.g., PRGs), sub-protocols (e.g., commitment schemes), and conventionalized notation (e.g., Boolean circuit specification), whose specification is mostly left implicit, without prejudice of some of them being parametrized by security parameters. Other primitives may require more explicit parametrization (e.g., type of BitCom scheme) and be configured per execution. Some of these symbols and primitives are explained hereafter.

Parties, Boolean circuit and wires.

- **Parties.** The distinct roles of constructor (P_A) and evaluator (P_B) of garbled circuits, with the later being the first to learn the S2PC-with-Coms output, namely the circuit output and the final outer commitments and randomnesses (331).
- **Original Boolean circuit.** A Boolean circuit C specification, which will be specified as part of the input of an execution (332).
- **Original wire indices.** The Boolean circuit specification implicitly defines the outer wires, defining the sets of indices of input wires of each party (I_A, I_B), the sets of private output wires of each party (O_{AA}, O_{BB}), and the set of common output wires O_{AB} (333). The set O_p of all output wires of each party P_p can thus contain private wires and common wires (334). Correspondingly, the set O_{pp} of private output wires of each party P_p can be obtained from the set of output wires of the party after removing the common output wires (335) The set of common output wires is the intersection of the sets of output wires of the two parties (336).
- **Adjusted Boolean circuit.** Whenever there are private output wires of P_A, they need to be masked for the purpose of evaluation by P_B. Thus, the actual evaluated circuit is an adjusted circuit C' (337), which for every original private output wire of P_A XORs an extra auxiliary random input bit of P_A, inputted via an additional input wire ($i \in I_{A'}$) (338). In other words, for each private output wire ($i \in O_{AA}$) of P_A there is a corresponding new input wire ($\theta_A(i) \in I_{A'}$) of P_A, and vice-versa (339).
- **Adjusted indices of common output wires.** While the previously mentioned adjustment is needed for privacy, another adjustment is required for the purpose of indexing the final outer-coms and final outer randomnesses of the output of a S2PC-with-Coms execution. Specifically, even though for each common output wire both parties will learn the same final output bit, each party will learn two different respective outer Coms, and each party will learn only one of the respective randomnesses. The set O_{AB} of original common output wire indices is thus mapped into two distinct sets ($O_{AA'}$, $O_{BB'}$) of common output wires, one for each party (340). This means that each party will now be considered "owner" of new output wire indices in number equal to the number of common output wires (341). Thus, the actual adjusted set $O_{p'}$ of indices of final output wires of each party is equal to the union of the set O_{pp} of original private output wires and the set $O_{pp'}$ of newly mapped output wire indices (one for each common output wire) (342). This means that the initial set O_p of output wires of a party (including private and common wires) is

SYMBOLS AND IMPLICIT PRIMITIVES

Parties, Boolean circuit and wires.

Parties: $\mathrm{P_A}$ (constructor); $\mathrm{P_B}$ (evaluator) (331)

Original Boolean circuit (to obliviously evaluate): C (332)

Original wire indices.

$(I_\mathrm{A}, I_\mathrm{B}, \mathrm{O_{AA}}, \mathrm{O_{BB}}, \mathrm{O_{AB}}) \equiv \mathrm{OuterWires}(C)$ (333)

$\mathrm{O}_p \equiv \mathrm{O}_{pp} \cup \mathrm{O_{AB}} : p \in \{\mathrm{A},\mathrm{B}\}$ (all output wires of P_p) (334)

$\mathrm{O}_{pp} \equiv \mathrm{O}_p \backslash \mathrm{O_{AB}} : p \in \{\mathrm{A},\mathrm{B}\}$ (private output wires of P_p) (335)

$\mathrm{O_{AB}} \equiv \mathrm{O_A} \cap \mathrm{O_B}$ (output wires common to both parties) (336)

Adjusted Boolean circuit (to mask private output of $\mathbf{P_A}$).

$C' = \mathrm{MaskOutA}(C)$ (337)

$(I_\mathrm{A}, I_{\mathrm{A}'}, I_\mathrm{B}, \mathrm{O_{AA}}, \mathrm{O_{BB}}, \mathrm{O_{AB}}) \equiv \mathrm{OuterWires}(C')$ (338)

$\theta_\mathrm{A} : I_{\mathrm{A}'} \to \mathrm{O_{AA}}$ (bijection, with $\#(I_{\mathrm{A}'}) = \#(\mathrm{O_{AA}})$) (339)

Adjusted indices of common output wires.

(to index Coms and Rands of different parties)

$\theta_{p'} : \mathrm{O_{AB}} \to \mathrm{O}_{pp'} : p \in \{\mathrm{A},\mathrm{B}\}$ (340)

$\#(\mathrm{O_{AB}}) = \#(\mathrm{O}_{pp'}) : p \in \{\mathrm{A},\mathrm{B}\}$ (341)

$\mathrm{O}_{p'} = \mathrm{O}_{pp} \cup \mathrm{O}_{pp'} : p \in \{\mathrm{A},\mathrm{B}\}$ (342)

$\theta_{p'} : \mathrm{O}_p \to \mathrm{O}_{p'} : p \in \{\mathrm{A},\mathrm{B}\}$ (343)

Security parameters.

Security parameters. $1^\kappa, 1^\sigma$ (344)

C&C selection style. C&C-STYLE $\in \{\text{NI-A}, \text{INT-B}\}$ (345)

C&C parameters. $(v_{\min}, v_{\max}), (e_{\min}, e_{\max}), s$ (346)

$(1 \le e_{\min} \le e_{\max},\ v_{\min} \le v_{\max} < s$

$v_{\min} + e_{\max} = v_{\max} + e_{\min} = s,$

$\sum_{e=e_{\min}}^{e_{\max}} \mathrm{Bin}(s,e) \gtrsim 2^s\ )$

Randomness and symmetric primitives.

RSC related. $\mathrm{Gen_{Seed}}, \text{CR-Hash}, \mathscr{C}_\mathrm{RSC} \equiv \mathscr{C}_\mathrm{RSC}^\mathrm{Equiv}$ (347)

Garbled-circuit related. $\mathrm{PRGen_{GC}}, \mathrm{GC_{Eval}}$ (348)

Garbled-keys related. $\mathrm{PRGen_{InKey}}, \epsilon, \mathscr{C}_\mathrm{InKey}$ (349)

Other randomness-related primitives.

$\mathrm{Gen_{\$ForCom}}, \mathrm{PRGen_{\$ForCom}}$ (350)

$\mathrm{PRGen_{BitString}}, \mathrm{PRGen_{AuxiSeed}}, \mathrm{PRGen_{\$ForLHT}}$ (351)

Outer BitCom schemes.

Related to outer BitComs. For $p \in \{\mathrm{A},\mathrm{B}\}$:

$\mathscr{B}_p$ (BitCom scheme for outer BitComs of P_p) (352)

$z_p \equiv \mathrm{PRGen_{\$ForCom}}[\mathscr{B}_p](1)$ (353)

(fixed "randomness" to commit bit 1)

$z'_p \equiv \mathscr{B}_p(1; z_p)$ (fixed BitCom of bit 1 — not private) (354)

Homomorphic operations.

$*'$ (homomorphic group operation for BitComs) (355)

$*$ or $+$ (homomorphic group operation for encodings) (356)

Extension from BitComs to BitStringComs. For $p \in \{\mathrm{A},\mathrm{B}\}$:

$\mathscr{C}_p$ (extension of $\mathscr{B}_p$ to BitStringCom scheme) (357)

$pos_{set} : set \to [\#(set)]$ (bijection) (358)

$\mathrm{Stringize} : \mathscr{B}_p^{\#(set)} \to \mathscr{C}_p$ (359)

If $\#(\text{committable space of } \mathscr{B}_\mathrm{B}) = 2$, then $\mathscr{C}_\mathrm{B} \equiv \vec{\mathscr{B}}_\mathrm{B}$ and:

$\mathrm{Stringize}\left(\langle \sigma'_i : i \in set \rangle\right) \equiv \sigma'_{set} = \langle \sigma'_i : i \in set \rangle$ (360)

$\mathrm{Stringize}\left(\langle \sigma_i : i \in set \rangle\right) \equiv \sigma_{set} = \langle \sigma_i : i \in set \rangle$ (361)

If $\left(\#(\text{committable space of } \mathscr{B}_\mathrm{B}) \ge 2^{|set|}\right)$ (e.g., ElG-Com) $\wedge$

($\mathscr{C}_\mathrm{B}$ is additively-pseudo-hom.), then $\mathscr{C}_\mathrm{B} \equiv \mathscr{B}_\mathrm{B}$ and:

$\mathrm{Stringize}\left(\langle \sigma'_i : i \in set \rangle\right) \equiv \sigma'_{set} = *'_{i \in set} \sigma'^{2^{pos_{set}(i)}}_i$ (362)

$\mathrm{Stringize}\left(\langle \sigma'_i : i \in set \rangle\right) \equiv \sigma_{set} = \sum_{i \in set} \sigma_i 2^{pos_{set}(i)}$ (363)

Intermediate BitCom schemes.

Related to input bits of $\mathbf{P_A}$

$\mathscr{B}_\mathrm{ConA}$ (BitCom scheme for connectors of input $\mathrm{P_A}$) (364)

$\mathscr{B}_\mathrm{FLA}$ (Ext-BitCom scheme for F&L) (365)

$\mathrm{Extract_{Bit}}$ (extractor of bits committed by $\mathscr{B}_\mathrm{FLA}$) (366)

Related to connectors of input of $\mathbf{P_B}$

$\mathrm{OT_{Level}} \in \{\text{BitComs}, \text{GC-InWires}\}$ (367)

$\mathrm{OT_{Type}} \in \{1/2, 2/1\}$ (368)

If $\mathrm{OT_{Type}} \stackrel{?}{=} 2/1$: (i.e., for 2-out-of-1-OT)

$\mathscr{B}_\mathrm{OT}$ (a 2-to-1-square-scheme) (369)

h (XOR-homomorphism, class of an encoding) (370)

$\mathrm{Extract_{PairOpenings}}$ (Sqrt extraction) (371)

If $\mathrm{OT_{Type}} \stackrel{?}{=} 1/2$ (i.e., for 1-out-of-2-OT)

$\mathscr{B}_\mathrm{OT} \equiv \mathscr{C}_\mathrm{OT}$ (additively-homomorphic Com-scheme) (372)

$(\mathscr{E}, f) : \mathscr{C}_\mathrm{OT}(\cdot) = \mathscr{E} \circ f(\cdot)$ (encryption of f-image) (373)

RandLHT (Randomized LHT over $\mathscr{C}_\mathrm{OT}$) (374)

Related to connectors of output of $\mathbf{P_B}$.

$\mathscr{B}_\mathrm{FLB} \in \mathrm{DUAL}(\mathscr{B}_\mathrm{FLA})$ (Equiv-BitCom scheme for F&L) (375)

$\mathrm{Gen_{PairOpenings}}$ (to generate openings for $\mathscr{B}_\mathrm{FLB}$) (376)

$\mathrm{Extract_{Trapdoor}}$ (to extract trapdoor of $\mathscr{B}_\mathrm{FLA}$) (377)

Figure B.3: **Symbols and implicit primitives for S2PC-with-Coms protocol.** (Protocol definition starts in Figure B.5.) Legend in §Notation.

mapped (with a bijection) into an adjusted set $\mathrm{O}_{p'}$ of output wires that has no intersection with the corresponding set $\mathrm{O}_{\bar{p}'}$ of the other party (343).

Security parameters.

- **Security parameters.** A (long-term) cryptographic security parameter κ and a statistical security parameter s', with which the protocol execution must conform (344). An optimization based on an additional (and smaller) "short-term binding" security parameter is given in §B.4.1.3.

- **Cut-and-choose selection style.** A cut-and-choose selection style (C&C-STYLE), either defined as interactive with decision by P_B (INT-B), or as non-interactive with decision by P_A (NI-A) (345). In the later case the statistical security parameter must at least equate the computational security (possibly a short-term binding parameter).
- **Cut-and-choose parameters.** C&C partitioning parameters (e_{min}, e_{max}, v_{min}, v_{max}, s), consistent with the statistical security parameter (346). For example, for a FIXED mode, the number of evaluation and check instances may be defined in advance, i.e., the minimum values (e_{min}, v_{min}) being equal to the respective maximum values (e_{max},v_{max}), though not yet deciding which challenge type will be attributed to each index. In a VARIABLE mode the number of check and evaluation indices is not defined in advance but is still bound to some restrictions, namely that their sum still needs to be the total number s of garbled circuits. In any case, the parameters must be consistent with the respective statistical security parameter, namely such that the error probability of a protocol execution (in the worst adversarial scenario with a malicious P_A and an honest P_B) must at most be less than a value negligibly close to one half raised to the power of the number s' of intended bits of statistical security (notwithstanding possible deviations negligible in the security parameter κ). For example, for a FIXED mode see (cell E6 in Table 3.1. For a VARIABLE mode see for example rows 8 to 10). For example, if the parameters require that there must be at least as many *check* (v) as *evaluation* (e) instances, then the minimum number e_{min} of evaluation instances is the largest integer not greater than half of the number of garbled circuits. If the number s' of desired bits of statistical security is integer, then the total number s of challenges is defined as the smallest integer larger than s' (e.g., see rows 8–9 in Table 3.2), and the values v and e remain undetermined until the later CHALLENGE stage of the protocol.

Randomness and symmetric primitives.

- **RSC related primitives (347).** A mechanism Gen_{Seed} to sample PRG seeds, i.e., to be used in diverse PRG generation primitives, and a collision-resistant hash function CR-Hash. An equivocable commitment scheme $\mathscr{C}^{Equiv}_{RSC}$ specifically used to create a single equivocable RSC commitment of the global hash value calculated in the RSC technique.
- **Garbled-circuit related primitives (348).** A primitive $PRGen_{GC}$ for pseudo-random generation of GCs; and a primitive GC_{Eval} for evaluation of GCs. There is no need for a specialized GC-verification algorithm, as the verification is made by reconstructing the GC

from a respective seed and then checking that it integrates well with a proper pre-image of the global hash. For simplicity of description, the garbling scheme is assumed to allow the circuit input keys to be specified prior to the construction of GCs. This is consistent with typical garbling scheme proposals, but it is nonetheless simple to avoid the assumption, using a technique described in §3.3.4.

- **Garbled-keys related (349).** A primitive $\text{PRGen}_{\text{InKey}}$ for pseudo-random generation of input keys; a regular commitment scheme (i.e., not necessarily equivocable or extractable) used to commit GC input keys of P_A. Also for simplicity, it is assumed that the circuit output keys (and not others) exceptionally reveal the underlying bit, via some efficiently computable predicate ϵ, e.g., their least significant bit. This simplifies a later step (567) in the Evaluate stage.
- **Other randomness-related primitives.** The protocol requires further generation of randomness or pseudo-randomness. A procedure $\text{Gen}_{\text{\$ForCom}}$ is defined for the random sampling of the "randomness" needed to produce commitments and to verify the correctness of an opened value (350). In order to be usable for different commitment schemes, this procedure requires as input the specification of the commitment scheme, and for generality also the value to be committed. This procedure will be used to produce *outer* and *intermediate* BitCcoms in the Produce initial BitComs of P_A and Produce initial BitComs of P_B stages, as well as the equivocable commitment of the RSC global hash in the Commit stage. Analogously, a procedure $\text{PRGen}_{\text{\$ForCom}}$ is also defined for pseudo-random generation of "randomness" needed to produce a commitment ($\mathscr{C}$) or BitCom ($\mathscr{B}$) and verify the correctness of an opened value. As extra input this procedure will also require a random or pseudo-random seed (e.g., 256 random bits).
  Other PRG primitives are included, such as a primitive $\text{PRGen}_{\text{BitString}}$ for pseudo-random generation of bit-strings of specified length, e.g., used to generate "random" permutation bits related to the connectors of input of P_A, and to For notational convenience, another PRG primitive $\text{PRGen}_{\text{AuxiSeed}}$ is considered for generating a new pseudo-random auxiliary seed to be used in other PRG procedures. An additional primitive $\text{PRGen}_{\text{\$ForLHT}}$ is also included to generate randomness for linear homomorphic transformations of ElGamal commitments. (351)
  Some notes about the implementation of the PRGs are given in Remark B.6. For simplicity, the security parameter is implicit in the notation used for PRG procedures.

Outer BitCom schemes. There is a set of *outer* and *intermediate* BitCom schemes. The *outer* BitCom schemes ($\mathscr{B}_{\mathrm{A}}$, $\mathscr{B}_{\mathrm{B}}$) are used to commit the input bits of $\mathrm{P_A}$ and $\mathrm{P_B}$ in the format that should be retained after the protocol evaluation. The *intermediate* BitCom schemes are related to connectors and/or to the auxiliary forge-and-lose ($\mathscr{B}_{\mathrm{FLA}}$, $\mathscr{B}_{\mathrm{FLB}}$) and oblivious-transfer ($\mathscr{B}_{\mathrm{OT}}$) techniques. Their actual specification is later described as part of the protocol setup. The parameter of the outer schemes are defined by a global trusted setup; the parameters of intermediate scheme may either be derived from outer schemes or be selected independently by each party and proved informed during the protocol execution.

- **Related to outer BitComs.** An Ext-BitCom scheme $\mathscr{B}_p$ for outer BitComs (of the input and output bits) of party P_p (352). The two BitCom schemes may possibly be the same, e.g., in case it is defined from a CRS trusted setup. The BitCom scheme is assumed extractable with trapdoor (e.g., GM and ElGamal BitCom schemes), i.e., whenever the trapdoor is available (e.g., after a ZKPoK of trapdoor in a PKI setting). For each BitCom scheme $\mathscr{B}_p$ devised for the outer BitComs of a party, the parties also establish by convention a fixed encoding z_p of bit 1, e.g., obtained pseudo-randomly from BitCom scheme specification (353), and/or some other arbitrary rule. Correspondingly, the BitCom z'_p of bit 1 (not intended as private) obtained from such encoding is also considered common knowledge (354). The fixed encoding and BitCom will later be useful for adjusting the output BitComs of each party, based on random offset bits.
- **Homomorphisms.** Each BitCom scheme is XOR pseudo-homomorphic (possibly XOR homomorphic), with the respective group operation in the BitCom space being denoted with a multiplicative notation ($*'$) (355). The respective group operation in the space of encodings (pre-images) is either denoted with multiplicative notation ($*$, e.g., for multiplication of square-roots, when using Blum or GM BitComs) or additive notation ($+$, e.g., for sum of exponents, when using ElGamal or Pedersen BitComs) (356).
- **BitCom to BitStringCom extension.**
  For each outer BitCom scheme ($\mathscr{B}_p$), a respective extension is considered to a BitStingCom scheme $\mathscr{C}_p$, with additively pseudo-homomorphic properties (357).
  Given the extension to bit-strings, it is useful to consider a function that for any fixed set of wire indices maps the set into a corresponding set of indices of a vector (i.e., with positions between 1 and the numebr of elements) (358). Then, an operation suggestively denoted "stringize" denotes the tranformation of a vector of BitComs into a respective single BitStringCom (359). The actual operation varies with the type of underlying

BitCom scheme.

If the committable space is two (e.g., GM BitComs), then the extension is just an identity vectorization, i.e., a vector of BitComs is logically tranSformed into a single BitStringCom that in practice is the original vector of BitComs (360). The exact same logic is applied in the space of randomnesses (361).

If the BitCom scheme is supported in a scheme that in practice allows larger elements to be committed, and if the underlying scheme is additively pseudo-homomorphic over the integers modulo the group order (e.g., ElGamal Com scheme), then the *stringize* operation is an homomorphic operation that compacts the vector of BitComs into a single BitStringCom of the size of a single BitCom. The operation is valid if the committable space is at least as large as two to the power of the number of BitComs being combined. Essentially, in the space of commitments, each BitCom is raised to the power of two to the power of its position, so that it becomes a bitstring commitment of the integer equal to two to the power of the position. Then, all such commitments are multiplied, to form a commitment of the string of respective bits (362). In the space of randomnesses, and assuming an additive notation, the randomnesses are stringized by computing each randomness multiplied by a coefficient equal to two to the power of the position of the bit, and then adding all such factors, thus obtaining a single "randomness" element (363). Given the homomorphic properties, the commitment of a bitstring using as randomness the stringized randomness of respective BitComs, is equal to the direct stringizing of the vector of BitComs.

Intermediate BitCom schemes.

- **Related to input bits of $\mathbf{P_A}$.** A BitCom scheme $\mathscr{B}_{\mathrm{ConA}}$ for the connectors of the input bits of $\mathrm{P_A}$, namely to commit permutation bits in the respective connectors (364). An Ext-BitCom scheme $\mathscr{B}_{\mathrm{FLA}}$ with trapdoor to commit input bits of $\mathrm{P_A}$ in connection with the forge-and-lose technique (365). The trapdoor (t_{FL}) must be initially known to $\mathrm{P_A}$ and unknown to $\mathrm{P_B}$. However, if the trapdoor is even learned by $\mathrm{P_B}$ (e.g., in a well defined circumstance of misbehavior by $\mathrm{P_A}$ — the forge-and-lose case), it allows $\mathrm{P_B}$, using $\mathrm{Extract_{Bit}}$, to extract the input bits committed by $\mathrm{P_A}$ (366).
- **Related to connectors of input of $\mathbf{P_B}$.** The connection for input bits of $\mathrm{P_B}$ is based on some type of oblivious transfer (OT) — two methods are described herein, depending on the level at which OTs are performed (BitComs vs. GC-InWires) (367) and the type of

OT (2-out-of-1 vs. 1-out-of-2) (368).

- **If 2-out-of-1 OT.** If the OT is of type 2-out-of-1, then the associated BitCom scheme $\mathscr{B}_{\text{OT}}$ is a 2-to-1 square scheme (see §2.5.1) (369). In this case it is also useful to consider explicitly the homomorphism between sets — an efficient function h that maps each encoding (i.e., the randomness used to create a BitCom) into the respective *class*, i.e., into the underlying encoded bit (370). There is also an associated extractor function $\text{Extract}_{\text{PairOpenings}}$ that, when with access to a trapdoor (secret of P_A), is able to extract the two openings (a.k.a. square-roots) from any BitCom (371).
- **If 1-out-of-2 OT.** If the OT is of type 1-out-of-2, then the considered OT is based on a XOR pseudo-homomorphic BitCom scheme $\mathscr{B}_{\text{OT}}$ supported on an additively homomorphic commitment scheme $\mathscr{C}_{\text{OT}}$ (372). In practice, the considered scheme (e.g., ElGamal) can be seen as an encryption scheme $\mathscr{E}$ composed with a 1-to-1 function f (which can be one-way) (373). Based on the homomorphic properties, it is possible to consider a randomized linear homomorphic transformation (RLHT) of the committed values (374), which allows a straightforward 1-out-of-2 OT procedure.

- **Related to connectors of output of $\mathbf{P_B}$.** An equivocable BitCom scheme $\mathscr{B}_{\text{FLB}}$ used to commit the output bits of P_B associated with connectors of output of P_B (375). The scheme has a trapdoor known by P_A, allowing generation of a random BitCom and respective pairs of openings (i.e., openings of different bits), by means of a well defined procedure $\text{Gen}_{\text{PairOpenings}}$ (376). Also, the scheme also has an associated well defined procedure $\text{Extract}_{\text{Trapdoor}}$ that allows extracting the trapdoor from any two openings (i.e., to different bits) of the same BitCom (377). Since the scheme has an equivocation-trapdoor equal to the extraction-trapdoor of $\mathscr{B}_{\text{FLA}}$, the two schemes are said to be dual with respect to one another.

B.2 Protocol description

This section gives a detailed description of the new S2PC-with-Coms protocol. The protocol implements a probabilistic 2-output functionality, since besides the circuit output of each party, both parties output random Coms of the input and output wires of both parties, and each party also outputs the randomness associated with the openings of her Coms.

§B.2.1 recalls the ideal S2PC-with-Coms functionality $\mathcal{F}_{\text{S2PCwC}}$ that the protocol will

emulate, describing its internal procedure. Then, §B.2.2 gives the low-level description of the real S2PC-with-Coms protocol, using succinct notation between Figures B.3, and B.11.

For simplicity of description, some aspects related with obvious syntactic or semantic verifications are left implicit; e.g., that received elements are within the expected domains or interpreted under an appropriate representation (e.g., modular reductions or reduction to proper elements in the context of Blum BitComs), that received NIZKPs and NIZKPoKs are verified for validity, that the opening of commitments is also verified by the receiver. Also, the contextual elements (e.g., session, sub-session and message identifiers) are ignored in the communication between $\mathrm{P_A}$ and $\mathrm{P_B}$, but would have to be present in an actual implementation, namely when envisioning concurrent executions. A textual description is given below.

B.2.1 Procedure of an ideal functionality

Figure B.4 describes how the ideal $\mathcal{F}_{\mathrm{S2PCwC}}$ handles incoming messages in the ideal world. In the respective S2PC-with-Coms protocol in the ideal world, the ideal parties ($\widehat{\mathrm{P}}_\mathrm{A}$ and $\widehat{\mathrm{P}}_\mathrm{B}$) simply relay to the ideal functionality and to their environment what they respectively receive from the environment and from the ideal functionality.

Implicit parametrization. The ideal functionality is parametrized by a computational security parameter κ, a specification F of a family of Boolean circuits, from which the parties will choose one to evaluate, and a Boolean flag *ver* that specifies whether or not the Com scheme parameters require verification of correctness based on a trapdoor (378). It is assumed that the parties (i.e., their upper environment when activating the parties with an input) are aware of those parameters. The initial activation of $\mathcal{F}_{\mathrm{S2PCwC}}$ initializes as empty the internal memory (tables) used to memorize inputs, outputs and contexts-to-ignore (379).

Each execution context defined by the environment and to be sent to each party for each new execution is defined by: a session identifier (*sid*) of the execution environment where the ideal functionality is embedded, a sub-session identifier (*cid*) that identifies a particular execution of a secure computation protocol via this ideal functionality, and the parties participating in the execution (380) (also defining the order of first and second receiver).

Message receiving loop. After initialization, the ideal functionality handles any received message (381), expecting to receive three types of well-formed messages: messages with

Implicit parametrization.

$\mathcal{F} \equiv \mathcal{F}_{\text{S2PCwC}}[\kappa, F, ver]$ (378)

$\text{Init}[\mathcal{F}, sid] = \{\mathcal{F} \equiv \mathcal{F}_{sid} : \text{Ignore} = \bot, \text{In} = \bot, \text{Out} = \bot\}$ (379)

$\mathcal{Z} : ctx \equiv (sid, cid, \text{P}_\text{B}, \text{P}_\text{A})$ (380)

Message receiving loop.

If $\widehat{\text{P}}_p \to \mathcal{F} : msg \equiv (mid, ctx[, ...])$ (381)

If $mid =^? \texttt{in-}i$, then $\text{Init}_{\text{Thread}}[\text{ValInput}, msg, \text{P}_p]$ (382)

If $mid =^? \texttt{OK}$, then $\text{Init}_{\text{Thread}}[\text{SecondOut}, msg, \text{P}_p]$ (383)

If $mid =^? \texttt{abort}$, then $\text{Init}_{\text{Thread}}[\text{Ab}, msg, \text{P}_p]$ (384)

Input validation.

ValInput $[(\texttt{in-}i, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), (t_\text{B}, x_\text{B})), \text{P}_p] = \{$ (385)

If $\text{Ignore}[ctx] = 1 \vee \text{In}[ctx, i] \neq \bot$, then $\downarrow \bot$ (386)

$bool = [\neg ver \vee (ver \wedge \text{Ver}(\mathscr{C}_\text{B}, t_\text{B}))]$ (387)

If $\neg bool$, then $\{$ (388)

$\text{Ignore}[ctx] = 1$ (389)

$\mathcal{F} \to \widehat{\text{P}}_{\bar{p}} : (\texttt{bad} - 1, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C))$ (390)

$\downarrow \bot \quad \}$ (391)

$\text{In}[ctx, i] = (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C, x_p)$ (392)

$\mathcal{F} \to \widehat{\text{P}}_{\bar{p}} : (\texttt{got-}i, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C))$ (393)

If $\text{In}[ctx, 3 - i] \neq \bot$, then $\text{FirstOut}[ctx] \quad \}$ (394)

First output.

FirstOut $[ctx] = \{$ (395)

$(y_\text{A}, y_\text{B}) = F_C(x_\text{A}, x_\text{B})$ (396)

$\underline{x}_p \leftarrow^\$ \text{Gen}_{\$\text{ForCom}}[\mathscr{C}_p](x_p) : p \in \{\text{A}, \text{B}\}$ (397)

$\overline{x}_p = \mathscr{C}_p(x_p; \underline{x}_p) : p \in \{\text{A}, \text{B}\}$ (398)

$y_p \leftarrow^\$ \text{Gen}_{\$\text{ForCom}}[\mathscr{C}_p](y_p) : p \in \{\text{A}, \text{B}\}$ (399)

$\overline{y}_p = \mathscr{C}_p(y_p; \underline{y}_p) : p \in \{\text{A}, \text{B}\}$ (400)

$res_\text{B} = \left(\overline{x}_\text{A}, (\underline{x}_\text{B}, \overline{x}_\text{B}), \overline{y}_\text{A}, (y_\text{B}, \underline{y}_\text{B}, \overline{y}_\text{B})\right)$ (401)

$res_\text{A} = \left((\underline{x}_\text{A}, \overline{x}_\text{A}), \overline{x}_\text{B}, (y_\text{A}, \underline{y}_\text{A}, \overline{y}_\text{A}), \overline{y}_\text{B}\right)$ (402)

$\text{Out}[ctx] = res_\text{A}$ (403)

$\mathcal{F} \to \widehat{\text{P}}_\text{B} : (\texttt{out-1}, ctx, res_\text{B}) \}$ (404)

Second output.

SecondOut $[(\texttt{OK}, ctx), \text{P}_p] = \{$ (405)

If $\text{P}_p \neq \text{P}_\text{B}$, then $\downarrow \bot$ (406)

If $\text{Ignore}[ctx] = 1 \vee \text{Out}[ctx] = \bot$, then $\downarrow \bot$ (407)

$\text{Ignore}[ctx] = 1$ (408)

$\mathcal{F} \to \widehat{\text{P}}_\text{A} : (\texttt{out-2}, ctx, \text{Out}[ctx]) \quad \}$ (409)

Handle `abort` requests.

Ab$[(\texttt{abort}, ctx), \text{P}_p] = \{$ (410)

If $\text{Ignore}[ctx] = 1$, then $\downarrow \bot$ (411)

If $\wedge_{i \in \{1,2\}} \text{In}[ctx, i] = \bot$, then $\downarrow \bot$ (412)

If $\text{Out}[ctx] \neq \bot \wedge \text{P}_p = \text{P}_\text{A}$, then $\downarrow \bot$ (413)

$\text{Ignore}[ctx] = 1$ (414)

$\mathcal{F} \to \widehat{\text{P}}_{\bar{p}} : (\texttt{abort}, ctx)$ (415)

Figure B.4: **Flow of S2PC-with-Coms execution in the ideal-world. Legend:** $\widehat{\text{P}}_p$ (party in the ideal world, with $p \in \{\text{A}, \text{B}\}$); C (Boolean circuit specification, implicitly defining the domain of circuit-inputs and circuit-outputs of both parties); x_p and y_p (circuit input and circuit output, respectively, of party $\widehat{\text{P}}_p$); $\mathscr{C}_p$ (commitment scheme used to commit the input and output of $\widehat{\text{P}}_p$. $\bar{\cdot}$ (commitment of $\cdot$); $\underline{\cdot}$ (randomness used to commit $\cdot$); $\leftarrow^\$ \cdot$ (random sampling using probabilistic procedure $\cdot$). For simplicity, some checks by $\mathcal{F}_{\text{S2PCwC}}$ are left implicit, e.g., that the circuit is of size polynomial in κ, and that the sender of a message is mentioned in the context ctx in the expected position. The communication between $\mathcal{F}_{\text{S2PCwC}}$ and $\mathcal{S}$ is omitted. The implicit parameters (κ, F, ver) are known by all parties. For simplicity, some concurrency checks (e.g., use of locks) are omitted within the ideal functionality — it may be assumed that in respect to the local actions of $\mathcal{F}$ the blocks of actions in each thread are atomic (386–404), (394–409) (411–415).

identifier `in-`i (for some $i \in \{1, 2\}$), committing the input of each party (382); messages where the first receiving party authorizes the output to be sent to the second receiver (383); and messages requesting an abort of an execution (384). Any well-formed message is handled concurrently in a new thread.

Input validation. The thread for input validation receives a message with an identifier `in-`i that identifies the ordering of the party in terms of output receiving, and with further

components that specify the execution context *ctx*, the public parameters ($\mathscr{C}_\mathrm{B}$, $\mathscr{C}_\mathrm{A}$) of the two Com schemes (possibly the same) of the two parties, the specification C of the circuit to evaluate, an optional trapdoor t_p of her own Com scheme $\mathscr{C}_p$ and the private circuit input x_p. (385) (some checks are left implicit).

If $\mathcal{F}_{\text{S2PCwC}}$ is already ignoring messages for this context, or if for this context and with this ordering it already received a valid input message from this party (386) (the case of the same party being in both sides is also possible, but trivial).

$\mathcal{F}_{\text{S2PCwC}}$ determines whether or not (*bool*) to accept the proposed Com scheme of the party that sent the message, in case the protocol is parametrized (with *ver* = `true`) for a respective verification (387). If the verification is necessary, it is performed with an implicit procedure (Ver) that uses the trapdoor of the scheme. If the Com scheme parameter is rejected (388), then the ideal functionality ignore further messages with the same execution context (389), sends to the other party $\mathrm{P}_{\bar{p}}$ a message of type `bad-`i, thus informing that the proposed parameters were rejected, and also informing the public parameters of the pair of proposed Com schemes and the circuit specification (390), and then exits the thread (391).

If instead the Com scheme is accepted, then $\mathcal{F}_{\text{S2PCwC}}$ stores the received input (392) (the trapdoor is not necessary), and sends a message to the other party $\mathrm{P}_{\bar{p}}$, similar to the case of bad parameters but replacing the message identifier to `got-i` (393). If a consistent input message has also been stored from the other party, then $\mathcal{F}_{\text{S2PCwC}}$ proceeds to compute the output (394) and then exits the thread. If the second output has not been received, then the ideal functionality simply exits the thread.

First output. Once two complementary inputs have been stored for the same context (395), $\mathcal{F}_{\text{S2PCwC}}$ computes the needed outputs, by evaluating the Boolean circuit (396), then generating randomness $\underline{x}_p$ suitable for committing each input x_p (397) and then using it to produce a respective commitment $\overline{x}_p$ (398), then doing the same with the circuit outputs (399–400), and preparing the tuple of outputs for each party (401–402), which contain the circuit output, all the commitments and only the randomnesses respect to the input and output of the party. $\mathcal{F}_{\text{S2PCwC}}$ stores (in a global variable) the output of the second receiver (P_A) (403) and then sends to the first receiver P_B a contextualized message of type `out-1` with his output (404). (Note: as defined in (407), after this point any abort request from P_A is ignored, so that an honest P_B outputs correctly even if P_A is malicious.)

Second output. Upon receiving a contextualized `OK` message, $\mathcal{F}_{\text{S2PCwC}}$ initiates a respective thread (405), checking that the requesting party is in the role of first receiver (P_B) (406), that the context is not yet being ignored or that the output has already been computed (407). If the validations pass, then $\mathcal{F}_{\text{S2PCwC}}$ decides to ignore any subsequent messages with the same context (408) and then sends to the second receiver (P_A) a contextualized message of type `out-2` with her output (409), and then exits the thread.

Handle abort requests. Requests for early abort must also be handled by $\mathcal{F}_{\text{S2PCwC}}$ (410). Abort requests are ignored, i.e., the thread exits immediately, if the context is already being ignored (411), if no input has yet been stored in respect to the indicated context (412) and in case the request is made by the second receiver after the final output has already been stored (413). If the request is accepted, then $\mathcal{F}_{\text{S2PCwC}}$ decides to ignore any subsequent messages with the same context (414) and then sends to the other party $\text{P}_{\bar{p}}$ a respective contextualized `abort` message (415), and terminates this thread with internally outputs `true` in this thread.

B.2.2 Stages of a real protocol execution

Summary. This subsection describes the S2PC-with-Coms protocol across several Figures, as follows. Figure B.5 describes the Setup stage. Figure B.6 describes the stage (Produce initial BitComs of P_B) where P_B produces his initial BitComs, and the fist part of the coin-flipping stage (Coin-flip Permutations (Start)) where P_B initiates the coin-flipping protocol. If all underlying commitments have non-interactive commit phase, and all ZK sub-protocols are non-interactive, then these components only include communication from P_B to P_A. Figure B.7 describes the stage (Produce initial BitComs of P_A) where P_A produces her initial BitComs, including those supporting the forge-and-lose technique. Figure B.8 describes the Commit stage of the cut-and-choose structure (using a RSC technique), where P_A commits to the connectors and the garbled circuits, using a single RSC short Equiv-commitment. Figure B.9 describes the decision of the cut-and-choose partition, the second part of the coin-flipping of permutations (Coin-flip Permutations (Continue)), and the Respond and Verify stages. Figure B.10 describes the Evaluate stage. Figure B.11 describes the Transmit Circuit Output of P_A, the Coin-flip Permutations (Finish), the Permute Outer Coms and the Final Output stages.

Stage 0 — SETUP. In a setup phase, both parties are initiated with concrete private and common parameters. In actual implementations, some of the common parameters (including the Boolean circuit) may instead be unilaterally proposed by one of the parties.

- **Implicit parametrization.** The protocol can be considered in two modes in respect to the need to perform trapdoor-based verification of Com scheme parameters. A Boolean flag *ver* determines whether or not each party has to use the trapdoor to prove correctness of the scheme (and in some instantiations this may as a side benefit facilitate simulatability of commitments) (416). The protocol is also parametrized by a computational security parameter κ (417) (and all primitives dependent solely on it, e.g., PRGs and CR-Hash) and a family of Boolean circuits F (418). These are the three parameters (besides the Com schemes) needed to parametrize the ideal functionality that is being emulated (419). There is also a set of BitComs scheme parameters from which it is assumed that a uniform selection is made by a trusted setup (420).
- **Trusted setup initialization.** For each of the two parties, the environment queries an ideal functionality for setup of Com-scheme parameters. (421). By default the trapdoor of the Com scheme will be empty (422), but if the protocol requires trapdoor-based verification then the ideal setup selects for each party a pair of public and private Com scheme parameters (423). If the scheme does not require trapdoor-based verification, then the ideal setup simply selects random public Com scheme parameters (424). The ideal setup sends to the environment the parameters of the Com scheme of each party (425).
- **Activation of parties.** The environment defines the circuit inputs for each party (426), the execution context containing the session and sub-session identifiers and the pair (P_B, P_A) of identities of the two parties (in the ordering in which the parties will learn their output) (427). It is assumed that the context tuple never repeats (in practice this can be implemented via timestamps, counters, cache and/or coin-flipping at the higher level of the calling environment). The environment activates each party with a initiating identifier `in-`i, the execution context, the Com scheme parameters, the circuit specification, the trapdoor of the Com scheme (or an empty symbol if it does not exist) and the private circuit input (428–429).

  Each party within the execution requests to a local setup functionality (not accessible to the environment) for a CRS (unique to the execution context) that can be used as a basis for the Ext and Equiv Coms needed throughout the protoco, namely within NIZKPs and NIZKPoKs (430). The local CRS answers a random CRS for each new tuple of execution

Stage 0. SETUP.

Implicit parametrization.

$ver \in \{\texttt{true}, \texttt{false}\}$ (trapdoor-based verification?) (416)
κ (computational security parameter) (417)
$F \equiv \{F_C : C \in \mathbb{N}\}$ (family of Boolean circuits) (418)
$\mathcal{F} \equiv \mathcal{F}_{\text{S2PCwC}}[\kappa, F, ver]$ (ideal functionality) (419)
Set-BITCOMS $\equiv$ XOR-PseudoHom-Ext-BITCOMS (420)

Trusted setup of Com scheme parameters.

For $p \in \{\text{A}, \text{B}\}$:
$\mathcal{Z} \to \mathcal{F}_{\text{SetupComs}} : ((sid, \text{P}_p), (\kappa, ver))$ (421)
$\mathcal{F}_{\text{SetupComs}} : t_p = \perp$ (422)
If ver : $\mathcal{F}_{\text{SetupComs}} : (\mathscr{C}_p, t_p) \leftarrow^{\$}$ Set-BITCOMS(κ) (423)
else $\mathcal{F}_{\text{SetupComs}} : \mathscr{C}_p \leftarrow^{\$}$ Set-BITCOMS(κ) (424)
$\mathcal{F}_{\text{SetupComs}} \to \mathcal{Z} : ((sid, \text{P}_p), (\mathscr{C}_p, t_p), ver)$ (425)

Activation of parties.

$\mathcal{Z} : (x_\text{A}, x_\text{B}) \equiv (\langle b_i : i \in I_\text{A}\rangle, \langle b_i : i \in I_\text{B}\rangle)$ (426)
$\mathcal{Z} : ctx = (sid, cid, (\text{P}_\text{B}, \text{P}_\text{A}))$ (427)
$\mathcal{Z} \to \text{P}_\text{B} : \big(\texttt{in-1}, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), (t_\text{B}, x_\text{B})\big)$ (428)
$\mathcal{Z} \to \text{P}_\text{A} : \big(\texttt{in-2}, ctx, (\mathscr{C}_\text{B}, \mathscr{C}_\text{A}, C), (t_\text{A}, x_\text{A})\big)$ (429)
For $p \in \{\text{A}, \text{B}\}$:
$\text{P}_p \to \mathcal{F}_{\text{CRS}} : (\texttt{value}, ctx)$ (430)
$\mathcal{F}_{\text{CRS}}$: If $\text{P}_p \notin \{\text{P}_\text{A}, \text{P}_\text{B}\}$, then ignore request (431)
Else $\mathcal{F}_{\text{CRS}} \to \text{P}_p : (\texttt{value}, ctx, \text{CRS})$ (432)

Define outer BitComs.

$\mathscr{B}_p = \text{Restrict}(\mathscr{C}_p, \{0,1\}) : p \in \{\text{A}, \text{B}\}$ (433)

Define intermediate BitComs.

(BitCom scheme for connectors of P_A.)
$\text{P}_\text{A}, \text{P}_\text{B} : \mathscr{B}_{\text{ConA}} \equiv \mathscr{B}_\text{A}$ (see alternatives in §B.4.1) (434)

If *ver*: (i.e., if GPKI setup with trapdoors)

(BitCom schemes and trapdoor for F&L.)
$\text{P}_\text{A}, \text{P}_\text{B} : \mathscr{B}_{\text{FLA}} \equiv \mathscr{B}_\text{A}$; $\mathscr{B}_{\text{FLB}} = \text{Dual}(\mathscr{B}_{\text{FLA}})$ (435)
$\text{P}_\text{A} : t_{\text{FL}} \equiv t_{\text{FLA}}^{(\text{Ext})} \equiv t_{\text{FLB}}^{(\text{Equiv})} \equiv t_\text{A}$ (436)
(BitCom scheme and trapdoor for OT).
If OT_{Type}=2/1 :
$\text{P}_\text{B}, \text{P}_\text{A} : \mathscr{B}_{\text{OT}} = \text{Dual}(\mathscr{B}_\text{A}))$ (437)
$\text{P}_\text{A} : t_{\text{OT}} \equiv t_{\text{OT}}^{(\text{Equiv})} = t_\text{A}$ (438)
$((\mathscr{B}_{\text{OT}}, t_{\text{OT}}) \in$ 2-to-1-SQUARE-SCHEMES) (439)
If OT_{Type}=1/2 :
$\text{P}_\text{B}, \text{P}_\text{A} : \mathscr{B}_{\text{OT}} \equiv \mathscr{B}_\text{B}$ (440)
$\text{P}_\text{B} : t_{\text{OT}} \equiv t_{\text{OT}}^{(\text{Ext})} \equiv t_\text{B}$ (441)

If $\neg$*ver*: (e.g., for GCRS-based Com-scheme params)

(BitCom schemes for F&L.)
$\text{P}_\text{A} : \big(\mathscr{B}_{\text{FLA}}, t_{\text{FLA}}^{(\text{Ext})}\big) \leftarrow^{\$}$ Set-BITCOMS (442)
$\text{P}_\text{A} : z_\text{A} = \text{NIZKPoK}_{\text{GoodScheme}}(\mathscr{B}_{\text{FLA}})$ (443)
$\text{P}_\text{A} : \mathscr{B}_{\text{FLB}} = \text{DUAL}\,(\mathscr{B}_{\text{FLA}})$ (444)
$\text{P}_\text{A} : t_{\text{FL}} \equiv t_{\text{FLB}}^{(\text{Equiv})} \equiv t_{\text{FLA}}^{(\text{Ext})}$ (445)
(BitCom scheme for OT.)
If OT_{Type}=2/1:
$\text{P}_\text{A} : \mathscr{B}_{\text{OT}} = \mathscr{B}_{\text{FLB}}$ (2-to-1-Square scheme) (446)
$\text{P}_\text{A} : t_{\text{OT}} = t_{\text{FLB}}$ (447)
$\text{P}_\text{A} \to \text{P}_\text{B} : (\texttt{BC-scheme-OT}, ctx, (\mathscr{B}_{\text{OT}}, z_\text{A}))$ (448)
$\text{P}_\text{B} : \mathscr{B}_{\text{FLB}} \equiv \mathscr{B}_{\text{OT}}; \mathscr{B}_{\text{FLA}} = \text{Dual}(\mathscr{B}_{\text{FLB}})$ (449)
If OT_{Type}=1/2:
$\text{P}_\text{B} : \big(\mathscr{B}_{\text{OT}} \equiv \mathscr{C}_{\text{OT}}, t_{\text{OT}}^{(\text{Ext})}\big) \leftarrow^{\$}$ Set-BitComs (450)
$\text{P}_\text{B} : z_\text{B} = \text{NIZKPoK}_{\text{trapdoor}}(\mathscr{B}_{\text{OT}})$ (451)
$\text{P}_\text{B} \to \text{P}_\text{A} : (\texttt{BC-scheme-OT}, ctx, (\mathscr{B}_{\text{OT}}, z_\text{B}))$ (452)

(Further communication of parameters)
If OT_{Type}=1/2:
defer $\text{P}_\text{A} \to \text{P}_\text{B} : (\texttt{BC-scheme-OT}, ctx, (\mathscr{B}_{\text{OT}}, z_\text{A}))$ (453)
(merge with communication in stage 1.2 (Fig. B.7)

Figure B.5: **Protocol S2PC-with-Coms (stage 0).** (See symbols and primitives in Figure B.3. See continuation in Figure B.6.) Legend in §Notation.

context, only once to each party identified in the execution context (431–432).

- **Define Outer BitComs.** Based on the commitment schemes for bit/strings, the parties locally know ho to apply them solely as BitCom schemes (433). (Basically it is useful here to use different notation between the cases of BitComs and bit/string Coms.)
- **Define Intermediate BitComs.**
  - **Connectors of P_A.** For simplicity, the BitCom scheme $\mathscr{B}_{\text{ConA}}$ for connetors of P_A is defined to be the same as the outer BitCom scheme $\mathscr{B}_\text{A}$ of P_A(434).
  - **If the outer schemes have known trapdoor (i.e., if *ver*).**
    - **BitComs for the forge-and-lose technique.** The BitCom scheme $\mathscr{B}_{\text{FLA}}$ for the

forge-and-lose (F&L) in respect to the input bits of P_A is the same as outer BitCom scheme $\mathscr{B}_A$, which has an explicitly known trapdoor; its dual is the BitCom scheme $\mathscr{B}_{FLB}$ for the F&L technique in respect to the output bits is the Dual (435) The trapdoor of both F&L bitcom schemes is the same and is known to P_A (436).

* **BitCom scheme for oblivious transfers.**
  - **2/1-OT.** If the protocol is parametrized for a 2-out-of-1 type of OT, the Equiv-BitCom scheme $\mathscr{B}_{OT}$ for OT is the dual of the outer BitCom scheme $\mathscr{B}_A$ of P_A (437), with trapdoor known by P_A (438). For this type of OT the BitCom scheme must be a 2-to-1-square-scheme (439).
  - **1/2-OT.** If the protocol is parametrized for a 1-out-of-2 type of OT, the BitCom scheme $\mathscr{B}_{OT}$ for OT is the same as the outer Ext-BitCom scheme of P_B(440), and in this case P_B knows the respective trapdoor (441).

(The OT BitCom scheme could also be selected independently of any of the outer BitCom schemes, as happens when the trapdoor of the outer scheme is not known.)

– **If the outer schemes do *not* have known trapdoor (i.e., if $\neg$*ver*).** If the parties do not know the trapdoor of their outer BitCom schemes (e.g., if they were provided by a global CRS, or by a key registration authority that does not provide the secret keys), then the intermediate BitCom schemes for F&L and OT need to be defined by the parties.

* **BitComs for the forge-and-lose technique.** P_A selects by herself a new Ext-BitCom scheme $\mathscr{B}_{FLA}$ with trapdoor, to commit her input bits for the purpose of the F&L technique (442). If need-be, P_A also generates a NIZKP of correct parameters (443) (e.g., needed for a GM BitCom scheme but not for an ElGamal BitCom scheme). (The sending of the parameters and the NIZKP can be deferred if it helps reducing the number of rounds of interaction.) The dual of the scheme is an Equiv BitCom scheme $\mathscr{B}_{FLB}$ used to commit the output bits of P_B (444). By definition, since the schemes are dual their trapdoor is the same (445).
* **BitCom scheme for oblivious transfers.**
  - **2/1-OT.** If the OT is of type 2-out-of-1, then the BitCom scheme $\mathscr{B}_{OT}$ is a 2-to-1-square scheme with trapdoor known by P_A, which for simplicity can be a dual of the F&L BitCom scheme $\mathscr{B}_{FLA}$ defined for the input bits of P_A, and thus can be the same as the F&L scheme $\mathscr{B}_{FLB}$ used for the circuit output bits (446). Again, the trapdoor remains private knowledge of P_A (447). Since P_B needs, in

Stage 1.1. PRODUCE INITIAL BITCOMS OF P_B.

1.1.1. OUTER BITCOMS OF INPUT BITS OF P_B.

$P_B : \sigma_i \equiv \sigma_i^{(b_i)} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_B](b_i) : i \in I_B$ (454)

$P_B \to P_A : \sigma'_i = \mathscr{B}_B(b_i; \sigma_i) : i \in I_B$ (455)

If BitComs from $\mathscr{B}_B$ need ZKP of correctness:

$P_B \to P_A : \mathrm{ZKP}_{\mathrm{GoodBitComs}}[\mathscr{B}_B]\left(\langle \sigma'_i : i \in I_B \rangle\right)$ (456)

1.1.2. OUTER BITCOMS OF OUTPUT OFFSETS OF P_B. (see (335))

$P_B : d_i \leftarrow^{\$} \{0,1\} : i \in O_{BB}$ (random offset bits) (457)

$P_B : \varsigma_i \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_B](d_i) : i \in O_{BB}$ (458)

$P_B \to P_A : \varsigma'_i = \mathscr{B}_B(d_i; \varsigma_i) : i \in O_{BB}$ (459)

If BitComs from $\mathscr{B}_B$ need ZKP of correctness:

$P_B \to P_A : \mathrm{ZKP}_{\mathrm{GoodBitComs}}[\mathscr{B}_B]\left(\langle \varsigma'_i : i \in O_{BB} \rangle\right)$ (460)

1.1.3. INTERMEDIATE OT BITCOMS OF INPUT OF P_B.

If $\mathscr{B}_{OT} \stackrel{?}{=} \mathscr{B}_B$, then $(\mu'_i, \mu_i) \equiv (\sigma'_i, \sigma_i) : i \in I_B$ (461)

Else if $\mathscr{B}_{OT} \neq \mathscr{B}_B$: (e.g., if $\mathrm{OT}_{\mathrm{Type}}$=2/1 or if GCRS)

$P_B : \mu_i \equiv \mu_i^{(b_i)} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_{OT}](b_i) : i \in I_B$ (462)

$P_B \to P_A : \mu'_i = \mathscr{B}_{OT}(b_i; \mu_i) : i \in I_B$ (463)

$P_B \to P_A : \mathrm{NIZKP}_{\mathrm{SameComBits}}[\mathscr{B}_B, \mathscr{B}_{OT}]\left(\langle (\sigma'_i, \mu'_i) : i \in I_B \rangle\right)$ (464)

If $\mathrm{OT}_{\mathrm{Type}} \stackrel{?}{=} 2/1$, then, for $i \in I_B$:

$P_A : \left(\mu_i^{(0)}, \mu_i^{(1)}\right) = \mathrm{Extract}_{\mathrm{PairOpenings}}[\mathscr{B}_{OT}, t_{OT}](\mu'_i)$ (465)

1.1.4. PARSE OUTER BITCOMS OF P_B. For $set \in \{I_B, O_{BB}\}$:

$P_B : \sigma_{set} \equiv \mathrm{Stringize}(\langle \sigma_i : i \in set \rangle)$ (466)

$P_A, P_B : \sigma'_{set} \equiv \mathrm{Stringize}\left(\langle \sigma'_i : i \in set \rangle\right)$ (467)

$P_B(\sigma_{set}) \to P_A : \mathrm{ZKPoK}_{\mathrm{ComOpening}}(\sigma'_{set})$ (e.g., §A.3.4) (468)

(A single ZKPoK of trapdoor might be sufficient, e.g., for GM BitComs, but not for ElGamal BitComs)

Stage CF.1. COIN-FLIP PERMUTATIONS (START).

(Initiate coin-flipping into a well — see Chapter 4)

Wire sets of P_B.

$P_A, P_B : f_1 \equiv \mathscr{C}_B(0; \cdot); f_2 \equiv \mathrm{Identity}$ (469)

$P_B \to \mathcal{F}_{\text{GMCF-1}} : \left(\texttt{in-1}, ctx_{\texttt{cf}}, (D(f_1))^2, f_1\right)$ (470)

$\mathcal{F}_{\text{GMCF-1}} \to P_A : \left(\texttt{req-1}, ctx_{\texttt{cf}}, (D(f_1))^2, f_1\right)$ (471)

Wire sets of P_A.

$P_B : \gamma_{set} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{C}_p]\left(0^{\#(set)}\right) : set \in \{I_A, O_A\}$ (472)

$P_B \to \mathcal{F}_{\text{MCom}} : \left(\texttt{commit}, ctx_{\texttt{com1}}, (\gamma_{I_A}, \gamma_{O_A})\right)$ (473)

$\mathcal{F}_{\text{MCom}} \to P_A : \left(\texttt{receipt}, ctx_{\texttt{com1}}, |(\gamma_{I_A}, \gamma_{O_A})|\right)$ (474)

Figure B.6: **Protocol S2PC-with-Coms (stage 1.1 and CF.1).** (See preceding in Figure B.5 and continuation in Figure B.7.) Legend in §Notation. The $\mathrm{NIZKP}_{\mathrm{GoodBitComs}}$ is for example needed for ElGamal BitComs (e.g., §A.3.2 but not for GM BitComs).

the 2-out-of-1 OT case, to use the OT BitCom scheme to commit his input bits, P_A needs to send the Com scheme parameters, possibly along with a NIZKP of correctness (the same as already produced for the F&L BitCom scheme) 448), which in practice is assumed to also define the parameters of the two F&L schemes (one is equal, the other is dual) (449).

- **1/2-OT.** If the OT is of type 1-out-of-2, then the BitCom scheme $\mathscr{B}_{OT}$ is defined directly by P_B as an Ext scheme with trapdoor (450). P_B generates a NIZKP of correctness (451) (if need be, e.g., for GM BitComs, but not needed for ElGamal BitComs) and sends it along with the public BitCom scheme parameter to P_A(452).

* **Further communication of parameters.** If the OT is of type 1-out-of-2, then P_A has not yet sent to P_B the parameters of the F&L Com schemes (and an eventually needed NIZKPof correctness). These parameters need to be sent (i.e., if the outer BitCom scheme of P_A does not have trapdoor known by P_A), but such communication can be deferred to the time when P_A commits her input bits and produces the F&L BitComs of output (453).

Stage 1.1 — PRODUCE INITIAL BITCOMS OF P_B.

- **1.1.1.** Outer BitComs of input bits of P_B**.** For each input wire of P_B ($i \in I_B$), P_B selects randomness ($\sigma_i^{(b_i)}$) to commit his input bit b_i (454) and then uses the BitCom scheme $\mathscr{B}_B$ for outer BitComs to compute and send a respective BitCom σ'_i to P_A (455). If the BitComs need a ZKP of correctness, then P_B sends a respective NIZK to P_A (456). For example, if using the ElGamal BitCom scheme, then P_B would give a ZKP of correct (a.k.a. *good*) ElGamal BitComs (e.g., as in §A.3.2).
- **1.1.2.** Outer BitComs of output offsets of P_B**.**
  The circuit specification defines what is the set O_{BB} of indices of private output wires of P_B, i.e., the output wires of P_B that are not also part of the output of P_A (see (333)). For each of these private output wires of P_B ($i \in O_{BB}$), P_B selects a random offset bit d_i (457), then selects randomness ς_{O_B} to commit to the respective bitstring d_{O_B} (458), and produces the respective BitStringCom ς'_{O_B} (459). If the outer Com scheme requires a ZKP of correctness, then P_B sends a respective NIZK to P_A (460).
- **1.1.3.** Intermediate OT BitComs of input of P_B**.**
  - **If $\mathscr{B}_B \neq \mathscr{B}_{OT}$.** If the outer BitCom scheme $\mathscr{B}_B$ (i.e., including its parameters) is the same as the BitCom scheme $\mathscr{B}_{OT}$ used for OT, then the respective OT BitComs μ'_i and respective randomnesses are simply defined to be the same across the two BitCom schemes (461).
  - **If $\mathscr{B}_B \neq \mathscr{B}_{OT}$.** If the OT BitCom scheme $\mathscr{B}_{OT}$ is different from the outer BitCom scheme $\mathscr{B}_B$, then P_B produces new BitComs, by first selecting the needed randomness ν_i (462) and then using it to compute the respective BitCom μ'_i (463) for each input bit b_i of P_B. Then, P_B gives a ZKP that the BitComs (μ'_i, σ'_i) produced by the two BitCom schemes commit to the same bits (464), with this ZKP sub-protocol also proving the correctness of the BitComs. Since the outer BitCom scheme $\mathscr{B}_B$ (defined in the Setup stage) used for initial BitComs of P_B is extractable (i.e., the simulator at this point would already be able to extract the input bits of P_B), the mentioned ZKP for the OT BitCom scheme $\mathscr{B}_{OT}$ does not need to (but it may) allow extraction of the committed bits.
  - **If 2-out-of-1 OT.** If the OT is of type 2-out-of-1, then the BitComs for the OT were produced with a 2-to-1-square- scheme, of which P_A necessarily knows the trapdoor t_{OT}. In this case, P_A implements the 2-out-of-1 OT, by using the trapdoor to extract from the BitCom two openings $(\mu'^{(0)}_i, \mu'^{(1)}_i)$, out of which it is ensured that P_B knows at least one (465).

- **1.1.4.** Parse Outer BitComs of P_B.
  P_B alone (i.e., not P_A) stringizes the two vectors of outer-randomnesses (i.e., one associated with private input bits and the other with random offset bits of output) (466) Then, both parties stringize the two vectors of outer-BitComs of P_B, (467).
  Then, if the trapdoor of the BitStringCom scheme does not allow extraction of the "randomness" of commitments, then P_B sends a NIZKPoK of the opening to P_A (468). In a DLC this may be a NIZKPoK of ElGamal openings §A.3.4.

Stage CF.1 — Coin-flip Permutations (Start). P_B initiates the protocol for coin-flipping of random permutations of outer-Coms and outer-randomnesses.

- **Wires sets of P_B.** In respect to wire sets of P_B, the goal is to initiate a coin-flipping that in the end yields random outer-randomnesses to P_B and respective outer Coms of 0 to P_A. For the wire sets of P_B, P_B sends a request for a generalized coin-flipping of type 1, with the respective function being the commitment to a 0, when having as input a respective randomness (469). The request is made considering a domain for two such commitments (i.e., for I_B and O_B) (470). As a result, the ideal functionality $\mathcal{F}_{\text{GMCF-1}}$ informs P_A of the request for such coin-flipping (471). Specialized simulatable protocols for GMCF type-1 are described in Section C.1.1, for both DLC (§C.2.3) and IFC (§C.2.4) instantiations.
- **Wires sets of P_A.** In respect to wire sets of P_A the goal is vice-versa to the goal for wires of P_B. However, since P_A will still commit her initial random outer-Coms, only P_B needs to contribute (i.e., decide) the respective permutation. For each set of wires of P_A, P_B selects a random randomness $\gamma_{set}^{(B)}$ for committing 0 (472), and then commits to it, using an Ext&Equiv Com. Using an ideal functionality, this corresponds to sending the randomnesses to the functionality (473), after which the functionality sends a receipt to P_A, which simply reveals the length of the committed message (474). An efficient simulatable protocol for Ext&Equiv Com is described in Section C.1.

Stage 1.2 — Produce initial BitComs of P_A.

- **1.2.1.** Outer BitComs of input bits of P_A. For each input wire of P_A ($i \in I_A$), P_A uses her outer BitCom scheme $\mathscr{B}_A$ to commit her own input bits, by first selecting random encodings σ_i (475) and then computing the respective BitComs σ_i' (476).
  If need be (depending on the BitCom scheme), P_A also sends a NIZKP of correct BitComs

Stage 1.2. PRODUCE INITIAL BITCOMS OF $\mathrm{P_A}$.

1.2.1. OUTER BITCOMS OF INPUT BITS OF $\mathrm{P_A}$.

$\mathrm{P_A}: \sigma_i \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_\mathrm{A}](b_i) : i \in I_\mathrm{A}$ (475)

$\mathrm{P_A} \to \mathrm{P_B}: \sigma'_i = \mathscr{B}_\mathrm{A}(b_i; \sigma_i) : i \in I_\mathrm{A}$ (476)

If BitComs from $\mathscr{B}_\mathrm{A}$ need ZKP of correctness:

$\mathrm{ZKP}_{\mathrm{GoodBitComs}}[\mathscr{B}_\mathrm{A}]\left(\langle \sigma'_i : i \in I_\mathrm{A} \rangle\right)$ (477)

1.2.2. OUTER BITCOMS OF OUTPUT OFFSETS OF $\mathrm{P_A}$. (see (335))

$\mathrm{P_A}: d_i \leftarrow^{\$} \{0,1\} : i \in \mathrm{O_{AA}}$ (random bit-masks) (478)

$\mathrm{P_A}: \varsigma_i \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_\mathrm{A}](d_i) : i \in \mathrm{O_{AA}}$ (479)

$\mathrm{P_A} \to \mathrm{P_B}: \varsigma'_i = \mathscr{B}_\mathrm{A}(d_i; \varsigma_i) : i \in \mathrm{O_{AA}}$ (480)

If BitComs from $\mathscr{B}_\mathrm{A}$ need ZKP of correctness:

$\mathrm{P_A} \to \mathrm{P_B}: \mathrm{ZKP}_{\mathrm{GoodBitComs}}[\mathscr{B}_\mathrm{A}]\left(\langle \varsigma'_i : i \in \mathrm{O_{AA}} \rangle\right)$ (481)

1.2.3. INTERMEDIATE CONNECTOR BITCOMS OF INPUT OF $\mathrm{P_A}$.

$\mathrm{P_A}: b_i \equiv d_{\theta_\mathrm{A}(i)} : i \in I_\mathrm{A}$, (see (338 and 339)) (482)

If $\mathscr{B}_\mathrm{ConA} =^{?} \mathscr{B}_\mathrm{A}$, then:

$(\mu_i, \mu'_i) \equiv (\sigma_i, \sigma'_i) : i \in I_\mathrm{A}$ (483)

$(\mu_i, \mu'_i) \equiv (\varsigma_{\theta_\mathrm{A}(i)}, \varsigma'_{\theta_\mathrm{A}(i)}) : i \in I_\mathrm{A}$, (see (339)) (484)

Else if $\mathscr{B}_\mathrm{ConA} \neq \mathscr{B}_\mathrm{A}$, then:

$\mathrm{P_A}: \mu_i \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_\mathrm{ConA}](b_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (485)

$\mathrm{P_A} \to \mathrm{P_B}: \mu'_i = \mathscr{B}_\mathrm{ConA}(b_i; \mu_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (486)

If $(\mathscr{B}_\mathrm{ConA} \neq \mathscr{B}_\mathrm{A}) \wedge (\mathscr{B}_\mathrm{FLA} \in \{\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{ConA}\})$, then :

$\mathrm{P_A} \to \mathrm{P_B}: \mathrm{NIZKP}_{\mathrm{SameComBits}}[\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{ConA}]$

$\left(\langle (\sigma'_i, \mu'_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'} \rangle\right)$ (487)

If $(\mathscr{B}_\mathrm{ConA} \neq \mathscr{B}_\mathrm{A}) \wedge (\mathscr{B}_\mathrm{FLA} \notin \{\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{ConA}\})$, then:

Defer ZKP to step (496)

1.2.4. PARSE OUTER BITCOMS OF $\mathrm{P_A}$. For $set \in \{I_\mathrm{A}, I_\mathrm{A'}\}$:

$\mathrm{P_A}: \sigma_{set} \equiv \mathrm{Stringize}\left(\langle \sigma_i : i \in set \rangle\right)$ (488)

$\mathrm{P_A}, \mathrm{P_B}: \sigma'_{set} \equiv \mathrm{Stringize}\left(\langle \sigma'_i : i \in set \rangle\right)$ (489)

$\mathrm{P_A}(\sigma_{set}) \to \mathrm{P_B}: \mathrm{ZKPoK}_{\mathrm{ComOpening}}(\sigma'_{set})$ (490)

(A single ZKPoK of trapdoor suffices for GM BitComs)

1.2.5. F&L-related Coms of input bits of PA.

If $\mathscr{B}_\mathrm{FLA} \equiv \mathscr{B}_\mathrm{A}$, then: $(\phi_i, \phi'_i) \equiv (\sigma_i, \sigma'_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (491)

Else if $\mathscr{B}_\mathrm{FLA} \equiv \mathscr{B}_\mathrm{ConA}$, then:

$(\phi_i, \phi'_i) \equiv (\mu_i, \mu'_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (492)

Else if $\mathscr{B}_\mathrm{FLA} \notin \{\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{FLA}\}$, then:

$\mathrm{P_A}: \phi_i \leftarrow^{\$} \mathrm{Get}_{\mathrm{RandForBitCom}}[\mathscr{B}_\mathrm{FLA}](b_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (493)

$\mathrm{P_A} \to \mathrm{P_B}: \phi'_i = \mathscr{B}_\mathrm{FLA}(b_i; \phi_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'}$, (494)

If $(\mathscr{B}_\mathrm{FLA} \neq \mathscr{B}_\mathrm{A} = \mathscr{B}_\mathrm{ConA})$, then $\mathrm{P_A} \to \mathrm{P_B}$:

$\mathrm{NIZKP}_{\mathrm{SameComBits}}[\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{FLA}]\left(\langle (\sigma'_i, \phi'_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'} \rangle\right)$ (495)

If $\mathscr{B}_\mathrm{FLA} \notin \{\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{FLA}\}) \wedge (\mathscr{B}_\mathrm{A} \neq \mathscr{B}_\mathrm{FLA})$, then $\mathrm{P_A} \to \mathrm{P_B}$:

$\mathrm{NIZKP}_{\mathrm{SameComBits}}[\mathscr{B}_\mathrm{A}, \mathscr{B}_\mathrm{ConA}, \mathscr{B}_\mathrm{FLA}]$

$\left(\langle (\sigma'_i, \mu'_i, \phi'_i) : i \in I_\mathrm{A} \cup I_\mathrm{A'} \rangle\right)$ (e.g., §A.4.1) (496)

1.2.6. F&L-related BitComs for output bits.

For $i \in \mathrm{O_{AA}} \cup \mathrm{O_B}: \mathrm{P_A}: \left(\left(\mu_i^{(0)}, \mu_i^{(1)}\right), \mu'_i\right) \leftarrow^{\$}$

$\mathrm{Gen}_{\mathrm{PairOpenings}}[\mathscr{B}_\mathrm{FLB}, t_\mathrm{FLB}]$ (497)

$\mathrm{P_A} \to \mathrm{P_B}: \mu'_i : i \in \mathrm{O_{AA}} \cup \mathrm{O_B}$ (498)

Figure B.7: **Protocol S2PC-with-Coms (stages 1.2 and 1.3).** (See preceding in Figure B.6 and continuation in Figure B.8.) Legend in §Notation. The BitComs produced by $\mathrm{P_A}$ with $\mathscr{B}_\mathrm{FLB}$ (498) do not require a ZKP of correctness, because they will later be verified as part of the cut-and-choose structure and the homomorphic relations with the elements in connectors (namely the multipliers) (see Remark B.1).

(477). The ZKP is not needed if the BitCom scheme (e.g., GM) is such that BitComs are directly verifiable as correct even without knowing the trapdoor.

- **1.2.2.** OUTER BITCOMS OF OUTPUT OFFSETS OF $\mathrm{P_A}$. Both parties know, from the circuit specification, what is the set $\mathrm{O_{AA}}$ of indices of private output wires of $\mathrm{P_A}$, i.e., the output wires of $\mathrm{P_A}$ that are not also part of the output of $\mathrm{P_B}$ (see (333)). For each of these wires, $\mathrm{P_A}$ selects a random offset bit d_i (478), then select randomness ς_i to commit to it (479) and uses it to produce a respective outer BitCom (480). Then, if need be, $\mathrm{P_A}$ also sends a NIZKP that the BitComs are correct (481).
- **1.2.3.** INTERMEDIATE CONNECTOR BITCOMS OF INPUT OF $\mathrm{P_A}$.
  Since the private output bit-masks d_i of $\mathrm{P_A}$ were defined in association with the wire indices associated with her private output wires, $\mathrm{P_A}$ uses the appropriate mapping θ_A of wire indices (from the set of indices ($i \in I_\mathrm{A'}$) of her auxiliary input bits into the set

($i \in \mathrm{O_{AA}}$) of respective indices of her private output wires) to define that the output bit masks can also be considered as auxiliary input bits b_i associated with the adjusted set of input wires of $\mathrm{P_A}$ (482).

  - **If $\mathscr{B}_{\mathbf{ConA}} \equiv \mathscr{B}_{\mathbf{A}}$.** If the BitCom scheme ($\mathscr{B}_{\mathrm{ConA}}$) for connectors of input of $\mathrm{P_A}$ is the same as the outer BitCom scheme ($\mathscr{B}_{\mathrm{A}}$), then both parties assume that the notation is superposed and accept the BitComs (μ'_i) for the connectors to be simply be the ones already defined outer BitComs and similarly for the respective encodings (μ_i). This is considered both for input bits of $\mathrm{P_A}$ (483), as well as for the private-output offset bits of $\mathrm{P_A}$ (484).
  - **New coms if $\mathscr{B}_{\mathbf{ConA}} \neq \mathscr{B}_{\mathbf{A}}$.** If the outer Com scheme is different from the one used for connectors, then $\mathrm{P_A}$ selects respective new random encodings μ_i (for $\mathscr{B}_{\mathrm{ConA}}$) for the input bits and the private-output offset bits (485) and then computes and sends the respective BitComs μ'_i to $\mathrm{P_B}$ (486).
  - **New NIZKPs if $\mathscr{B}_{\mathbf{ConA}} \neq \mathscr{B}_{\mathbf{A}}$.** If the intermediate BitCom scheme ($\mathscr{B}_{\mathrm{ConA}}$) is different from the outer Com scheme ($\mathscr{B}_{\mathrm{A}}$), and the later BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) to support the forge and lose is equal to any of the previous two, then $\mathrm{P_A}$ sends to $\mathrm{P_B}$ a NIZKP proving that the BitComs for connectors commit to the same bits as the outer BitComs (487). If instead all three BitCom schemes are different, then the ZKP is deferred to after the third set of Coms is produced (using $\mathscr{B}_{\mathrm{FLA}}$).

- **1.2.4.** Parse Outer BitComs of $\mathrm{P_A}$**.** Similarly to what was done with the BitComs (and respective randomness) of outer bits of $\mathrm{P_B}$, now the BitComs σ'_i of bits of $\mathrm{P_A}$ are also parsed into a single BitsStringCom σ'_{I_A} (488–489).

  Then, $\mathrm{P_A}$ also sends a NIZKPoK of the openings (490). If the randomness is extractable with the trapdoor, then a single ZKPoK of trapdoor might be sufficient (e.g., for GM). For ElGamal commitments the ZKPoK can be reduced to a single ZKPoK of DL per BitStringCom (i.e., besides the ZKPoK of trapdoor and the ZKPs of good initial BitComs).

1.2.5. F&L-related Coms of input bits of PA.

- **New BitComs.** If the Ext-BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) used to support the forge-and-lose technique with respect to the input of $\mathrm{P_A}$ is the same as the outer BitCom scheme $\mathscr{B}_{\mathrm{A}}$ of $\mathrm{P_A}$, then both parties assume that the notation is superposed and accept the BitComs (μ'_i) and respective randomnesses to be the same (491). If otherwise it is equal to the Com scheme $\mathscr{B}_{\mathrm{ConA}}$ used for connectors, then the parties also accept a respective superposition

of notation (492). If the forge-and-lose BitCom scheme is different from the previous two, then P_A selects new randomness for BitComs of the input bits and of the output-offset bits (493). Then, P_A produces respective BitComs for the input bits of P_A and all output-offset output bits of P_A (494).

- **ZKPs of same committed bits.** If the outer-BitCom scheme of P_A is equal to the one used for connectors but is different from the FLA-BitCom scheme, then P_A sends a NIZK of same committed bits between the BitComs produced across the two different schemes (495). If all three BitComs schemes are different, then P_A sends a NIZKP that the BitComs across the three schemes commit to the same bits (496).

1.2.6. F&L-related BitComs for output bits. For each output wire, both of $P_B(i \in O_B)$ and P_A $(i \in O_A)$, P_A uses the trapdoor t_{FLB} of the respective intermediate BitCom scheme $\mathscr{B}_{FLB}$ (used for the forge-and-lose technique) to produce one random (equivocable) BitCom μ'_i and respective possible pairs of encodings $(\mu_i^{(0)}, \mu_i^{(1)})$, i.e., for bit 0 and 1 (497), and then sends only the BitCom μ'_i to P_B (498). This is performed before the respective output bits are computed, but the protocol will later ensure that (in a successful execution) P_B receives one corresponding opening per output wire.

Remark B.1 (Intermediary BitComs of output bit of PB for forge-and-lose do not need ZKP of correctness). After P_A defines the intermediate BitComs for the output bits of P_B (498), a ZKP of correct BitComs is not needed because the BitComs will later be verified as part of the cut-and-choose structure and the homomorphic relations with the elements in connectors (namely the multipliers). This is not a general statement, but applies to Blum and Pedersen BitComs, for which it is possible to assume that the correctness of a BitCom can be directly verified without knowledge of the trapdoor (i.e., assuming that the Blum integer has already been proven correct, and that in the Pedersen scheme the group order can be computed). For Blum integers this assumes a reduction to *proper* group elements, such that all group elements (even actual non-quadratic residues in the regular group representation) can be considered as pseudo-squares and thus having two non-trivially correlatable square-roots.

Stage 2 — Commit.

For each of the s challenge indices $j \in [s]$, P_A randomly selects an appropriate random seed

λ_j (499) to be used as input in the secure pseudo-randomness generators (PRGs) associated with the RSC technique that follows. Also, $\mathrm{P_A}$ pseudo-randomly generates an auxiliary seed ($\lambda_j^{(auxi)}$) using as seed the initial random RSC seed (500).

- **Connectors of Input $\mathbf{P_A}$.**
  - **Group elements.** For each input wire of $\mathrm{P_A}$ ($i \in I_\mathrm{A}$): $\mathrm{P_A}$ generates a pseudo-random permutation bit $\pi_{j,i}$ (501). Then, using the BitCom scheme $\mathscr{B}_{\mathrm{ConA}}$ for the connectors of $\mathrm{P_A}$ it selects a pseudo-random encoding $\alpha_{j,i}$ (a group element, dubbed *multiplier*) (502) of the permutation bit and then computes the respective BitCom $\alpha'_{j,i}$ (503). Then, $\mathrm{P_A}$ combines the multiplier BitCom ($\alpha'_{j,i}$) and the outer BitCom (μ'_i) of the connectors, in either a XOR-homomorphic or a pseudo XOR-homomorphic manner, (depending on the type of homomorphism allowed by the BitCom scheme), in order to obtain an *inner Bitcom* ($\nu'_{j,i}$) that serves as BitCom of the XOR-permuted bit (504), (i.e., it commits to the XOR-combination of the permutation bit $\pi_{j,i}$ with the input bit b_i of $\mathrm{P_B}$). (Figure B.8 describes the pseudo-homomorphic version ($*'^{(\pi)}$) that applies to both cases.)
  - **Circuit Keys and their commitments.** For each input wire of $\mathrm{P_A}$, $\mathrm{P_A}$ also generates two pseudo-random circuit input keys ($k_{j,i}^{[c]} : c \in \{0,1\}$) (506). $\mathrm{P_A}$ pseudo-randomly generates the "randomness" $\underline{k}_{j,i}^{[c]}$ needed for their respective commitments (505), and then computes the respective two commitments $\overline{k}_{j,i}^{[c]}$ (507); finally, $\mathrm{P_A}$ reorders the commitments in each pair (508), based on the generated permutation bit ($\pi_{j,i}$), and then aggregates all such ordered pairs of commitments in a vector (509).
- **Connectors of Input $\mathbf{P_B}$.**

  For each input wire of $\mathrm{P_B}$ ($i \in I_\mathrm{B}$):
  - **If OTs are at the BitCom level (and are of 2-out-of-1 type).** (I.e., if the respective BitCom scheme $\mathscr{B}_{\mathrm{OT}}$ is a 2-to-1-square scheme, from which $\mathrm{P_A}$ has already extracted two square-roots from each BitCom of $\mathrm{P_B}$,) $\mathrm{P_A}$ generates two pseudo-random independent group elements ($\nu_{j,i,c}$), dubbed *inner encodings*, respectively encoding bits 0 and 1 (510). Then, $\mathrm{P_A}$ uses each inner encoding as "randomness" to generate respective inner BitComs (511).
  - **If OTs are at the GC-InWires level (and are of 1-out-of-2 type).** $\mathrm{P_A}$ starts by producing a triplet composed of two random coefficients ($\beta_{j,i,0}, \beta_{j,i,1}$) for a linear homomorphic transformation and an additional randomizer element $r_{j,i}$ for a randomization of a commitment (512). Then, $\mathrm{P_A}$ uses the triplet as input of a respective *randomized linear homomorphic transformation* operation, to produce a new commitment $\nu'_{j,i}$, de-

Stage 2. COMMIT. $\mathrm{P_A}$:

2.1. Select RSC and auxiliary seeds. $\mathrm{P_A}$:

$$\lambda_j \leftarrow^{\$} \mathrm{Gen}_{\mathrm{Seed}}(\kappa_{\mathrm{PRG}}) : j \in [s] \quad (499)$$

$$\lambda_j^{(auxi)} = \mathrm{PRGen}_{\mathrm{AuxiSeed}}[\lambda_j][0] : j \in [s] \quad (500)$$

2.2. Connectors input $\mathbf{P_A}$. $\mathrm{P_A}$:

(Permutation bits.)

$$\langle \pi_{j,i} : i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'} \rangle = \mathrm{PRGen}_{\mathrm{BitString}}[\lambda_j][1](\ell_{\mathrm{A}} + \ell'_{\mathrm{A}}) : j \in [s] \quad (501)$$

(Group elements.)

For $j \in [s], i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'}$:

$$\alpha_{j,i} = \mathrm{PRGen}_{\$\mathrm{ForCom}}[\lambda_j][2;i](\mathscr{B}_{\mathrm{ConA}}, \pi_{j,i}) \quad (502)$$

$$\alpha'_{j,i} = \mathscr{B}_{\mathrm{ConA}}(\pi_{j,i}; \alpha_{j,i}) \quad (503)$$

$$\nu'_{j,i} = \alpha'_{j,i} \star'_{(\pi_{j,i})} \mu'_i \quad (504)$$

(Input-keys and commitments.)

For $j \in [s]$: (see Remark B.7 in §B.5.1)

$$\underline{k}_j = \mathrm{PRGen}_{\$\mathrm{ForCom}}[\lambda_j][3;i](\mathscr{C}_{\mathrm{InKey}}) \quad (505)$$

For $j \in [s], i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'}, c \in \{0,1\}$:

$$k_{j,i}^{[c]} = \mathrm{PRGen}_{\mathrm{InKey}}[\lambda_j][4;i](c) \quad (506)$$

$$\overline{k}_{j,i}^{[c]} = \mathscr{C}_{\mathrm{InKey}}\left(k_{j,i}^{[c]}; \underline{k}_j\right) \quad (507)$$

$$\langle c \rangle \equiv [c \oplus \pi_{j,i}] \quad (508)$$

$$\overline{\mathrm{InKeys}}_{A,j}^{(2)} = \left\langle \left(i, \bar{k}_{j,i}^{\langle 0 \rangle}, \bar{k}_{j,i}^{(1)}\right) : i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'} \right\rangle : j \in [s] \quad (509)$$

2.3. Connectors input $\mathbf{P_B}$. $\mathrm{P_A}$:

If $\left(\mathrm{OT}_{\mathrm{Level}}, \mathrm{OT}_{\mathrm{Type}}\right) = (\mathrm{BitComs}, 2/1)$:

For $j \in [s], i \in I_{\mathrm{B}}, c \in \{0,1\}$:

$$\nu_{j,i,c} = \mathrm{PRGen}_{\$\mathrm{ForCom}}[\lambda_j][5;i](\mathscr{B}_{\mathrm{OT}}, c) \quad (510)$$

$$\left(\nu_{j,i,c} \equiv \nu_{j,i,c}^{(c)}\right)$$

$$\nu'_{j,i,c} = \mathscr{B}_{\mathrm{OT}}\left(c; \nu_{j,i,c}\right) \quad (511)$$

If $\left(\mathrm{OT}_{\mathrm{Level}}, \mathrm{OT}_{\mathrm{Type}}\right) = (\mathrm{GC\text{-}InWires}, 1/2)$:

For $j \in [s], i \in I_{\mathrm{B}}$:

$$(\beta_{j,i,0}, \beta_{j,i,1}, r_{j,i}) = \mathrm{PRGen}_{\$\mathrm{ForLHT}}[\lambda_j][6;i](\mathscr{C}_{\mathrm{OT}}) \quad (512)$$

$$\nu'_{j,i} = \mathrm{RandLHT}\left[\beta_{j,i,0}, \beta_{j,i,1}, r_{j,i}\right]\left(\mu'_i\right) \quad (513)$$

$$\nu_{j,i,c} \equiv \nu_{j,i,c}^{(c)} = f(\beta_{j,i,c}) \left(\equiv \mathscr{E}^{-1}(\nu'_{j,i})\right) : c \in \{0,1\} \quad (514)$$

For $j \in [s], i \in I_{\mathrm{B}}, c \in \{0,1\}$:

$$k_{j,i}^{[c]} = \mathrm{PRGen}_{\mathrm{InKey}}\left[\lambda_j^{(auxi)}\right]\left[\nu_{j,i,c}^{(c)}\right](c) \quad (515)$$

2.4. Garbled circuits (GCs). $\mathrm{P_A}$:

For $j \in [s]$:

$$\mathrm{InKeys}_j^{(2)} \equiv \left\langle \left(i, c, k_{j,i}^{[c]}\right), i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'} \cup I_{\mathrm{B}}, c \in \{0,1\} \right\rangle \quad (516)$$

$$\left\langle GC_j, \mathrm{OutKeys}_j^{(2)} \right\rangle = \mathrm{PRGen}_{\mathrm{GC}}\left[\lambda_j\right][7]\left(C, \mathrm{InKeys}_j^{(2)}\right) \quad (517)$$

$$\left\langle \left(i, c, k_{j,i}^{[c]}\right) : i \in \mathrm{O_A} \cup \mathrm{O_B}, c \in \{0,1\} \right\rangle \equiv \mathrm{OutKeys}_j^{(2)} \quad (518)$$

2.5. Connectors Output $\mathbf{P_B}$. $\mathrm{P_A}$:

For $i \in \mathrm{O_B} \cup \mathrm{O_A}$ and $c \in \{0,1\}$:

$$\nu_{j,i,c}^{(c)} = \mathrm{PRGen}_{\$\mathrm{ForCom}}\left[\lambda_j^{(auxi)}\right]\left[k_{j,i}^{[c]}\right](\mathscr{B}_{\mathrm{FLB}}, c) \quad (519)$$

$$\nu'_{j,i,c} = \mathscr{B}_{\mathrm{FLB}}(c; \nu_{j,i,c}) \quad (520)$$

2.6. Global hash and RSC commitment.

$\mathrm{P_A}$: For $j \in [s]$:

$$N'_{\mathrm{InA},j} = \left\langle (i, \nu'_{j,i}) : i \in I_{\mathrm{A}} \cup I_{\mathrm{A}'} \right\rangle \quad (521)$$

$$N'_{\mathrm{InB},j} = \left\langle (i, c, \nu'_{j,i,c}) : i \in I_{\mathrm{B}}, c \in \{0,1\} \right\rangle \quad (522)$$

$$N'_{\mathrm{OutB},j} = \left\langle (i, c, \nu'_{j,i,c}) : i \in \mathrm{O_A} \cup \mathrm{O_B}, c \in \{0,1\} \right\rangle \quad (523)$$

$$N'_j = (N'_{\mathrm{InA},j}, N'_{\mathrm{InB},j}, N'_{\mathrm{OutB},j}) \quad (524)$$

$$R_j = \left(GC_j, \overline{\mathrm{InKeys}}_{A,j}^{(2)}, N'_j, \lambda_j^{(auxi)}\right) \quad (525)$$

$$\mathrm{P_A} : \Lambda = \text{CR-Hash}\left(\langle R_j : j \in [s] \rangle\right) \quad (526)$$

$$\mathrm{P_A} : \underline{\Lambda} \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{C}_{\mathrm{RSC}}^{\mathrm{Equiv}}](\Lambda) \quad (527)$$

$$\mathrm{P_A} \rightarrow \mathrm{P_B} : \overline{\Lambda} = \mathscr{C}_{\mathrm{RSC}}^{\mathrm{Equiv}}(\Lambda; \underline{\Lambda}) \quad (528)$$

Figure B.8: **Protocol S2PC-with-Coms (stage 2).** (See preceding in Figure B.7 and continuation in Figure B.9.) Legend in §Notation.

noted *inner* commitment, of the linear coefficient β_{j,i,b_i} with (third) index equal to the bit b_i committed by the initial BitCom μ_i of $\mathrm{P_B}$ (513). It is worth noticing that this is no longer a BitCom, but rather a commitment of an element from a larger set. The commitment scheme $\mathscr{B}_{\mathrm{OT}}$ used herein for (1-out-of-2) OTs is the composition of an encryption scheme $\mathscr{E}$ and a 1-to-1 function f (with the decryption key being known by $\mathrm{P_B}$ and not by $\mathrm{P_A}$). Thus, the commitment can also be seen as an encryption of the

f-image of the coefficient β_{j,i,b_i}. In the case of ElGamal encryption this is the encryption of the base generator g_0 (of the ElGamal scheme) raised to the power of the coefficient β_{j,i,b_i}. Then, $\mathrm{P_A}$ computes the two possible values that may be encrypted, denoted inner encodings, i.e., the f-images of the two coefficients (514).

  - **For any OT type.** Regardless of the OT type, $\mathrm{P_A}$ uses the auxiliary seed $(\lambda_j^{(auxi)})$ and each inner encoding $\nu_{j,i,c}$ as a seed to pseudo-randomly generate a respective circuit input key $k_{j,i}^{[c]}$ (515).

- **Garbled circuits.** $\mathrm{P_A}$ combines a sequence of all generated pairs of circuit input keys (516) and uses them to generate a GC which accepts those keys in the respective input wires (517). As part of building the GC, $\mathrm{P_A}$ also obtains the respective pairs of output keys (518).
- **Connectors of Output.** For each circuit output wire of the GC ($i \in \mathrm{O_B}$), $\mathrm{P_A}$ uses a PRG procedure to produce, from each output key $(k_{j,i}^{[c]})$, and respective underlying bit, one respective *inner encoding* $(\nu_{j,i,c}^{(c)})$ (519) (i.e., a group element encoding the bit underlying the respective output key) that can be used as "randomness" to produce a respective BitCom $\nu'_{j,i,c}$. In order to guarantee enough entropy (important against "birthday attacks"), the PRG procedure uses as additional seed the previously computed auxiliary seed $(\lambda_j^{(auxi)})$ (520).
- **Aggregate info, compute global hash and RSC commitment.** Intuitively, the GC, the key commitments and the inner BitComs computed above constitute all elements that should be sent to $\mathrm{P_B}$ if the protocol was not using a RSC technique. For each challenge index (j), $\mathrm{P_A}$ aggregates all inner BitComs of input wires of $\mathrm{P_A}$ (521), all inner BitComs of input wires of $\mathrm{P_B}$ (522), all inner BitComs of output wires of $\mathrm{P_B}$ (523); and then aggregates them all together (524). For each challenge index, $\mathrm{P_A}$ aggregates all elements into a single tuple R_j (525), containg the garbled circuit, all respective pairs of ordered commitments of input keys of $\mathrm{P_A}$ and all inner BitComs (525). In order to enable communication reduction (i.e., via the RSC technique), and also to enable (in a simple way) equivocability of the committed values, $\mathrm{P_A}$ joins these elements into a single tuple and computes a respective (compressive) CR-Hash Λ, hereinafter denoted global hash (526). Finally, $\mathrm{P_A}$ selects randomness $\underline{\Lambda}$ (527) needed to commit to the global hash, and then sends a respective non-malleable Equiv-Com $\overline{\Lambda}$ to $\mathrm{P_B}$ (528).

Stage 3 — Challenge.

The set of possible cut-and-choose partitions is defined by the C&C parameters implicitly agreed in the SETUP stage (529).

If the cut-and-choose selection style is interactive with selection by P_B (i.e., INT-B), then P_B randomly selects a cut-and-choose partition from within the set of allowed partitions, i.e., satisfying the constraints imposed by the C&C parameters implicitly agreed in the SETUP stage, and sends it to P_A, thus defining a subset J_V of indices for check and a complementary subset of evaluation J_E indices (530).

If the selection style is instead non-interactive with selection by P_A (NI-A), and in this case the statistical security parameter must equate the computational security parameter, then P_A calculates the partition pseudo-randomly as the hash output of a non-programmable oracle, using as input the execution context, the CRS and the RSC Equiv Com (531). (Another non-interactive alternative (NI-B) is possible with P_B deciding the cut-and-choose partition before the commit stage in and committing to it using an oblivious transfer §2.4.2.)

The party that selected the cut-and-choose partition sends it to the other party (532). Then, the receiver of the partition (P_A if INT-PB, P_B if NI-PA) locally verifies that the subsets are disjoint and complementary (533), and that they satisfy the implicitly agreed restrictions on the minimum and maximum numbers of each challenge type (534).

Stage CF.2 — COIN-FLIP PERMUTATIONS (CONTINUE)**.**

As a continuation of the coin-flipping protocol previously initiated by P_B, and in regard to the wire sets of P_B, P_A sends a respective message to the ideal coin-flipping functionality (535), which leads the ideal functionality to send the outer-randomness permutations γ to P_B (536). From these values, P_B is already able to compute the respective outer-com permutations γ' (537).

It is noteworthy that P_A does not have to send any contribution related to the wires of P_A, because the initial outer-Coms of P_A are already random (i.e., in the view of a possibly malicious P_B, if P_A is honest). Thus, the outer-Coms of P_A are what need to be permuted by a random contribution (already committed) from P_B alone.

While some logical stages can be interleaved, this coin-flipping component (i.e., the communication part of it, from P_A to P_B) must occur in conjunction with the communication, from P_A to P_B, of the RESPOND stage, so that P_B learns the circuit output and the respective final outer-coms at the same time (i.e., without any further communication being required in

Stage 3. CHALLENGE **(cut-and-choose).**
(see §2.4.2 for alternatives in non-interactive setting)

$\Pi \equiv \text{PARTITIONS}[(v_{\min}, v_{\max}), (e_{\min}, e_{\max})][s]$ (529)

If INT-B $=^?$ C&C-STYLE, then: $((\text{P}_\text{S}, \text{P}_\text{R}) \equiv (\text{P}_\text{B}, \text{P}_\text{A}))$

$\text{P}_\text{B} : (J_V, J_E) \leftarrow^\$ \Pi$ (530)

If NI-A $=^?$ C&C-STYLE, then: $((\text{P}_\text{S}, \text{P}_\text{R}) \equiv (\text{P}_\text{A}, \text{P}_\text{B}))$

$\text{P}_\text{A} : (J_V, J_E) = \text{NPRO}[ctx, CRS, \overline{\Lambda}](\Pi)$ (531)

$\text{P}_\text{S} \to \text{P}_\text{R} : (ctx, \texttt{c\&c-partition}, (J_V, JE))$ (532)

$\text{P}_\text{R} : J_V \cap J_E =^? \varnothing,\ J_V \cup J_E =^? [s]$ (533)

$\text{P}_\text{R} : e_{\min} \le^? e \equiv |J_E| \le^? e_{\max},\ v_{\min} \le^? v \equiv |J_V| \le^? v_{\max}$ (534)

CF.2. COIN-FLIP PERMUTATIONS (CONTINUE)**.**

$\text{P}_\text{A} \to \mathcal{F}_{\text{GMCF-1}} : (\texttt{in-2}, ctx_{\texttt{cf}})$ (see (470)) (535)

$\mathcal{F}_{\text{GMCF-1}} \to \text{P}_\text{B} : (\texttt{out-1}, ctx_{\texttt{cf}}, (\gamma_{I_\text{B}}, \gamma_{\text{O}_\text{B}}))$ (536)

$\text{P}_\text{B} : \gamma'_{set} = \mathscr{C}_\text{B}\left(0^{\#(set)}; \gamma_{set}\right) : set \in \{I_\text{B}, \text{O}_\text{B}\}$ (537)

Stage 4. RESPOND. $\text{P}_\text{A} \to \text{P}_\text{B}$:

4.1. ***Check*** **indices.** $\lambda_j : j \in J_V$ (RSC seeds) (538)

4.2. ***Evaluation*** **indices.** For $j \in J_E$:

$\lambda_j^{(auxi)}$ (auxiliary seed — **not the RSC seed**) (539)

GC_j (garbled circuit) (540)

4.2.1. Connectors of Input of P_A.

$c_{j,i} = b_i \oplus \pi_{j,i} : i \in I_\text{A}$ (541)

$c_{j,i} = d_i \oplus \pi_{j,i} : i \in I_\text{A}$, (542)

$\nu_{j,i} \equiv \nu_{j,i}^{(c_{j,i})} = \alpha_{j,i} \overset{\ddagger}{*}_{(\pi_{j,i})} \mu_i^{(b_i)} : i \in I_\text{A} \cup I_\text{A}$, (543)

$\left(\xi_{j,i}, \underline{\xi}_{j,i}\right) \equiv \left(k_{j,i}^{\langle c_{j,i}\rangle}, \underline{k}_{j,i}^{\langle c_{j,i}\rangle}\right) : i \in I_\text{A} \cup I_\text{A}$, (544)

$\overline{k}_{j,i}^{[1\oplus b_i]} \equiv \overline{k}_{j,i}^{\langle 1\oplus c_{j,i}\rangle} : i \in I_\text{A} \cup I_\text{A}$, (545)

4.2.2. Connectors of Input of P_B. For $i \in I_\text{B}$:

If $(\text{OT}_\text{Level}, \text{OT}_\text{Type}) = (\text{BitComs}, 2/1)$:

$\beta_{j,i,c} \equiv \beta_{j,i,c}^{(0)} = inv\left(\mu_i^{(c)}\right) * \nu_{j,i,c}^{(c)} : c \in \{0,1\}$ (546)

If $(\text{OT}_\text{Level}, \text{OT}_\text{Type}) = (\text{GC-InWires}, 1/2)$:

$\nu'_{j,i} \left(\equiv \mathscr{E}(f(\beta_{b_i}))\right)$ (547)

4.2.3. Connectors of Outputs.

For $i \in \text{O}_\text{B} \cup \text{O}_\text{A}, c \in \{0,1\}$:

$\beta_{j,i,c} \equiv \beta_{j,i,c}^{(0)} = inv\left(\nu_{j,i,c}^{(c)}\right) * \mu_i^{(c)}$ (548)

4.3. Open global hash. $(\Lambda, \underline{\Lambda})$ (549)

Stage 5. VERIFY. P_B:

5.1. ***Check*** **indices.** For $j \in J_V$:

$\lambda_j^{(auxi)} = \text{PRGen}_{\text{AuxiSeed}}[\lambda_j][0]$ (as in (500)) (550)

Do steps (501)-(524) to get $\left(GC_j, \overline{\text{InKeys}}_{A,j}^{(2)}, N'_j\right)$ (551)

5.2. ***Evaluation*** **indices.** For $j \in J_E$:

5.2.1. Connectors of input of P_A. For $i \in I_\text{A} \cup I_\text{A}$, :

$\overline{\xi}_{j,i} \equiv \overline{k}_{j,i}^{\langle c_{j,i}\rangle} = \mathscr{C}_{\text{InKey}}\left(\xi_{j,i};\ \underline{\xi}_{j,i}\right)$ (552)

$\nu'_{j,i} = \mathscr{B}_{\text{ConA}}(c_{j,i};\ \nu_{j,i})$ (553)

5.2.2. Connectors of input of P_B. For $i \in I_\text{B}$:

If $(\text{OT}_\text{Level}, \text{OT}_\text{Type}) = (\text{BitComs}, 2/1)$:

$\beta'_{j,i,c} = \mathscr{B}_{\text{OT}}(0;\ \beta_{j,i,c}) : c \in \{0,1\}$ (554)

$\nu'_{j,i,c} = \mu'_i *' \beta'_{j,i,c} : c \in \{0,1\}$ (555)

If $(\text{OT}_\text{Level}, \text{OT}_\text{Type}) = (\text{GC-InWires}, 1/2)$:

(do nothing — P_B knows $\nu'_{j,i}$ from (547)) (556)

5.2.3. Connectors of outputs. For $i \in \text{O}_\text{B} \cup \text{O}_\text{A}$:

$\beta'_{j,i,c} = \mathscr{B}_{\text{FLB}}(0;\ \beta_{j,i,c}) : c \in \{0,1\}$ (557)

$\nu'_{j,i,c} = \mu'_i *' (1/\beta'_{j,i,c}) : c \in \{0,1\}$ (558)

5.3. Recompute and verify global hash.

Λ^* : (as in (526), but using (550)–(551) for $j \in J_V$, and using (539), (540), (541), (542) (545) (552), (553), (555), and (558) for $j \in J_E$) (559)

$\Lambda =^? \Lambda^*$ (i.e., opened (549) vs. regenerated (559)) (560)

Figure B.9: **Protocol S2PC-with-Coms (stages 3, CF.2, 4 and 5).** (See preceding in Figure B.8 and continuation in Figure B.10.) Legend in §Notation.

the opposite direction).

Stage 4 — RESPOND.

- **Check indices.** For each check index ($j \in J_V$), P_A simply reveals the respective random RSC seed λ_j (538).
- **Evaluation indices.** For each evaluation index ($j \in J_E$), P_A does not send the RSC seed, but sends the elements derived therefrom in the COMMIT stage, as follows.
  - The auxiliary (pseudo-randomly generated) PRG seed $\lambda_j^{(auxi)}$ (539)

- The garbled circuit GC_j (540).
- **For each input wire of $\mathbf{P_A}$:** the permuted bit $c_{j,i}$, both for the original input wires (541) and for the adjusted input wires (542); the respective inner encoding $\nu_{j,i}$ (543), computed as the pseudo-homomorphic product between the multiplier $\alpha_{j,i}$ and the outer BitCom μ_i; the circuit input key $\xi_{j,i} \equiv k_{j,i}^{[b_i]}$ corresponding to the input bit b_i of $\mathrm{P_A}$, and the randomness $\underline{\xi}_{j,i} \equiv \underline{k}_{j,i}^{[b_i]}$ used to generate the respective key commitment (544); and the commitment $\overline{k}_{j,i}^{[1 \oplus b_i]}$ of the circuit input key corresponding to the complementary bit (545);
- **For each input wire of $\mathbf{P_B}$**, the response depends on the level and type of OT. For each input wire of $\mathrm{P_B}$:
  - If the OT is at the level of BitComs and of 2-out-of-1 type, then $\mathrm{P_A}$ sends to $\mathrm{P_B}$ the two multipliers $\beta_{j,i,c}$ that lead the two outer encodings $\mu_{j,i}$ (the two possible openings of the input BitComs of $\mathrm{P_B}$), of which $\mathrm{P_B}$ only knows one, into the respective two inner encodings $\nu_{j,i,c}$. In this case, each multiplier is obtained by taking the multiplicative inverse of the respective outer encoding and then computing the group product by the respective inner encodings (546).
  - If the OT is at the level of GC-InWires and of 1-out-of-2 type, then $\mathrm{P_A}$ sends to $\mathrm{P_B}$ the single inner commitment $\nu'_{j,i}$ (547), which $\mathrm{P_B}$ has previously computed as a randomized linear homomorphic transformation (513) of the outer BitCom μ'_i. In the perspective of $\mathrm{P_A}$, the resulting inner commitment is an encryption of a function f of the linear coefficient (selected by $\mathrm{P_A}$) of degree equal to the bit committed by $\mathrm{P_B}$.
- **For each output wire of $\mathbf{P_B}$:** the two multipliers $\beta_{j,i,c}$ that lead the two inner encodings $\nu_{j,i,c}$ into the respective two outer encodings (the opening of the output BitComs of $\mathrm{P_B}$), of which $\mathrm{P_B}$ does not yet know any (548).

- **Open global hash.** Finally, $\mathrm{P_A}$ opens the committed global hash Λ, by revealing its value and the randomness $\underline{\Lambda}$ used to commit it (549) Note: this must be done using an equivocable commitment scheme, in order to allow the simulator in the proof of security to open any value in the later Respond stage.

Stage 5 — Verify.

- **Check indices.** For each check index ($j \in J_V$), $\mathrm{P_B}$ uses the respective RSC seed λ_j to locally regenerate the auxiliary PRG seed $\lambda_j^{(auxi)}$ (550) and recompute all remaining elements that were used to prepare the RSC commitment Λ (see steps (501) through

(524)). Specifically, for each check index, P_B regenerates the GC, regenerates the permuted pairs of commitments of input keys of P_A and combines them into a tuple ($\overline{\text{InKeys}}^{(2)}_{A,j}$), and regenerates the inner BitComs of connectors of wires of P_A and P_B and combines them into a tuple (N'_j) (551).

- **Evaluation indices.** For each evaluation index ($j \in J_E$), P_B needs to reconstruct the same elements, but without knowing the RSC seed. Instead, P_A uses the other elements received from P_A, as follows, for each evaluation index:
  - **Connectors of Input P_A.** For each input wire of P_A ($i \in I_A$), P_B computes: the commitment $\overline{\xi}_{j,i}$ of the received input key $\xi_{j,i}$ (also using the received associated randomness $\underline{\xi}_{j,i}$) (552); then it computes the inner BitCom $\nu'_{j,i}$ corresponding to the received inner encoding $\nu_{j,i}$ (of the permuted bit) (553)
  - **Connectors of Input of P_B.** The check operation for connectors of input bits of P_B depends on the level and type of OT. For each input wire of P_B ($i \in I_B$):
    - **If the OT is at the level of BitComs and of 2-out-of-1 type:** P_B uses the two received multipliers $\beta_{j,i,c}$ as randomnesses associated with bit 0, to compute respective BitComs (554). If using the original Blum BitCom scheme notation, then P_B must also verify that the multipliers are in class 0; if instead using the alternative description the mapping from randomness to BitCom already explicitly depends on the bit being committed. Then, P_B uses the obtained BitComs to obtain the respective two inner BitComs $\nu'_{j,i,c}$ (both of bit 0), by multiplying the single (outer) input BitCom μ'_i with the respective multiplier BitComs $\beta'_{j,i,c}$ (555).
    - **If the OT is at the level of GC-InKeys and of 1-out-of-2 type:** P_B does not need to do any specific verification (556), because it has already received from P_A the inner commitment $\nu'_{j,i}$ (547) (which is what is needed to produce the global RSC hash).
  - **Connectors of Output of P_B.** For each output wire of P_B ($i \in I_B$): P_B uses the received multipliers $\beta_{j,i,c}$ as randomness to compute respective BitComs $\beta'_{j,i,c}$ of 0 (557) (see Remark B.2). Then, P_B computes the multiplicative inverse of these BitComs and multiplies each of them by the single intermediate output BitCom μ'_i, thus obtaining the respective inner BitComs $\nu'_{j,i,c}$ (558).

Remark B.2 (group operations in the randomness space). In an IFC instantiations, when using the Blum BitCom scheme (e.g., for the input connectors of $P_B$in case

of 2/1-OT, and for output connectors of P_B), a special care is required in respect to the operations in the randomnesses space. For example, if using the original description of Blum BitComs, then P_B must check that the multipliers β sent by P_A have the expected class 0, i.e., that they have Jacobi symbol 1 (554, 557). If instead using the alternative description, then Jacobi symbol computations can be avoided, because of the commitment operation explicit dependency of the committed bit (511,520,569) In that case the homomorphic operations in the randomness space also depend explicitly on the committed bits. For example, multiplying two randomnesses implies further multiplying the auxiliary element z if and only if both bits underlying the two original randomnesses are equal to 1 (this is however not needed for the multipliers associated with bit 0). (546, 548) are indeed treated as randomness related to BitComs of 0. Also, the homomorphic inverse of a randomness involves an extra division by the auxiliary group element z if the underlying bit is 1 (546,548).

- **Recompute and verify global hash.** Using the reconstructed elements, P_B computes the respective global hash, which essentially depends on the garbled circuits GC_j (for all indices), the inner BitComs $\nu'_{j,i}$ for all input and output bits, and the pair of commitments $\overline{k}^{\langle c \rangle}_{j,i}$ of the keys of input wires of P_A (559). Finally, P_B verifies that the global hash previously opened by P_A (549) is equal to the one just reconstructed (560).

  If throughout this VERIFY stage any verification has failed, then P_B outputs `abort`.

Stage 6 — EVALUATE.

- **Compute keys of input wires of P_B.** For each evaluation index ($j \in J_E$) and each input wire of P_B ($i \in I_B$), P_B computes the inner encoding ν_{j,i,b_i} corresponding to its own input bit, as follows:
  - If the OT level is BitComs and the type is 2-out-of-1, then P_B multiplies the known outer encoding $\mu^{(b_i)}_{j,i}$ (the BitCom opening that encodes the private input bit of P_B) with the respective received multiplier $\beta^{(0)}_{j,i,b_i}$ (561), thus obtaining the respective inner encoding $\nu^{b_i}_{j,i,b_i}$.
  - If the OT level is GC-InWires and the type is 1-out-of-2, then P_B simply takes the inner commitment $\nu'_{j,i}$ and decrypts its content, thus obtaining the respective inner encoding ν_{j,i,b_i} (562), which is the f-image of one of the linear coefficients $\beta_{j,i,c}$ used by P_A when homomorphically transforming the outer BitCom of the input bit of P_B.

Stage 6. EVALUATE. $\mathrm{P_B}$:

6.1. Compute keys of input wires of $\mathrm{P_B}$.

For $j \in J_E, i \in I_\mathrm{B}$:

If $(\mathrm{OT_{Level}}, \mathrm{OT_{Type}}) = (\mathrm{BitComs}, 2/1)$:

$$\nu_{j,i,b_i} = \mu_i^{(b_i)} * \beta_{j,i,b_i}^{(0)} \tag{561}$$

If $(\mathrm{OT_{Level}}, \mathrm{OT_{Type}}) = (\mathrm{GC\text{-}InWires}, 1/2)$:

$$\nu_{j,i,b_i} = \mathscr{E}^{-1}\left(\nu'_{j,i}\right) (\equiv f(\beta_{j,i,b_i})) \tag{562}$$

$$\xi_{j,i} \equiv k_{j,i}^{[b_i]} = \mathrm{PRGen_{InKey}}\left[\lambda_j^{(auxi)}\right]\left[\nu_{j,i,b_i}^{(b_i)}\right](b_i) \tag{563}$$

6.2. Evaluate GCs. For $j \in J_E$:

$$\mathrm{InKeys}_j^{(1)} \equiv \langle (i, \xi_{j,i}) : i \in I_\mathrm{A} \cup I_{\mathrm{A}'} \cup I_\mathrm{B} \rangle \tag{564}$$

$$\langle (i, \xi_{j,i}) : i \in \mathrm{O_A} \cup \mathrm{O_B} \rangle = \mathrm{GC_{Eval}}\left(GC_j, \mathrm{InKeys}_j^{(1)}\right) \tag{565}$$

6.3. Get output encodings. For $j \in J_E$:

$$J_\mathrm{Ignore} = \varnothing \tag{566}$$

For $j \in J_E$ and $i \in \mathrm{O_A} \cup \mathrm{O_B}$:

$$c_{j,i} = \epsilon(\xi_{j,i}) \tag{567}$$

$$v_{j,i} = \mathrm{PRGen_{\$ForCom}}\left[\lambda_j^{(auxi)}\right][\xi_{j,i}]\left(\mathscr{B}_\mathrm{B}, c_{j,i}\right) \tag{568}$$

If $\mathscr{B}_\mathrm{FLB}(c_{j,i}; v_{j,i}) \stackrel{?}{=} \nu'_{j,i,c_{j,i}}$ (see (558)) (569)

then $u_{j,i} = v_{j,i} \ddagger_{(c_{j,i})} \beta_{j,i,c_{j,i}}$ (570)

else $J_\mathrm{Ignore} = J_\mathrm{Ignore} \cup \{j\}$ (571)

6.4. Check for inconsistencies.

$$\text{For } i \in \mathrm{O_A} \cup \mathrm{O_B} : z_i = \cup_{j \in J_E \setminus J_\mathrm{Ignore}} \{(c_{j,i}, u_{j,i})\} \tag{572}$$

$$\text{If } \max\{\#(z_i) : i \in \mathrm{O_A} \cup \mathrm{O_B}\} \stackrel{?}{=} 1 : \tag{573}$$

$$\text{then } \left(b_i, \mu_i^{(b_i)}\right) \leftarrow z_i : i \in \mathrm{O_A} \cup \mathrm{O_B} \text{ (regular path)} \tag{574}$$

$$\text{Else enter Forge-and-Lose path:} \tag{575}$$

$$t_\mathrm{FLA} = \mathrm{Extract_{Trapdoor}}(\langle z_i : i \in \mathrm{O_A} \cup \mathrm{O_B} \rangle) \tag{576}$$

$$b_i = \mathrm{Extract_{Bit}}[t_\mathrm{FLA}](\phi'_i) : i \in I_\mathrm{A} \cup I_{\mathrm{A}'} \tag{577}$$

$$\langle (i, b_i) : i \in \mathrm{O_B} \rangle \,||\, \langle (i, e_i) : i \in \mathrm{O_{AA}} \rangle = C'\left(\langle (i, b_i) : i \in I_\mathrm{A} \cup I_{\mathrm{A}'} \cup I_\mathrm{B} \rangle\right) \tag{578}$$

$$\left(\mu_i : \mathscr{B}_\mathrm{FLB}(b_i; \mu_i) = \mu'_i\right) \leftarrow z_i : i \in \mathrm{O_B} \tag{579}$$

$$\left(\mu_i : \mathscr{B}_\mathrm{FLB}(e_i; \mu_i) = \mu'_i\right) \leftarrow z_i : i \in \mathrm{O_{AA}} \tag{580}$$

Figure B.10: **Protocol S2PC-with-Coms (stage 6).** (See preceding in Figure B.9 and continuation in Figure B.11.) Legend in §Notation.

- Regardless of the OT type, $\mathrm{P_B}$ uses a PRG $\mathrm{PRGen_{InKey}}$ to pseudo-randomly generates the respective (tentative) circuit input wire key $\xi_{j,i}$, using as seed the previously received "auxiliary seed" and also the inner encoding ν_{j,i,b_i}.
  (563). This PG generation also uses as additional seed In this way, $\mathrm{P_A}$ learns one tentative key per circuit input wire of $\mathrm{P_A}$. (The keys are herein called *tentative* because they may be incorrect in case $\mathrm{P_A^*}$ acted maliciously.) Since $\mathrm{P_B}$ already knew also one tentative key for each input wire of $\mathrm{P_A}$ (544), $\mathrm{P_B}$ is able to parse one key for all input wires of the garbled circuit (564)
- **Evaluate GCs.** For each *evaluation* index ($j \in J_E$), $\mathrm{P_A}$ uses the keys obtained across all input wires to evaluate the GC and thus obtain one tentative key per circuit output wire (565).
- **Get output bit encodings.** $\mathrm{P_B}$ initializes an empty list (J_IGNORE), dubbed IGNORE list (566), where it will add any challenge index (j) for which it is not able to obtain a valid inner encoding (i.e., in respect to the expected inner BitCom).
  As mentioned, for simplicity it is assumed that each output key directly reveals the respective underlying bit, e.g., its least significant bit ($\epsilon(\cdot)$) (567). (The assumption, which is not essential, is simplifying by allowing $\mathrm{P_B}$ to directly know for which bit value it should generate an encoding and against which BitCom to verify its correctness.)

Then, for each output wire index ($i \in \mathrm{O_B}$) and each evaluation instance ($j \in J_E$), $\mathrm{P_B}$ uses a PRG procedure, seeded with the auxiliary seed ($\lambda_j^{(auxi)}$) and also with the obtained output wire key $\xi_{j,i}$ (tentative), to obtain a tentative inner encoding ("randomness") $v_{j,i}$ of the respective tentative output bit $c_{j,i}$ (568). $\mathrm{P_B}$ then verifies that the BitCom that can thus be generated (569) coincides with the respective inner BitCom $\nu'_{j,i,c}$ computed in the VERIFY stage (558).

If this is the case, $\mathrm{P_B}$ computes an outer encoding ($u_{j,i}$, opening of the respective output BitCom $\mu_{j,i}$) (570), by multiplying the obtained inner encoding ($v_{j,i}$) with the respective earlier received multiplier ($\beta_{j,i,c_{j,i}}$) (548). Otherwise, if the tentative inner encoding is not valid, then $\mathrm{P_B}$ adds the challenge index j to the IGNORE list, and ignores this index henceforth (571).

Remark B.3 (The Ignore list). The IGNORE list corresponds to indices ignored with respect to computations in the remainder of the procedure, but in practice the index may have to be accounted in counter-measures put in place to avoid side channel attacks. For example, a malicious $\mathrm{P_A}$ must not be able to find whether or not there are indices which were *ignored*, or otherwise such knowledge could enable a selective failure attack.

- **Check for inconsistencies.** For each output wire index, $\mathrm{P_B}$ gathers all valid pairs of bit and respective calculated encodings (572), i.e., ignoring indices that may have been selected for the IGNORE list.
  - **Regular path:** If for every output wire index there are no inconsistencies across different GCs (i.e., if for each wire index all pairs are the same) (573), then $\mathrm{P_B}$ simply accepts these bits ($b_i = c_{j,i}$) and respective bit encodings ($\mu_{j,i} = u_{j,i}^{b_i}$) as correct (574).
  - **Forge-and-lose path:** If $\mathrm{P_A}$ finds instead that for some output wire there is more than one outer encoding (i.e., valid opening) for the respective BitCom, then $\mathrm{P_B}$ activates the forge-and-lose path (575) to recover the final outputs. Within this path, $\mathrm{P_B}$ extracts the trapdoor t_{FLA} associated with the intermediate BitCom scheme ($\mathscr{B}_{\mathrm{FLA}}$) devised for the forge-and-lose path in respect to the input BitComs of $\mathrm{P_A}$. Essentially, this is obtained from a non-trivially-correlated pair of proper openings of one of the output BitComs of $\mathrm{P_B}$ (576), which has been produced with a dual BitCom scheme ($\mathscr{B}_{\mathrm{FLB}}$). $\mathrm{P_B}$ then uses the trapdoor to obtain the input bits of $\mathrm{P_A}$ from the respective intermediate forge-and-lose Com or BitComs (577), which had been committed in the

PRODUCE INITIAL BITCOMS OF P_A stage (491, 492, 493–494), then uses the original Boolean Circuit (C) to compute the circuit output bits (578) and finally chooses the openings (μ_i) that encode the correct output bits, including in output wires of P_B (579) and in the masked private output wires of P_A (580). (The encodings could also be extracted with the newly obtained trapdoor, but in practical instantiations that would require more exponentiations.)

In respect to the next steps, P_B interacts in a way that prevents P_A from learning via which path the output of P_B was obtained (regular (574) vs. forge-and-lose (575)). This may have to imply controlling the time of execution, eventually inserting an artificial delay, so that P_B does not detect a time difference between the two paths.

Stage 7 — TRANSMIT CIRCUIT OUTPUT OF P_A.

- **Private output bits of P_A.** P_B sends to P_A the masked bits obtained for private output wires ($i \in O_{AA}$) of P_A (581). Then, P_B proves that these are indeed the bits that it obtained in the garbled circuit evaluation. For simulatability under general instantiations, this can be a "NIZKPoK" of openings corresponding to the transmitted (masked) circuit output bits (582), which in turn can be trivially reduced to a statistically sound proof of committed zeros, upon a (pseudo) XOR-homomorphic BitCom adjustment performed locally by each party. In rigor, this ZK sub-protocol does not (and does not intend to) conform with the definition of *proof of knowledge* in the non-rewinding setting, in the sense that it does not need to allow extraction of the committed bits from the "NIZKPoK" transcript (indeed, the masked bits were simply sent in clear and P_A already knows all openings), but it is in a rewinding setting in order to ensure soundness. Thus, it may alternatively be informally understood as a NIZKP that the revealed bits are the correct ones, considering the underlying evaluation of garbled circuits.

Remark B.4 (Different proofs of knowledge). Depending on the type of instantiation, the proof of knowledge (582) could be allowed to actually reveal the learned openings, namely considering that P_A already knows the two openings per output wire — it only did not yet know in a verifiable way which openings P_B learned. For example, in a PKI setting where the forge-and-lose Equiv-BitCom scheme ($\mathscr{B}_{FLB}$) has the same trapdoor as the outer Ext-Com scheme ($\mathscr{B}_A$) defined by the PKI, and where P_A had provided a NIZKPoK of the trapdoor (enabling the simulator to extract it), the simulator (impersonating an

honest P_B playing against a possibly malicious P_A) may be able to extract a second opening of the forge-and-lose BitComs, once a first opening is known (e.g., learned in the Evaluate stage). In this case the proof of openings could correspond to P_B simply sending a CR-Hash of the learned openings ($\mu_i^{(b_i)}$), without affecting simulatability. Since P_A knows all openings, it could directly recompute the hash and verify that it is correct.

- **Common output bits.** For simulatability under corruption of any party, it would be enough for P_B to send in clear the bits obtained in the wires corresponding to common output wires. However, interestingly, the case of no corrupted party requires that the bits are not visible to a passive eavesdropper (the real adversary), and so the bits need to be encrypted between P_B and P_A. In practice P_B can encrypt each such bit using the Ext-BitCom scheme $\mathscr{B}_{\text{FLA}}$ used by P_A for the forge-and-lose technique. Specifically, P_B produces and sends one BitCom per common output bit (583–584), and then proves that the known openings correspond to the same bits as the known openings of the forge-and-lose output BitComs ($\mathscr{B}_{\text{FLB}}$) obtained from the garbled circuit evaluations (585). P_A is able to use her trapdoor to open the BitComs and discover the bit values (586).

Stage CF.3 — Coin-flip Permutations (Finish)**.** The last stage of the coin-flipping of permutations takes place.

- **Wire sets of P_B.** P_B asks the ideal generalized coin-flipping functionality to send the output to P_A (587). As a result, the ideal functionality $\mathcal{F}_{\text{GMCF-1}}$ sends to P_A one outer Com permutation γ_{I_B} for the set of input wires of P_B, and another permutation γ_{I_B} for the set of output wires $\gamma_{O_{B'}}$ (588)
- **Wire sets of P_A.** P_B opens his previously committed contribution, which in the hybrid-$\mathcal{F}_{\text{MCom}}$ model corresponds to P_B asking the functionality to open the commitment (589) and then the functionality sending the committed value to P_A (590). From the opened outer-randomness permutations γ decided by P_B for the wire sets of P_A, both parties locally compute the respective outer-com permutations γ' (591).

Stage 8 — Permute Outer Coms**.**

- **8.1. Adjust outer private output wires of P_B.**
  For each private output bit index $i \in O_{BB}$ of P_B, P_B computes the bit offset e_i between the

Stage 7. TRANSMIT CIRCUIT OUTPUT OF $\mathrm{P_A}$.

7.1. Private output wires of $\mathrm{P_A}$.

$\mathrm{P_B} \to \mathrm{P_A} : e_i : i \in \mathrm{O_{AA}}$ (see (578)) (581)

$\mathrm{P_B} \to \mathrm{P_A} : \mathrm{NIZKPoK}_{\mathrm{OpenKnownBits}}[\mathscr{B}_{\mathrm{FLB}}]$
$\left(\left\langle (\mu'_i, e_i) : i \in \mathrm{O_A} \right\rangle\right)$ (582)

7.2. Common output wires of $\mathrm{P_A}$.

$\mathrm{P_B} : \phi_i \leftarrow^{\$} \mathrm{Gen}_{\$\mathrm{ForCom}}[\mathscr{B}_{\mathrm{FLA}}](b_i) : i \in \mathrm{O_A} \backslash \mathrm{O_{AA}}$ (583)

$\mathrm{P_B} \to \mathrm{P_A} : \phi'_i = \mathscr{B}_{\mathrm{FLA}}(b_i; \phi_i) : i \in \mathrm{O_A} \backslash \mathrm{O_{AA}}$ (584)

$\mathrm{P_B} \to \mathrm{P_A} : \mathrm{NIZKP}_{\mathrm{SameComBits}}[(\mathscr{B}_{\mathrm{FLA}}, \mathscr{B}_{\mathrm{FLB}})]$
$\left(\left\langle (\phi'_i, \mu'_i) : i \in \mathrm{O_A} \backslash \mathrm{O_{AA}} \right\rangle\right)$ (585)

$\mathrm{P_A} : b_i = \mathrm{Extract}_{\mathrm{Bit}}[t_{\mathrm{FLA}}](\phi'_i) : i \in \mathrm{O_A} \backslash \mathrm{O_{AA}}$ (586)

CF.3. COIN-FLIP PERMUTATIONS (FINISH).

Wire sets of $\mathrm{P_B}$.

$\mathrm{P_B} \to \mathcal{F}_{\mathrm{GMCF\text{-}1}} : (\mathtt{OK}, ctx_{\mathtt{cf}})$ (see (470,535)) (587)

$\mathcal{F}_{\mathrm{GMCF\text{-}1}} \to \mathrm{P_A} : \left(\mathtt{out\text{-}2}, ctx_{\mathtt{cf}}, \left(\gamma'_{I_\mathrm{B}}, \gamma'_{\mathrm{O_B}}\right)\right)$ (588)

Wire sets of $\mathrm{P_A}$.

$\mathrm{P_B} \to \mathcal{F}_{\mathrm{MCom}} : (\mathtt{open}, ctx_{\mathtt{com1}})$ (see (473)) (589)

$\mathcal{F}_{\mathrm{MCom}} \to \mathrm{P_A} : \left(\mathtt{open}, ctx_{\mathtt{com1}}, \left(\gamma_{I_\mathrm{A}}, \gamma_{\mathrm{O_{A'}}}\right)\right)$ (590)

$\mathrm{P_A}, \mathrm{P_B} : \gamma'_{set} = \mathscr{C}_{\mathrm{B}}\left(0^{\#(set)}; \gamma^{(\mathrm{B})}_{set}\right) : set \in \{I_\mathrm{A}, \mathrm{O_{A'}}\}$ (591)

Stage 8. PERMUTE OUTER COMS.

8.1. Adjust outer private output wires of $\mathrm{P_B}$.

$\mathrm{P_B} \to \mathrm{P_A} : e_i = b_i \oplus d_i : i \in \mathrm{O_{BB}}$ (see (457)) (592)

$\mathrm{P_B} : \sigma_i = (z_\mathrm{B}{}^{e_i}) \ddagger_{(e_i)} \varsigma_i : i \in \mathrm{O_{BB}}$ (see (459)) (593)

$\mathrm{P_B} : \sigma_{\mathrm{O_{BB}}} = \mathrm{Stringize}\left(\langle \sigma_i : i \in \mathrm{O_{BB}} \rangle\right)$ (594)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_i = z'_\mathrm{B}{}^{e_i} \ast'_{(e_i)} \varsigma'_i : i \in \mathrm{O_{BB}}$ (see (459)) (595)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_{\mathrm{O_{BB}}} = \mathrm{Stringize}\left(\langle \sigma'_i : i \in \mathrm{O_{BB}} \rangle\right)$ (596)

$\mathrm{P_B} \to \mathrm{P_A} : \mathrm{NIZKP}_{\mathrm{SameComBits}}[(\mathscr{B}_{\mathrm{B}}, \mathscr{B}_{\mathrm{FLB}})]$
$\left(\left\langle (\sigma'_i, \mu'_i) : i \in \mathrm{O_{BB}} \right\rangle\right)$ (597)

8.2. Adjust outer private output wires of $\mathrm{P_A}$.

$\mathrm{P_A} : b_i = e_i \oplus d_i : i \in \mathrm{O_{AA}}$ (see (581) and (478)) (598)

$\mathrm{P_A} : \sigma_i = (z_\mathrm{A}{}^{e_i}) \ddagger_{(e_i)} \varsigma_i : i \in \mathrm{O_{AA}}$ (see (479)) (599)

$\mathrm{P_A} : \sigma_{\mathrm{O_{AA}}} = \mathrm{Stringize}\left(\langle \sigma_i : i \in \mathrm{O_{AA}} \rangle\right)$ (600)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_i = z'_\mathrm{B}{}^{e_i} \ast'_{(e_i)} \varsigma'_i : i \in \mathrm{O_{AA}}$ (see (480)) (601)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_{\mathrm{O_{AA}}} = \mathrm{Stringize}\left(\langle \sigma'_i : i \in \mathrm{O_{AA}} \rangle\right)$ (602)

8.3. Integrate common output wires.

For $p \in \{\mathrm{A}, \mathrm{B}\}$:

$\mathrm{P_A}, \mathrm{P_B} : \sigma_{\mathrm{O}_{pp'}} = \mathrm{Stringize}\left(\langle z_p{}^{b_i} : i \in \mathrm{O}_{pp'} \rangle\right)$ (603)

$\mathrm{P}_p : \sigma_{\mathrm{O}_{p'}} = \sigma_{\mathrm{O}_{pp}} \| \sigma_{\mathrm{O}_{pp'}} : p \in \{\mathrm{A}, \mathrm{B}\}$ (604)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_{\mathrm{O}_{pp'}} = \mathrm{Stringize}\left(\left\langle z'_p{}^{b_i} : i \in \mathrm{O}_{pp'} \right\rangle\right)$ (605)

$\mathrm{P_A}, \mathrm{P_B} : \sigma'_{\mathrm{O}_{p'}} = \sigma'_{\mathrm{O}_{pp}} \| \sigma'_{\mathrm{O}_{pp'}}$ (606)

8.4. Apply random permutations.

$\mathrm{P_B} : \rho_{set} = \sigma_{set} \ddagger \gamma_{set} : set \in \{I_\mathrm{B}, \mathrm{O_{B'}}\}$ (see (536)) (607)

$\mathrm{P_A} : \rho_{set} = \sigma_{set} \ddagger \gamma_{set} : set \in \{I_\mathrm{A}, \mathrm{O_{A'}}\}$ (see (590)) (608)

$\mathrm{P_A}, \mathrm{P_B} : \rho'_{set} = \sigma'_{set} \ast' \gamma'_{set} : set \in \{I_\mathrm{A}, I_\mathrm{B}, \mathrm{O_{A'}}, \mathrm{O_{B'}}\}$ (609)

Stage 9. FINAL OUTPUT.

$\mathrm{P_B} : y_\mathrm{B} \equiv \langle b_i : i \in \mathrm{O_B} \rangle$ (610)

$\mathrm{P_B} : res_\mathrm{B} \equiv \left(\rho'_{I_\mathrm{A}}, (\rho_{I_\mathrm{B}}, \rho'_{I_\mathrm{B}}), \rho'_{\mathrm{O_{A'}}}, (y_\mathrm{B}, \rho_{\mathrm{O_{B'}}}, \rho'_{\mathrm{O_{B'}}})\right)$ (611)

$\mathrm{P_B} \to \mathcal{Z} : (\mathtt{s2pcwC\text{-}out\text{-}1}, ctx, res_\mathrm{B})$ (612)

$\mathrm{P_A} : y_\mathrm{A} \equiv \langle d_i \oplus e_i : i \in \mathrm{O_{AA}} \rangle \| \langle b_i : i \in \mathrm{O_{AA'}} \rangle$ (613)

$\mathrm{P_A} : res_\mathrm{A} \equiv \left((\rho_{I_\mathrm{A}}, \rho'_{I_\mathrm{A}}), \rho'_{I_\mathrm{B}}, (y_\mathrm{A}, \rho_{\mathrm{O_{A'}}}, \rho'_{\mathrm{O_{A'}}}), \rho'_{\mathrm{O_{B'}}}\right)$ (614)

$\mathrm{P_A} \to \mathcal{Z} : (\mathtt{s2pcwC\text{-}out\text{-}2}, ctx, res_\mathrm{A})$ (615)

Figure B.11: **Protocol S2PC-with-Coms (stages 7, CF.3, 8 and 9).** (See preceding in Figure B.10.) Legend in §Notation.

previously computed random bits d_i and the actual obtained circuit bit b_i, and sends it to $\mathrm{P_A}$ (592). Then, $\mathrm{P_B}$ uses the mutually agreed auxiliary fixed encoding z_B (of the BitCom scheme ($\mathscr{B}_\mathrm{B}$) of $\mathrm{P_B}$) (353), to adjust the initial outer-randomness (ς_i) of her output bits into randomness (σ_i) that effectively correspond to her output circuit bits (b_i) (593). This is done as follows: if the offset bit is 0, then no change is required; if the offset bit is 1, then the adjustment corresponds to dividing the fixed encoding (z_B) by the inverse of the initial outer encoding (ς_i) (as in a pseudo-homomorphism). $\mathrm{P_B}$ stringizes the result, to obtain the initial bit-string outer-com $\sigma_{\mathrm{O_{BB}}}$ of her private output wires (594).

Both parties (i.e., including $\mathrm{P_A}$) locally make a respective adjustment in the BitCom space, leading the initial outer BitComs (ς'_i) of random bits (d_i) of $\mathrm{P_B}$ into BitComs of the

(supposed) output obtain by P_B, also based on a XOR pseudo homomorphism (595): if the offset bit (e_i) is 0, then no change is required; if the offset bit is 1, then the adjustment corresponds to dividing the fixed BitCom of 1 (z'_B) by the inverse of the initial outer BitCom (ς'_i). Both parties also stringize the result, to obtain a single bit-string outer-Com of the private output bits of P_B (596).

If P_B acted honestly, then the BitComs (σ'_i) resulting from the adjustment are supposedly committing to the same bits (b_i) that P_B obtained from the S2PC evaluation. To ensure this, P_B sends to P_A a respective proof, in the form of a ZKP of same committed bits ($\text{NIZKP}_{\text{SameComBits}}$) between the BitComs μ'_i associated with the forge-and-lose technique ($\mathscr{B}_{\text{FLB}}$) (of which P_B has learned one opening) and the newly permuted outer BitComs (σ'_i) of P_B (597) (this can also be done directly from the respective stringized BitComs).

- **8.2. Adjust outer private output wires of P_A.** Based on the offset bits e_i received from P_B and proven correct, and the private bit-masks previously decided, P_A computes her private output bits b_i (598). Then, also based on the offset bits e_i, P_A adjusts the randomness ς_i associated with her private output wires (which were committing to random bit-masks), into initial outer-randomnesses σ_i for committing to her actual private output bits (599). Once again, this is achieved with the help of the auxiliary group element z_A associated with her BitCom scheme. The vector of BitComs is then stringized into a single outer-randomness element $\sigma_{O_{AA}}$ (600). Correspondingly, both parties make the same adjustments at the level of BitComs, i.e., to produce an initial outer-BitCom of the private output bit of P_A (601), and stringize it into an initial outer-Com for the set of private output wire of P_A (602).
- **8.3. Integrate common output wires.** Both parties stringize also an initial randomness $\sigma_{O_{pp'}}$ associated with the common output wires of each party — it is worth recalling here that for common output wires there are two independent BitComs produced, each party leaning an opening of each. Since each party known the value of the common output bits, the initial randomnesses are defined locally, based on the auxiliary group element z_p of each BitCom scheme (603). Then, each party is able to concatenate the initial outer-randomness of her private output wires set, with the randomness just computed of the common output wires set and associated with the respective BitCom scheme (604). The parties can then repeat the procedure at the level of BitComs: both parties locally produce and stringize initial outer-BitComs for the common output wires, for each BitCom scheme (605); both parties concatenate, for each type of wire (input and output) of each

party P_p, the initial outer-Com $\sigma'_{O_{pp}}$ of private output wires of the party, with the initial outer-Com $\sigma'_{O_{p'}}$ of the common output wires of the party, thus obtaining a single initial outer-Com $\sigma'_{O_{p'}}$ of (all) the output wires of each party (606).

- **8.4. Apply random BitCom permutations.** Finally, each party locally applies the previously decided random outer-randomness permutations (γ_{set}) directly to the initial known (i.e., possibly modified) outer randomnesses (σ_{set}) (the openings related with their own circuit input and circuit output bits). Specifically: P_A applies it to the initial (outer) randomnesses of her own input and output bits (608); P_B applies it to the initial (outer) randomnesses of his input and output bits (607), and to the modified initial outer encodings of his output bits. Al the level of outer-Coms (instead of outer-randomnesses), each party also applies the respective permutations (γ'_{set}) associated to the respective initial (including modified) input and output outer-Coms (σ'_{set}) of both parties (609). The resulting permuted values (σ_{set} and σ'_{set}) are dubbed final outer-randomnesses and final outer-Coms, respectively.

Stage 9 — Final Output. Each party prepares her final outputs as follows. P_B parses his output bits into a respective bit-string y_B (610), and then computes the expected tuple of needed final outer-randomnesses and outer-Coms, namely including the outer-Coms ρ'_{set} of the sets of input and output wires of both parties and the outer-randomnesses ρ_{set} of his own input and output wires, and also including his final output wires (611). This tuple is what an honest P_B outputs (in a contextualized manner) to the environment (612).

P_A also parses her output bits into a bit-string y_A (613), and then prepares a corresponding tuple for her final output, namely including the final outer-Coms ρ'_{set} for the sets of input and output bits of both parties, and the final outer-randomnesses for the sets of her input and output bits (614), and also including her final circuit output bit-string y_A. Finally, P_A outputs the tuple to the environment (615).

B.3 Simulators for the S2PC-with-Coms protocol

This section describes simulators to prove security of the S2PC-with-Coms protocol. §B.3.1 shows the simulator for the case of a malicious P^*_A (the constructor of garbled circuits). §B.3.2 gives a modular proof of soundness in case of malicious P^*_A, to complete the argument of

ideal/real indistinguishability. §B.3.3 shows the simulator for the case of a malicious P_B^* (the garbled circuit evaluator) and a respective analysis.

In the following subsections, the description considers the simplification (possible in the static model with up to one corrupted party) whereby, in each world, the adversary and the corrupted party are mentioned as the same entity. Whenever the simulator ($\mathcal{S} \equiv \widehat{P}_p^*$) receives an input from the environment ($\mathcal{Z}$) in the ideal world, it relays it to the real adversary ($\mathcal{A} \equiv P_p^*$) in the simulated execution. Whenever the adversary in the simulated execution outputs something to its environment, $\mathcal{S}$ relays it to $\mathcal{Z}$.

B.3.1 Simulator for the case of malicious P_A^*

The simulator $\mathcal{S} \equiv \mathcal{S}^{A^*}$, with access to a black-box malicious P_A^*, simulates the beginning of a real protocol execution. $\mathcal{S}^{A^*}[P_B]$ impersonates a real honest P_B while interacting in the simulated execution with the malicious P_A^*. As the simulation proceeds, if P_A^* induces the simulated execution to abort (i.e., takes actions that would make an honest P_B abort), then $\mathcal{S}^{A^*}[\widehat{P}_A]$ (in the role of ideal $\widehat{P}_A^*$ in the ideal world) sends `abort` to the TTP, thus leading the ideal $\widehat{P}_B$ to also receive (and then output) `abort` (384–415), and then $\mathcal{S}^{A^*}[\widehat{P}_A]$ outputs in the ideal world whatever P_A^* outputs in the simulated execution. It is worth noticing that after P_B is able to find his output in a real execution it will no longer abort in the real world. Similarly, an `abort` message sent from $\mathcal{S}^{A^*}[\widehat{P}_A]$ to the TTP after $\widehat{P}_B$ has already received the output from the TTP will no longer make the ideal $\widehat{P}_B$ abort, because it has already outputted to $\mathcal{Z}$.

1. **Impersonate honest P_B with arbitrary input.**
   - **Commit arbitrary input bits of P_B.** In the Outer BitComs of input bits of P_B stage, $\mathcal{S}^{A^*}[P_B]$ simulates an honest P_B with a circuit input composed of all zeros, i.e., a zero for each input bit of P_B (454–456). Given the hiding property of BitComs, the view of P_A^* is indistinguishable from the case where P_B would commit any other input.
   - **Other BitComs as honest P_B.** Then, $\mathcal{S}^{A^*}[P_B]$ continues as an honest P_B, producing outer Coms of random output offset bits of P_B and proving them correct (457–460), eventually producing (i.e., if-need-be) new intermediate BitComs for the OTs and a respective ZKP of same commited bits (461–464).
   - **ZKPoK of openings.** After *stringizing* the Outer BitComs of P_B, i.e., parsing the outer BitComs into two BitSringComs (one for the set of input wires and the other

for the set of output wires) (467), $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ gives a ZKPoK of the openings of the outer Coms (468). In a PKI setting, where the ZKPoK of openings may be facilitated by a ZKPoK of the trapdoor, $\mathcal{S}$ uses his power to give a fake ZKPoK of trapdoor (e.g., by equivocating the opening of a RSC Equiv-Com used in a NPRO-transformation to define the proof challenge). If the ZKPoK of openings does not involve the trapdoor, e.g., in a CRS setting, then $\mathcal{S}$ gives an honest ZKPoK of the openings (that it indeed knows) of the outer Coms σ'_{set}.

- **Initiate coin-flip of permutations.** $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ initiates the coin-flipping stage as an honest $\mathrm{P_B}$ would, committing to outer-randomness permutations γ for the wire sets of $\mathrm{P_A}$ (472–474), and initiating the generalized coin-flipping of outer-Coms of 0 (permutations) for the wire sets of $\mathrm{P_B}$ (470–471). In the considered ($\mathcal{F}_{\text{GMCF-1}}$,$\mathcal{F}_{\text{MCom}}$)-hybrid model, $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ will later be able to equivocate any of these permutations.

2. **Extract circuit input and randomness of $\mathbf{P}^*_{\mathbf{A}}$.** The simulated execution then enters the stage of $\mathrm{P}^*_{\mathrm{A}}$ committing to her bits (private input bits and random bit-masks for her future output) (475–489). From the respective NIZKPoK of openings (490), $\mathcal{S}$ extracts the private circuit input bits and the random output bit-masks of $\mathrm{P_A}$, as well as the randomnesses $(\sigma_{I_{\mathrm{A}}}, \varsigma_{\mathrm{O_{AA}}})$ used to commit them. For IFC a single NIZKPoK of trapdoor is enough, as it enables computing square-roots. For DLC the extraction is enabled via NIZKPoKs of ElGamal opening.
3. **Continue simulation till receiving all elements from $\mathbf{P}^*_{\mathbf{A}}$.** $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ continues the simulation as an honest $\mathrm{P_B}$. It is assumed that the cut-and-choose partition is selected in a way that prevents $\mathrm{P}^*_{\mathrm{A}}$ from guessing the exact partition before producing the previously committed elements (see §3.1.4 and §B.3.2). After the decision and validation of the cut-and-choose partition (530–534), the simulation enters the next stage of the coin-flipping. (Coin-flip Permutations (Continue)), where $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ receives outer-randomness permutations γ for the set of input wires and the set of output wires of $\mathrm{P_B}$ (536). $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ also receives the respective responses from $\mathrm{P}^*_{\mathrm{A}}$, including RSC seeds for check instances (538), other elements (e.g., multipliers) for evaluation instances (539–548), and the opening of the global hash from the RSC Equiv-Com $\overline{\Lambda}$ (549). As in a regular protocol execution, $\mathcal{S}$ verifies the correctness of all the responses (550–560), which includes verifying the multipliers and the global hash.
4. **Interact with the TTP.** If some verification fails during or before the Verify stage, then $\mathcal{S}^{\mathrm{A}^*}[\widehat{\mathrm{P}}_{\mathrm{A}}]$ emulates an abort in the ideal world, and outputs in the ideal world whatever

P_A^* outputs in the simulated execution. The abort in the ideal world leads the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ to send an `abort` message to the ideal P_B (415), which in turn leads P_B to output an `abort` to $\mathcal{Z}$.

Otherwise, if all verifications are successful, an honest P_B would have all the elements needed to compute his own output of the protocol. However, $\mathcal{S}$ does not perform the evaluation stage in the simulated execution. Instead, it pauses the simulation and assumes its role in the ideal world, as the ideal $\widehat{P}_A$, and sends to the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ the initial message of the protocol in the ideal world, containing the input extracted from P_A^* (392).

Once $\mathcal{F}_{\text{S2PCwC}}$ receives from the ideal $\widehat{P}_B$ the respective input (392), and once the public parameters of the Com schemes of both parties are known, the functionality makes the needed local computations (396- 402), sends the output initially to the ideal honest $\widehat{P}_B$ (404) and then, after receiving an OK message from $\widehat{P}_B$ (383), $\mathcal{F}_{\text{S2PCwC}}$ sends the final output to the simulator $\mathcal{S}^{A^*}[\widehat{P}_A]$ (409), which includes the circuit output y_A of $\widehat{P}_A$, the commitments $(\overline{x}_A, \overline{x}_B, \overline{y}_A, \overline{y}_B) \equiv (\rho'_{I_A}, \rho'_{I_B}, \rho'_{O_{A'}}, \rho'_{O_{B'}})$ (402) of inputs and output of both parties, and the randomnesses $(\underline{x}_A, \underline{y}_A) \equiv (\rho_{I_A}, \rho_{O_{A'}})$ used to commit the input and output of $\widehat{P}_A$.

5. **Enforce the circuit output of P_A computed by the TTP.** $\mathcal{S}^{A^*}[P_B]$ resumes the simulation in order to induce P_A^* to obtain the output decided by $\mathcal{F}_{\text{S2PCwC}}$, as follows. $\mathcal{S}^{A^*}[P_B]$ executes the EVALUATE stage (561–580), evaluating the garbled circuits until obtaining all circuit outputs and one valid opening per forge-and-lose outer BitCom. Assuming the cut-and-choose partition has adequate parameters (§3.1.4), there is a negligible probability that a malicious P_A^* could have prevented $\mathcal{S}^{A^*}[P_B]$ from a successful evaluation (i.e., conditioned to the previous validation in the VERIFY stage). Thus, hereafter the analysis assumes a successful evaluation.

   $\mathcal{S}^{A^*}[P_B]$ has previously extracted the bit-masks d_i that P_A^* defined for the private output wires of P_A (see (478,482,490)), so it is able to compute the respective offset bits e_i that are required to induce the private circuit output bits b_i of P_A as decided by $\mathcal{F}_{\text{S2PCwC}}$ (613). $\mathcal{S}^{A^*}[P_B]$ sends these offset bits to P_A^* (581). For the common output wires it commits to the bits informed by the ideal functionality (583–584). Then, $\mathcal{S}^{A^*}[P_B]$ sends to P_A an appropriate proof that these are the correct bits (582).

6. **Enforce the final outer-Coms and outer-randomnesses computed by the TTP**

   - **Summary of relevant values already held by $\mathcal{S}$.** In regard to the outer-Coms

and outer-randomnesses, $\mathcal{S}^{A^*}$ has obtained the following relevant sets of elements: (i) from the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$, it received the final outer-Coms ρ' for wires of P_B and final outer-randomnesses ρ for wires of P_A (409), which need to be induced to the final output that P_A^* would at this point accept if it is honest; (ii) from the malicious P_A^*, it received the initial outer Coms σ of input bits of P_A (476), the outer coms ς of random bit-masks for the output bits of P_A (479), and has extracted the respective openings from the respective NIZKPoK (490); (iii) from the ideal generalized coin-flipping $\mathcal{F}_{\text{GMCF-1}}$, it received the outer-randomness permutations γ for the wires sets of P_B (536); (iv) from the initial commitments of the impersonated honest P_B, it already sent to P_A the initial outer-Coms σ' of input bits of P_B (455), as well as of output offset masks of output bits of P_B (458), and about which it knows the respective randomnesses σ;

- **Compute initial outer-Coms and initial outer-randomnesses for output wires.** $\mathcal{S}^{A^*}[P_B]$ looks ahead in the protocol structure (to the PERMUTE OUTER COMS stage), and locally makes the needed adjustments to obtain the "initial" outer-Com ($\sigma'_{O_{BB}}$) of the set of private output wires of P_B (595- 596), the "initial" outer-randomnesses ($\sigma_{O_{AA}}$) of private output wires of P_A (599–600), and the initial outer-randomness ($\sigma_{O_{pp'}}$) and outer-Coms ($\sigma'_{O_{pp'}}$) of common output wires of both parties (i.e., one version for each party $p \in \{A, B\}$) (603,605).
- **Determine the needed permutations.** Based on the mentioned values, and considering the group structure upon which the outer-Coms are supported, $\mathcal{S}^{A^*}[P_B]$ computes what are the necessary **outer-Com** permutations ($\gamma'_{I_B}, \gamma'_{O_{B'}}$) of wire sets of $\mathbf{P_B}$ that lead the respective initial outer-Coms ($\sigma'_{I_B}, \sigma'_{O_{B'}}$) into the respective final outer-Coms ($\sigma'_{I_B}, \sigma'_{O_{B'}}$), and what are the needed **outer-randomness** permutations ($\gamma_{I_A}, \gamma_{O_{A'}}$) of wire sets of $\mathbf{P_A}$ that lead the respective initial outer randomnesses ($\sigma_{I_A}, \sigma_{O_{A'}}$) into the respective final outer randomnesses ($\rho_{I_A}, \rho_{O_A} line$) of P_A.
- **Equivocate the coin-flipping result.** $\mathcal{S}^{A^*}[P_B]$ uses its equivocation power to equivocate the opening of the generalized coin-flipping and the Ext-and-Equiv Com scheme to the needed values. In the ($\mathcal{F}_{\text{GMCF-1}}$,$\mathcal{F}_{\text{MCom}}$)-hybrid model this equivocation is achieved by impersonating the respective ideal functionalities and simply opening the intended values to P_A^* (588,590).
- **Enforce needed adjustments to initial outer-Coms of output wires of $\mathbf{P_B}$.** $\mathcal{S}^{A^*}[P_B]$ then sends the remaining values expected by P_A^*, namely the offset bits

associated with output wires of P_B (592) and a respective ZKP of correctness with respect to the evaluated bits and the initially committed offsets (597).

- **Final output.** P_A^* has now all the permutations (588–591) needed to permute the initial outer-randomnesses σ (600, 603, 604) and initial outer-Coms σ' (602, 605, 606) into the respective final outer-randomnesses ρ (606) and final outer-coms ρ' (608), 609, thus obtaining the output proposed by the ideal functionality $\mathcal{F}_{\text{S2PCwC}}$ (613–614). $\mathcal{S}$ outputs in the ideal world whatever P_A^* outputs in the real world 615, be it the correct output or anything else, including `abort`.

B.3.1.1 Analysis of the simulation

The indistinguishability of S2PC-with-Coms between the two worlds (ideal and real) follows directly from the simulatability of the underlying coin-flipping protocol, the hiding of BitComs of P_B and the ability to produce fake NIZKPs and NIZKPoKs.

A high level protocol structure. The execution is analyzed in three macro phases:

1. In the first phase, P_B sends commitments to P_A, along with NIZKPs and NIZKPoKs.
2. In the second phase, P_A commits her input, and proceeds with the cut-and-choose approach, until the point where it sends respective responses to the cut-and-choose partition challenge, as well as her contribution to the coin-flipping of outer-randomness permutations for wire sets of P_B. Except for an eventual influence in the soundness error probability, it is irrelevant whether the decision of the cut-and-choose partition involves interaction (i.e., if decided by P_B between the Commit and the Respond stages) or non-interactively (e.g., by P_A based on a NPRO applied to the RSC Equiv-Com, of by P_B and hidden by initial OTs P_B).
3. In the third phase, P_B determines his output of the execution and sends to P_A the elements that allow P_A to compute her own output of the protocol execution

Analysis in a hybrid world. The simulation is initially analyzed assuming that all sub-protocols and primitives are replaced by respective ideal functionalities. Each NIZKP and NIZKPoK sent by $\mathcal{S}^{A^*}[P_B]$ is replaced by a message from a trustworthy ideal functionality simply asserting correctness of the assertion/knowledge being proven. The Ext-and-Equiv commitment is replaced by a respective ideal commitment functionality $\mathcal{F}_{\text{MCom}}$.

In the first phase, $\mathcal{S}$ acts as an honest $\mathrm{P_B}$ with an arbitrary circuit input. The only part where it may deviate is if the chosen NZKPoK involves (for efficiency reasons) a ZKPoK of trapdoor, which $\mathrm{P_B}$ would possess but the $\mathcal{S}$ does not have. Yet, in a hybrid instantiation with an ideal ZKPoK functionality the view by $\mathrm{P_A^*}$ is equal to the case where an honest $\mathrm{P_B}$ would use the same input. Also, the use of an arbitrary input (e.g., all zeros) is in the view of $\mathrm{P_A^*}$ indistinguishable from any other input, because of the hiding property of the BitComs. Also, since the final output decided by the ideal functionality $\mathcal{F}_{\mathrm{S2PCwC}}$ will instead be based on the actual input of the ideal $\widehat{\mathrm{P}}_\mathrm{B}$, the arbitrary input simulated by $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$ is irrelevant.

In the second macro phase, a malicious $\mathrm{P_A^*}$ may adaptively decide her circuit input depending on the actual values it received, e.g., the initial outer-Coms produced by $\mathcal{S}^{\mathrm{A}^*}[\mathrm{P_B}]$. Regardless, the circuit input x_A that the malicious $\mathrm{P_A^*}$ decides to use (i.e., unless it decides to abort or lead the honest $\mathrm{P_B}$ to abort) is the one that the simulator will extract and use in the ideal world. In other words, even in the ideal world a malicious party is already able to be malicious in respect to her input, so this action is indistinguishable between the two worlds. Since the initial outer-Coms are not part of an honest output, namely will not be outputted by a real $\mathrm{P_B}$, they also cannot be convincingly used by a malicious $\mathrm{P_A^*}$ to induce indistinguishability, i.e., a real malicious $\mathrm{P_A^*}$ cannot convince a party external to the protocol that the initial outer-Coms of $\mathrm{P_B}$ were actually produced by $\mathrm{P_B}$ (it may as well have been $\mathcal{S}^{\mathrm{A}^*}[\widehat{\mathrm{P}}_\mathrm{A}]$ to simulate it).

Besides breaking the hiding assumption of the outer BitComs, which would imply breaking a cryptographic assumption, the only other avenue for attempting a distinguishable execution is inducing $\mathrm{P_B}$ in the real world to fail to obtain a correct output. This would happen if $\mathrm{P_B}$ is not able to decide a final circuit output, or if it is lead to decide an incorrect output, e.g., by using inputs different from those that have been committed by outer-Coms. However, this cannot happen, except with at most a negligible probability dependent on the cut-and-choose configuration. This aspect of *soundness against a malicious* P_A^* is proven separately in §B.3.2.

The intuition is as follows. Given the forge-and-lose technique, $\mathrm{P_A^*}$ would accept an incorrect output only if the output of all evaluation circuits is identically incorrect. However, since incorrect circuits or incorrect input bits would be caught in any *check* challenge, $\mathrm{P_A^*}$ would have to guess in advance the C&C partition. If $\mathrm{P_A}$ would correctly build all circuits, but would use different inputs across several evaluation circuits, then either the output would be different across several garbled circuits and thus enable $\mathrm{P_B}$ to use the forge-and-lose path,

or the output would nonetheless be the same everywhere (e.g., possibly by virtue of a specific circuit and specific inputs) and in that case the output would be correct.

The respective negligible probability for guessing the C&C partition, and respective needed numbers of garbled circuits are summarized in Table 3.1 and Table 3.2.

§B.3.2 proves that, if P_A^* is not able to guess in advance the C&C partition, then with overwhelming probability one of the following three occurs: some inconsistency is detected in the VERIFY stage; or inconsistencies in the EVALUATE stage will lead P_B to obtain a correct final output via the forge-and-lose path, or no inconsistency is detected and the computation leads to a correct final output.

The third macro phase, which in terms of interaction only involves a communication from P_B to P_A, is, in the view of P_A^*, also indistinguishable between the real and the simulated execution. Specifically, the actions that in the ideal world are different from those of an honest P_B in the real world are the equivocation of the Ext-and-Equiv com used in the coin-flipping, the faking of the NIZKPoK revealed in the coin-flipping, and the faking of NIZKPs. All of these are indistinguishable when using the respective ideal functionalities, as well as when using concrete simulatable instantiations.

B.3.2 Soundness against P_A^*

Soundness requires that a malicious P_A^* cannot, except with negligible probability, make an honest P_B accept an incorrect output. This means that P_B must only accept a circuit output that would be computed by a correct Boolean circuit, that the circuit inputs used therein are the inputs committed in the outer Coms of both parties, and that the outer-Coms of P_A are indeed openable by P_A. The ZKPoK of openings of outer Coms of bits of P_A (490) already ensures that P_B only accepts those Coms if P_A knows respective openings. Thus, it remains to show that the committed bits are indeed the ones used in the garbled circuit evaluation. Soundness also requires that the probability of P_B obtaining his (correct) circuit output vs. aborting without circuit output be independent (except variations up to a negligible amount) of his private circuit input, and consequently also of any intermediate or output bit of the computation. This follows from the hiding property of the commitments of P_B, as well as from the statistical nature of the cut-and-choose that with overwhelming probability renders ineffective any selective failure attack by P_A.

B.3.2.1 Proof sketch

In the S2PC-with-Coms protocol, all inconsistencies found in the VERIFY stage are *complainable*, even if referring to *evaluation* indices. Conversely, all inconsistencies found in the EVALUATE stage are, for privacy-reasons (e.g., to resist a selective failure attack), *non-complainable*. This *complainability* notion is expressed from the point of view of whether of not P_B is allowed to *complain* without any security property being broken. Since the protocol does not specify a concrete *complaining* action, besides aborting, a non-complainable inconsistency is one whose detection should not lead P_B to abort, lest it breaks some security property. Thus, soundness is broken not only if P_B is lead to output an incorrect circuit output, but also when it is lead to abort due to a non-complainable inconsistency, i.e., when in fact it should have not aborted. More concretely, if the VERIFY stage is successful, then a subsequent abort by P_B could endanger the privacy of P_B. For example, since P_B is the first to learn the output, an inability for an honest P_B to send a proper final message to P_A could be understood as P_A learning that P_B found an inconsistency and failed to complete the circuit evaluation.

Types of revealed elements. The potential for breaking soundness is associated with the correctness and consistency of garbled circuits and connectors, which are committed and partially revealed for check or evaluation, differently for each type of challenge. Before elaborating the proof of soundness, it is useful to recall the three types revealed in the RESPOND stage of the protocol, in association with the RSC technique and the cut-and-choose (see §3.1.2 and Table 3.5):

- **Type-C — *Base-commitment* elements.** The RSC commitment $\overline{\Lambda}$ commits directly to the garbled circuits (which commit the Boolean circuit), to the inner Coms ν' (which are a one-way function of the respective randomnesses ν), and to the commitments $\overline{k}$ of the input keys k of P_A. These are the elements that would be revealed in the COMMIT stage if the RSC technique was not in place.
- **Type-RC — *Reveal-for-check* elements.** The *reveal for check* of connectors is achieved by simply revealing the seed λ_j of each check instance (538). Besides the mentioned Type-C elements, P_B is able to derivable from the seed the following elements: for each input wire of P_A, the opening $(k, \underline{k})$ of the two wire keys (506–505), the permutation bit π (501) and the randomness α selected for a corresponding commitment of the permutation

bit (502); for each input and output wire of P_B, an inner randomness ν (510, 514)

- **Type-RE — *Reveal-for-evaluation* elements.** The RSC seed is not revealed for evaluation challenges. Instead of the seed, P_B directly receives some Type-C and the Type-RE elements. The Type-C elements are those described above, including the garbled circuits, the inner Coms ν' for all types of wires, and the commitments of wire keys of P_A. However, only one out of two wire key commitments needs to be revealed for each input wire of P_A (545), because the other one will be derivable from some Type-RE elements. The type RE elements are: the multipliers β for the input and output wires of P_B (546, 548); and, for each input wire of P_A, one inner randomness ν of the permuted bit (i.e., of the position of the permuted key commitment that corresponds to the input bit of P_A) (541–543) and one circuit input key opening $(k, \underline{k})$ (544).

Sketch. The proof of soundness comprehends three steps:

1. §B.3.2.2 shows that if Type-C and Type-RC elements are incorrect (e.g., a GC that encodes an incorrect Boolean circuit) for some challenge index j, then the respective instance cannot withstand the Verify stage (i.e., the incorrectness will be detected by P_B while still allowing a safe abort), unless if P_A can break some cryptographic assumption.
2. §B.3.2.3 shows that if Type-C elements are correct (and consequently also the Type-RC elements known by P_A), then P_A^* is not able to produce *non-complainable bad* Type-RE elements, except if it could break some cryptographic assumption. In other words, if P_A^* acted honestly in the Commit stage for a certain challenge index j (i.e., if it used Type-C and Type-RC elements properly generated from a seed), then any incorrect Type-RE element is detectable by P_B and safely complainable (based on the respective verification procedure also carried out for evaluation challenges).
3. Finally, §B.3.2.4 notices that P_B is able to determine a correct output if for at least one challenge index j the respective Type-C and Type-RE elements are correct. This means that soundness can be broken only if P_A^* produces correct Type-C and Type-RC elements for all *check* indices $j \in J_V$, and incorrect Type-C and/or Type-RC elements for all remaining *evaluation* indices $j \in J_E$. §3.1.4 shows that for several C&C partitioning methods the probability of this event is negligible in the number s of challenges.

B.3.2.2 If Type-C and/or Type-RC elements are incorrect.

Assume a successful verification of the RSC global hash Λ (560) in the VERIFY stage. Then, by correctness of the PRG procedures (e.g., PRGen_{GC} (348), $\text{PRGen}_{\text{InKey}}$ (349), $\text{PRGen}_{\$\text{ForCom}}$, $\text{PRGen}_{\text{BitString}}$ (351)), including the pseudo-random generation of garbled circuits (which uses as input the Boolean circuit specification), the Type-C and Type-RC elements associated with check instances ($j \in J_V$), and derived by P_B from the RSC seeds λ_j revealed by P_A, are correct and do indeed contribute to a valid pre-image of the global hash Λ that P_A was able to open from an Equiv-Com $\overline{\Lambda}$. By reduction to absurd, assume that, when composing the pre-image of the global hash, P_A had used some incorrect Type-C element (e.g., an incorrect garbled circuit) that it would "hope" to be selected for evaluation, but which was instead selected for check. If the VERIFY stage were nonetheless successful (i.e., without P_B noticing any inconsistency), then it means that the incorrect elements would also contribute to a valid pre-image of the committed global hash. Then, this would mean that P_A was able to compute two different pre-images of the same global hash, or two different hashes with the same RSC commitment. Therefor, assuming collision resistance of the hash and binding of the RSC Equiv-Com (binding to a collision-resistant hash implies binding to the known pre-image of the hash), it follows that any incorrect Type-C or Type-RC elements used in the pre-image known by P_A lead P_B to a detect a complainable consistency, i.e., a global hash different from the one opened by P_A.

B.3.2.3 If Type-C and Type-RC elements are correct.

If an index is selected for *evaluation*, then P_A needs to send, in the RESPOND stage, the Type-C and Type-RE elements. Even for *evaluation* challenges, P_B also performs some verifications, associated with the Type-RE elements. The following paragraphs show that if the Type-C and Type-RC elements for a particular instance would be validated by a *check* instance (i.e., if they were correct — §B.3.2.2), then P_A^* is not capable of *forging* a respective response for an *evaluation* challenge, i.e., it is not able to produce incorrect Type-RE elements that would not lead to a valid pre-image of the global hash. The above consideration does not claim anything about the case where the Type-C and/or Type-RC elements would be incorrect (for an index selected for *evaluation*). In fact, if Type-C or Type-RC elements associated with an evaluation challenge are incorrect (e.g., an incorrect GC), then it is possible to have incorrect Type-RE elements pass by undetected (i.e., forgeries). The properties of

forgeries are now examined:

- **Input wires of $\mathbf{P_A}$ ($i \in I_\mathbf{A} \cup I_\mathbf{A}$, — see Illustration 3.4).** For *evaluation* indices ($j \in J_E$), $\mathrm{P_A}$ reveals: the permuted bit ($c_{j,i}$) (541–542); the inner-randomness $\nu_{j,i}$ (543) used to produce the inner-commitment $\nu'_{j,i}$ used in the global hash pre-image; the input wire key $k_{j,i}^{[b_i]}$ and the randomness $\underline{k}_{j,i}^{[b_i]}$ (544) used to produce the key commitment $\overline{k}_{j,i}^{[b_i]}$ in the adequate position $< c_{j,i} >$ of the permuted pair of commitments that was also used in the global hash pre-image; and the complementary commitment $\overline{k}_{j,i}^{[1 \oplus b_i]}$ (545).
  - **Forged permuted bit.** Since it is assumed that the inner Com $\nu'_{j,i}$ (a Type-C element) is correct, a forgery related with a permuted bit $c_{j,i}$ would imply $\mathrm{P_A}$ revealing an inner randomness $\nu_{j,i}$ corresponding to the opening of a different permuted bit (i.e., $1 - c_{j,i}$, for some wire index i). However, this would mean breaking the binding property of the Com scheme ($\mathscr{B}_{\mathrm{ConA}}$). Hereafter it is assumed that $\mathrm{P_A}$ reveals the correct inner randomness $\nu_{j,i}^{(c)}$.
  - **Forged circuit input key.** By assumption of correct Type-C elements, the pair of key-commitments $(\overline{k}_{j,i}^{<0>}, \overline{k}_{j,i}^{<1>})$ commits to two correct keys. Further assuming that the revealed permuted bit ($c_{j,i}$) is correct, then the only remaining possibility of forgery would be for $\mathrm{P_A}$ to open a different (i.e., to equivocate an incorrect) key $\xi_{j,i}^{<c>}$ in the correct position $c_{j,i}$, i.e., to reveal a key-randomness pair $(\xi_{j,i}, \underline{xi}_{j,i})$ that verifies well against the key-commitment (552), but which would be different from the one derived from the RSC seed. However, this would mean breaking the binding property of the intermediate commitment scheme ($\mathscr{B}_{\mathrm{ConA}}$). (More precisely, given the RSC technique, a forgery could also be from an opening whose derivable key commitment would lead to a correct verification of the global hash, but again this would mean breaking collision resistance of the CR-Hash or binding of the RSC Equiv-Com.)

  Thus, if the Type-C and Type-RC elements would withstand a *check* challenge, $\mathrm{P_A^*}$ is bound to respond with correct Type-RE elements for the input wires of $\mathrm{P_A}$, or else be detected in a *complainable* condition.
- **Input wires of $\mathbf{P_B}$ ($i \in I_\mathbf{B}$).** For each *evaluation* index ($j \in J_E$):
  - **If OT if of type 2-out-of-1 (see Illustration 3.5a).** In the RESPOND stage, $\mathrm{P_A}$ reveals each possible bit value ($c \in \{0,1\}$), an independent *multiplier* $\beta_{j,i,c}^{(0)}$ (randomness for committing 0) (546) that leads the intermediate randomness ($\mu_i^{(c)}$) of the bit (of which $\mathrm{P_B}$ knows one and $\mathrm{P_A}$ knows both) into the respective independent *inner randomness* $\nu_{j,i,c}^{(c)}$ (a proper square-root of the respective *inner BitCom*) of the bit. For each possible

bit value $c \in \{0,1\}$ in each input wire $i \in I_{\mathrm{B}}$, there is only one *multiplier* ($\beta_{j,i,c}$) that is simultaneously a randomness for committing 0 and a proper opening of the multipler BitCom $\beta'_{j,i,c}$ (554). $\mathrm{P_B}$ verifies correctness homomorphically (which could be done even if it did not know any intermediate randomness $\mu_{j,i}^{(c)}$), by calculating the two multiplier BitComs $\beta'_{j,i,c}$ of 0 (554), then multiplying each of them by the intermediate BitCom $\mu'_{j,i}$ (555), and checking that the resulting pair of inner BitComs $\nu'_{j,i}$ contributes to a correct pre-image of the global hash. Thus, $\mathrm{P_A^*}$ cannot not produce forged multipliers for input wires of $\mathrm{P_B}$. Furthermore, assuming that the Type-C elements are all correct, if follows that the revealed multipliers allow $\mathrm{P_B}$ to obtain one correct input key per input wire of $\mathrm{P_B}$.

  - **If OT if of type 1-out-of-2 (see Illustration 3.5b).** In the RESPOND stage, $\mathrm{P_A}$ reveals to $\mathrm{P_B}$ an inner Com $\nu'_{j,i}$ (547) that was obtained by homomorphically encrypting (probabilistically), on top of the intermediate OT BitCom μ'_i of the input bit of $\mathrm{P_B}$, a linear combination of two random multipliers ($\beta_{j,i,0}$, $\beta_{j,i,1}$) (512- 513). This inner Com is the value that is used directly as a contribution to the pre-image of the global hash (556), so it does not involve any further verification by $\mathrm{P_B}$. Since it is being assumed here that the Type-C elements (including the inner Coms) are correct, it follows that an honest $\mathrm{P_B}$, who knows the decryption key (a trapdoor of the intermediate OT BitCom scheme $\mathscr{B}_{\mathrm{OT}}$) and has previously provided (a verifiably or proven) correct commitment of a bit (i.e., 0 or 1), is able to decrypt the image $f(\beta_{j,i,b_i})$, under a particular one-way function f, of the multiplier corresponding to the committed input bit b_i of $\mathrm{P_B}$.

- **Output wires of $\mathbf{P_B}$ ($i \in \mathbf{O_B}$ — see Illustration 3.6).**
  For each evaluation index $j \in J_E$, the garbled circuit is by assumption correct; also, as discussed above, the circuit input keys $k_{j,i}$ are assumed to be correct. Thus, $\mathrm{P_B}$ is able to evaluate the garbled circuit to obtain one and only one correct key $k_{j,i}^{[b_i]}$ per output wire $i \in \mathrm{O_B}$ (565). Assuming (see §3.3.4) that each output wire key reveals the respective underlying bit c, the output wire key is then transformed via a (also using the extra auxiliary seed $\lambda_j^{(auxi)}$) into a single inner randomness $\nu_{j,i,c}$ (568), which is verified as being a proper opening of the respective inner BitCom $\nu'_{j,i,c}$ (569) (also assumed to be correct). Thus, the only possible forgery would be for $\mathrm{P_A^*}$ to reveal an incorrect multiplier in the RESPOND stage. However, similarly to the case of input wires of $\mathrm{P_B}$, the multiplier (randomness $\beta_{j,i,c}$ for committing 0 (554)) can be verified homomorphically, by checking the relation between the respective BitCom $\beta'_{j,i,c}$, the inner BitCom $\nu_{j,i,c}$ and the intermediate

BitCom μ'_i previously proposed by P^*_A(555), even without P_B knowing the respective inner encodings of the output wire. Furthermore, since the BitCom scheme is assumed to only have one possible valid randomness per committed bit (e.g., in case of Blum BitComs the square-roots have to be proper), it follows that there is no proper forgery.

B.3.2.4 Decision of final output.

Above, it was shown that (i) incorrect Type-C and/or Type-RC elements are detected as incorrect if they are selected for *check*; and (ii) that correct Type-C and Type-RC elements selected for *evaluation* either lead to a correct output (if the Type-RE elements are correct) or allow detection of incorrect (and complainable) responses. Thus, to lead P_B into accepting an incorrect output, P^*_A needs to produce incorrect Type-C and Type-RC elements, and be lucky that none becomes associated with a check challenge. Furthermore, in order for the bad instances selected for evaluation to be validated by the *reveal for evaluation* mode, they need to lead P_B to find one valid randomness of the forge-and-lose output BitCom of each output bit. However, by combining a forged (i.e., undetected and incorrect) output with a correct output, P_B can obtain the trapdoor of P^*_A (a pair of different openings to the same forge-and-lose BitCom of an output wire; e.g., in IFC a pair of non-trivially correlated square-roots of the same square, or in DLC a pair of different representations of the same Pedersen BitCom), and thus activate the forge-and-lose evaluation path to obtain a correct final output. Thus, the only way that P_A has to lead P_B to accept an incorrect output or to abort in the Evaluate stage, after a successful Verify stage, is to guess in advance exactly what indices will be selected for evaluation and then build incorrect elements for all these and only for these indices. As shown in §3.1.4, there is a negligible probability of such guess being correct, for practical C&C partition parameters.

B.3.3 Simulator for the case of malicious P^*_B

The simulator $\mathcal{S} \equiv \mathcal{S}^{\mathrm{B}^*}$, with access to a black-box malicious P^*_B, simulates the beginning of a real protocol execution. $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_\mathrm{A}]$ impersonates a real honest P_A while interacting in the simulated execution with the malicious P^*_B. As the simulation proceeds, if P^*_B takes actions that would make an honest P_A abort, then $\mathcal{S}^{\mathrm{B}^*}[\widehat{\mathrm{P}}_\mathrm{B}]$ (in the role of ideal $\widehat{\mathrm{P}}^*_\mathrm{B}$ in the ideal world) sends `abort` to the TTP. In this case the TTP sends an `abort` message to $\widehat{\mathrm{P}}_\mathrm{A}$, which

leads $\widehat{\mathrm{P}}_{\mathrm{A}}$ to also abort in the ideal world (i.e., send an `abort` message to $\mathcal{Z}$). Then $\mathcal{S}^{\mathrm{B}^*}[\widehat{\mathrm{P}}_{\mathrm{B}}]$ outputs in the ideal world whatever $\mathrm{P}^*_{\mathrm{B}}$ outputs in the simulated execution.

1. **Extract trapdoor and circuit input of $\mathbf{P}^*_{\mathbf{B}}$.** In the Produce initial BitComs of P_{B} stage, $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ receives the outer Com of the input bit-string of P_{B} and the Com of the offset bit-string for the future private output bit-string of P_{B}, and, from the respective ZKPoK (of trapdoor and/or of openings), extracts the circuit input bits of P_{B} and respective openings (454–468). If any NIZKP or NIZKPoK is invalid, then $\mathcal{S}^*_{\widehat{\mathrm{B}}}$ emulates an abort in the ideal world (in the role of ideal $\widehat{\mathrm{P}}^*_{\mathrm{B}}$).
2. **Initiate coin-flipping protocol.** The simulation enters the Coin-flip Permutations (Start) stage, where $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ receives from P_{B} the initial message of the coin-flipping of permutations, which contains an Ext-and-Equiv Com of contributions $(\gamma^{(\mathrm{B})}_{I_{\mathrm{A}}}, \gamma^{(\mathrm{B})}_{\mathrm{O}_{\mathrm{A}}})$ of permutations for the outer-randomness for sets of input and output wires of P_{A}, and of contributions $(\gamma'^{(\mathrm{B})}_{I_{\mathrm{B}}}, \gamma'^{(\mathrm{B})}_{\mathrm{O}_{\mathrm{B}}})$ for the permutation of the initial outer-Coms for sets of input and output wires of P_{B}. The coin-flipping message also contains a committed ZKPoK of the openings of the contributions of P_{B} to the permutation of the initial outer-Coms for sets of wires of P_{B} (472–474). If $\mathrm{P}^*_{\mathrm{B}}$ has done anything that would make an honest P_{A} abort, then $\mathcal{S}^*_{\widehat{\mathrm{B}}}$ emulates an abort in the ideal world.
3. **Interact with the TTP.** $\mathcal{S}^*_{\widehat{\mathrm{B}}}$ in the role of the ideal $\widehat{\mathrm{P}}^*_{\mathrm{B}}$ sends to the TTP the private circuit input that was extracted from the initial outer Coms of $\mathrm{P}^*_{\mathrm{B}}$ (455, 468). The TTP then makes the necessary internal procedures and returns the output first to $\mathcal{S}^*_{\widehat{\mathrm{B}}}$ (404), which includes the outer-Coms of the inputs and outputs of $\widehat{\mathrm{P}}_{\mathrm{A}}$ and $\widehat{\mathrm{P}}_{\mathrm{B}}$, the outer-randomnesses for the input and output bits of $\widehat{\mathrm{P}}_{\mathrm{B}}$, and the circuit output y_{B} of $\widehat{\mathrm{P}}_{\mathrm{B}}$.
4. **Enforce the BitCom values selected by the TTP.** Upon receiving the output from the TTP ($\mathcal{F}_{\mathrm{S2PCwC}}$), $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ needs to induce that output to the malicious $\mathrm{P}^*_{\mathrm{B}}$ in the simulated execution, except of course if $\mathrm{P}^*_{\mathrm{B}}$ aborts.

   In regard to the outer Coms and outer-randomnesses of sets of wires of P_{A}, $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ knows: (i) the outer-randomness permutations γ extracted from the Ext-and-Com commitment provided by $\mathrm{P}^*_{\mathrm{B}}$; (ii) the final outer-Coms ρ' (but not the respective outer-randomnesses) decided by $\mathcal{F}_{\mathrm{S2PCwC}}$. (iii) the common circuit output bits. $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ uses these elements to calculate what needs to be the initial outer-Com $\sigma'_{I_{\mathrm{A}}}$ of input of P_{A} (476) and the outer-Com $\sigma'_{\mathrm{O}_{\mathrm{A}}}$ for output bit-masks of P_{A} (479), so that a final adjustment and permutation will lead into the final outer-Coms ρ' decided by the TTP. $\mathcal{S}^{\mathrm{B}^*}[\mathrm{P}_{\mathrm{A}}]$ resumes the simulation and in the Outer BitComs of input bits of P_{A} stage sends those outer-Coms to $\mathrm{P}^*_{\mathrm{B}}$,

and (if need-be, e.g., for DLC) forges a respective NIZKPoK of openings (490) and NIZKP of correct BitComs (477,481).

In regard to the outer Coms and outer-randomnesses of sets of wires of P_B, $\mathcal{S}^{B^*}[P_A]$ knows: (i) the initial outer-randomness σ extracted from the ZKPoK of opening of the initial outer-Com of input of P_B (468); (ii) the final outer-randomness ρ (and also outer-Coms ρ') decided by the TTP ($\mathcal{F}_{\text{S2PCwC}}$). $\mathcal{S}^{B^*}[P_A]$ uses these elements to calculate the needed outer-randomness permutations γ that are needed to induce P_B^* to accept the final outer-randomnesses ρ decided by $\mathcal{F}_{\text{S2PCwC}}$. $\mathcal{S}^{B^*}[P_A]$ impersonates the ideal $\mathcal{F}_{\text{GMCF-1}}$ to output the calculated outer-randomness to P_B^* (536).

In regard to the circuit output y_B, $\mathcal{S}^{B^*}[P_A]$ equivocates the opening of the RSC Equiv-Com (549), to yield a hash of the tuples composed of correct elements for the check instances (derived by respective random seeds), and of incorrect elements (e.g., garbled circuits) for the evaluation instances, such that the output of the evaluation garbled circuits is the circuit output decided by $\mathcal{F}_{\text{S2PCwC}}$.

5. **Finalize the execution.**

   If P_B^* decides to do a correct execution, it will proceed by transmitting to P_A the masked private output bits of P_A and the common output bits, and to correctly open his contribution in the last message of the coin-flipping of permutations (i.e., opening her contributions and the NIZKPoK), and giving a NIZKP that the outer-Com permutation contribution for the wires of P_B are commitments to all zeros. $\mathcal{S}^*_{\widehat{B}}$ checks the validity of all received values. If they are correct, then in the ideal world it sends an `OK` message to the TTP (383), otherwise it sends an `abort` (384).

   Finally, $\mathcal{S}^*_{\widehat{B}}$ outputs in the ideal world whatever the P_B^* outputs in the simulated execution.

Analysis of the simulation. The described simulation leads the parties in the ideal world to produce an output indistinguishable from the corresponding one in the real world. Essentially, simulatability is reduced to the extractability of the openings (circuit input bits and respective randomnesses for commitment) of outer Coms of P_B^*, and of the respective committed contributions to permutations, and of the outer-randomness of the committed outer-Coms contributions, from the respective committed ZKPoK. and the equivocability of the RSC challenge from the respective Equiv-Commitment.

The type-RE elements cannot be derived directly from the Type C elements. Unless P_B^* could break the hiding property of commitments, P_B^* cannot open commitments ($\overline{k}$) of the

circuit input keys (k) of the input wires of $\mathrm{P_A}$; since the inner BitComs are hiding, $\mathrm{P_B^*}$ does not learn anything about the permutation bits;

in the case of a 2-out-of-1 OT, $\mathrm{P_B^*}$ cannot compute the multipliers (β) of wires of $\mathrm{P_B}$, unless it could calculate modular square-roots (i.e., if it could find the integer factorization of the Blum integer modulus); in case of a 1-out-of-2 OT, $\mathrm{P_B^*}$ can really only extract one multiplier from the OT answer from the honest $\mathrm{P_A}$.

B.4 Optimizations and other details about connectors

This section describes optimizations to the connectors.

B.4.1 Connectors of input of $\mathrm{P_A}$

B.4.1.1 Additive pseudo-homomorphism of bit-string Coms

Based on the idea of XOR pseudo-homomorphism, a further optimization is possible based on an additively-homomorphic commitment scheme $\mathscr{C}_\mathrm{A}$ (e.g., Pedersen or ElGamal), using it to directly commit to bit-strings, i.e., more compactly than a concatenation of BitComs. The optimization being considered still makes use of BitComs of the input bits of $\mathrm{P_A}$. However, the goal is to use BitStringComs to improve the communication and computational complexity associated with connectors (which overall have a complexity proportional to the number of evaluated garbled circuits), and also the complexity associated with the needed random permutation of commitments for the purpose of producing final random commitments.

The (multiplicative) group operation $*'$ in the commitment space depends on the commitment scheme. For example: for Pedersen commitments it corresponds to multiplication of the group elements, resulting in a new group element; for ElGamal commitments it corresponds to the pair-wise multiplication of the two components (group elements), resulting in a new pair of components. With Blum and GM BitComs the group operation in the randomness space (i.e., the space of square-roots) uses multiplicative notation. With Pedersen and ElGamal BitComs the group operation in the randomness space (i.e., the space of exponents) uses additive notation. For generality, when considering a generic BitCom scheme the notation $\overset{+}{*}$ in the randomness space may simultaneously show the multiplicative and additive options.

Pre-condition. The input of P_A can be encoded as a bit-string x_A (616). Correspondingly, an adequately weighed combination (617) of the randomnesses μ_i of the respective BitComs μ'_i results in a commitment μ_A of the input string of P_A (618). If the input is longer than the length allowed by the commitment scheme, then the bit-string encoding can be partitioned across several bit-strings, e.g., with 128 bits per bit-string. Also, as a pre-condition it is assumed that P_A proves that the BitCom $\mu_{j,i}$ of each input bit is indeed a BitCom, i.e., that it commits to a 0 or a 1.

Commit. Regarding the connectors, the idea now is that all inner BitComs $\nu'_{j,i}$ of a particular challenge instance j can be condensed into a single BitStringCom ν'_j. First, P_A samples a random BitString permutation $\pi_j \leftarrow \{0,1\}^{\ell_A}$ (619), which essentially corresponds to a sequence of the isolated random bit permutations $\pi_{j,i}$. Then, P_A uses the additively homomorphic BitStringCom scheme $\mathscr{C}_{\text{InA}}$ to produce a BitStringCom α'_j of each bit-string permutation π_j (620), with the respective randomness α_j being a random integer (a.k.a. scalar, typically an exponent) between 1 and the order of the group. For example: if using Pedersen Commitments, then P_A must not known the trapdoor, but P_B may (but does not need to) know it; if using ElGamal Commitments, then P_B may (but does not need to) know the trapdoor, and P_A must not know it.

The construction uses the additive homomorphism to enable a bit-string XOR pseudo-homomorphism. Each connector (now one per challenge index j) has an associated permuted string c_j, which is equal to the bit-wise XOR of the input string x_A of P_A and the permutation string π_j of the challenge index (621). The bit-wise XOR can also be obtained via a weighed integer sum bit by bit, with the weighing not only depending on the position of the bit, but also on the value of each input bit of P_A (622). Associated with the permuted bit c_j there is a corresponding BitStringCom ν'_j (623), denoted *inner* BitStringCom, which can be obtained pseudo-homomorphically from the BitStringCom α'_j of the permutation string π_j and the BitComs μ'_i of the input bits of P_A. Given the additive homomorphism, the "randomness" ν_j associated with inner BitStringCom can also be obtained pseudo-homomorphically from the randomness α_j of the commitment of the permutation string π_j, and a weighed sum of the "randomness" μ_i of the BitComs of the input bits of P_A, with the (signal of the) weighs depending on the permutation bits $\pi_{j,i}$ (624). Given the homomorphic properties, the inner BitStringCom ν'_j can be obtained directly from the combination of the permuted string c_j and the respective pseudo-homomorphically obtained "randomness" ν_j (625).

$$x_A \equiv \sum_{i\in[\ell_A]} 2^{i-1} b_i \quad (616)$$

$$\mu_A \equiv \sum_{i\in[\ell_A]} 2^{i-1} \mu_i \quad (617)$$

$$\mu'_A = \mathscr{C}_{\mathrm{InA}}(x_A, \mu) = \mathop{\divideontimes'}_{i\in[\ell_A]} (\mu'^{2^i}_i) \quad (618)$$

$$\pi_j \equiv \sum_{i\in[\ell_A]} 2^{i-1} \pi_{j,i} \quad (619)$$

$$\alpha'_j = \mathscr{C}_{\mathrm{InA}}(\pi_j, \alpha_j) \quad (620)$$

$$c_j = \pi_j \oplus x_A \quad (621)$$

$$\equiv \pi_j + \left(\sum_{i\in[\ell_A]} (-1)^{\pi_{j,i}} 2^{i-1} b_i\right) \quad (622)$$

$$\nu'_j = \alpha'_j *' \left(*_{i\in[\ell_A]} (\mu'_i)^{2^{i-1}(1-2\pi_{j,i})}\right) \quad (623)$$

$$\nu_j = \alpha_j + \left(\sum_{i\in[\ell_A]} (\mu_i) 2^{i-1} (-1)^{\pi_{j,i}}\right) \quad (624)$$

$$\mathscr{C}_{\mathrm{InA}}(c_j, \nu_j) \equiv \nu'_j \quad (625)$$

Figure B.12: **Optimization of connectors InA — from BitComs to BitStringComs.** For simplicity of notation it is here assumed that $I_A = [\ell_A] \equiv \{1, ..., \ell_A\}$. For concreteness, equations (617,624) are shown with additive notation, but there may be multiplicative instantiations as well. Intuitively, equations (623, 624) could be expressed as $\alpha'_j *'_{(\pi_j)} \mu'_A$ and $\alpha_j +_{(\pi_j)} \mu_A$, respectively, with the subscript (π_j) in the group operation symbol denoting the pseudo homomorphism, varying accordingly to the respective permutation π_j — however, in fact the operations require knowledge of the BitComs μ'_i and respective randomness μ_i.

Reveal for check. For each *check* instance ($j \in J_V$), P_A reveals the respective RSC seed λ_j to P_B. This gives P_B the ability to: pseudo-randomly determine the permutation bits $\pi_{j,i}$ and from there determine the permutation string π_j (619); pseudo-randomly determine the "randomness" α_j needed to commit the permutation bit, and from there compute the respective commitment α'_j (620) (needed to check a correct pre-image of the RSC hash). The permutation bits $\pi_{j,i}$ allow P_B to check whether the revealed keys $k^{\langle b\rangle}_{j,i}$ are properly permuted.

Reveal for evaluation. For each *evaluation* instance ($j \in J_E$), P_B reveals the permuted bit-string c_j (621) and the encoding ν_j (i.e., the "randomness") of the respective homomorphic commitment (624). This also allows P_B to determine the inner BitStringCom ν'_j, also needed to check the correctness of the RSC hash. The permuted string c_j informs the position $c_{j,i}$ of the input key in the respective wire i of each garbled circuit j that P_A should open, from within the randomly permuted pair of key commitment, but without informing what is the respective underlying bits.

Complexity advantages. In this construction the number of exponentiations is reduced to just one per challenge index $j \in [s]$, apart from two extra multiplications per pair (i, j) of challenge index $j \in [s]$ and input wire index $i \in I_A$. In terms of exponentiations this is approximately as good as the described IFC instantiations based on Blum BitComs or GM BitComs. It is conceivable that a similar communication optimization can be achieved with an IFC-based additively homomorphic Com scheme, e.g., based on Pailier encryption

[Pai99] or an encryption scheme that generalizes the Goldwasser Micali encryption scheme to bit-strings (e.g., [JL13]), but a concrete instantiation is not explored in this dissertation.

B.4.1.2 Blum-based BitString Com scheme

This subsection describes a generalization of the Blum BitCom scheme to a compact equivocable commitment scheme for bit-strings, taking advantage of the existence of a principal square-root of each square, i.e., a square-root that is itself a square. Similarly to the described Blum BitCom scheme, let there be an auxiliary public group element z with Jacobi Symbol -1 (possibly the smallest), and let its respective square z' also be pre-computed. Let there also be a fixed length k, e.g., 256, smaller than the size of the Blum integer, defining the size of the bit-string being committed. (The Blum BitCom scheme is the case of length 1.) A description with succinct notation is presented in row 5 of Table 2.2.

Procedure.

- **Commit.** To commit a non-negative integer m, smaller than two raised to the power of the fixed length k (cell C5), the sender (i) computes a first power by modularly raising the auxiliary square z' to the power of the integer m being committed; then (ii) computes a second power by selecting a random multiplicative residue r (cell E5) and modularly raising it to the power of two raised to the power of the length k plus 1; and finally (iii) sends the modular product of the two powers to the receiver (cell E5).
- **Open type 1.** To open, the sender reveals the randomness r and the committed value m (cell G5). The receiver accepts only if it recomputes the expected commitment (cell H5).
- **Open type 2.** To open without revealing the randomness and without knowing the trapdoor, the sender: (i) reveals the committed value; then (ii) computes the quotient between the commitment and the auxiliary square z' to the power of the integer m being committed; and (iii) sends a NIZKPok of the 2^{k+1}-th root of the obtained quotient.
- **Efficient equivocation with trapdoor.** A simulator knowing the trapdoor is able to equivocate the opening of any value m', by simply sending m, and the 2^{k+1}-th root of the modular quotient between the commitment and the m-th power of the auxiliary square z'.

Analysis.

- **Binding.** Breaking the biding property implies the ability to factor the Blum integer.

Suppose that for a certain commitment c the sender knows two valid openings to distinct values, i.e., a pair (m, m') of distinct messages and a pair of randomnesses (r, r'), such that a pair composed of the first message and the first randomness leads to the same commitment as the commitment obtained from the second message and second randomness. The sender is able to manipulate the equation of commitments to isolate in the left side a power of the quotient of randomnesses, and in the right side a power of the difference of messages. In the left side, the exponent applied to the (quotient of) randomnesses remains being two raised to the power of the fixed length k. In the right side, the exponent that is equal to twice the difference of messages is less than two raised to the power of the fixed length. Upon removing from both sides the even factor of exponent present in the right side, the sender obtains in both sides (no longer equal) square-roots of the same value, with the left-side one having Jacobi Symbol 1, and the right-side one having Jacobi Symbol −1 (because it is an odd power of the auxiliary element z). This pair of non-trivially correlated square-roots allows calculation of the Blum integer trapdoor. To ensure that the sender is not able to find a short cycle, the Blum integer can be chosen such that the Euler totient of each prime factor of the Blum integer has an exponentially large smallest odd prime, e.g., by selecting a Blum integer as a product of two safe primes (i.e., each equal to one plus twice a prime).

- **Hiding and equivocability.** Any commitment has a possible opening to any integer with the allowed length. Also, a simulator is able to equivocate the opening of any intended value, by computing the respective needed randomness. With the trapdoor, a simulator is able to take the commitment and divide it by the term corresponding to the square of the auxiliary value to the power of the intended value to decommit. Then, the simulator iteratively computes square-roots a number of times equal to the length of the committable space. The result is the randomness needed to equivocate the intended value.
- **Additive homomorphism.** The multiplication of two commitments values yields a new commitment of the sum of the two originally committed values, modulo two raised to the power of the fixed length. The randomness of the product commitment is the product of the two original randomnesses, further multiplied by the auxiliary element if and only if the sum of the messages exceeds the allowed length.

B.4.1.3 Shorter connectors, based on short-term binding

While the outer BitComs of input bits of P_A require long-term security (binding and hiding), the inner BitComs and the multiplier BitComs used in connectors of input of P_A are used only to ensure that P_A (who knows her own input bits) opens the (one out of two) correct input garbled keys to P_B. The respective binding properties are only needed during the protocol execution, in contrast with the hiding properties that need to be ensured in the long term. Thus, if it can be assumed that the protocol duration is short, the security parameter associated with short-term binding of these connectors can be shortened, with respective complexity improvements. In other words, the cryptographic security related to the integrity of connectors can be reduced, while retaining the confidentiality of the input bits of P_A. For example, if it is adequate to define a time-out duration of 1 minute for an oblivious AES-128 evaluation (e.g., see some benchmarks take about 1 second to execute [KSS12]), then it may be valid to use shorter Blum integers assumed not to be factorable in such amount of time. One could propose to use original (long) BitComs with 3,248 bits in size for BitComs of the input bits (assumed equivalent to 128 bits of cryptographic security), and new (short) BitComs with 1,776 bits for use for the connectors of P_A (assumed equivalent to 96 bits of cryptographic security). This paragraph merely intends to exemplify the effects that a smaller modulus for the connectors of P_A may have in the communication complexity of the protocol, but not to discuss what are the shorter adequate sizes. Naturally, the optimization is worth only if the complexity arising from the new unconditionally hiding BitComs and the extra ZKP does not exceed the communication and/or computation savings obtained in the connectors. Given the improvement already obtained with the use of bit-string Coms, the "short term binding" optimization is mostly overshadowed, but is here mentioned to highlight one more tradeoff between security properties.

B.4.2 Connectors of input of P_B

B.4.2.1 ElGamal-based 1-out-of-2 OT for honest-sender

In the S2PC-with-Coms protocol in this dissertation, oblivious transfers are used to sustain connectors for the input of P_B, allowing a connection between BitComs and keys of garbled circuits. The following paragraph describes a straightforward approach for a single round 1-out-of-2 OT based on ElGamal BitComs, facilitated by the underlying additive homomorphism of

ElGamal encryption. Considering its application integrated with the cut-and-choose approach of the S2PC-with-Coms protocol, the protocol does not have to be simulatable against a malicious sender, and so can be simpler than the PVW protocol [PVW08].

The goal is to let P_A send (*a la* OT) a pair of exponentiations (ν_0, ν_1), denoted *inner values*, for which it knows the respective random discrete-logs (β_0, β_1), base the global generator g_0, such that P_B only receives one exponentiation (the result of raising the global generator g_0 to the power of the exponent β_b with index equal to the bit of P_B, without P_A knowing which). For notation consistency across different connectors, the exponents of the inner values are also denoted as *multipliers*. The basis of the method is to have P_A use a linear combination of two multipliers to homomorphically transform an ElGamal BitCom produced by P_B, such that P_B is then able to decrypt the result of exponentiating the base generator to the power of the multiplier with index equal to the input bit of P_B. P_A does not need to give a ZKP of correct transformation, because in the S2PC-with-BitComs application the respective verification is performed statistically based on the cut-and-choose structure.

Procedure. The 1-round 1-out-of-2 OT protocol proceeds as follows.

- **Setup.** The protocol is based on an ElGamal BitCom scheme. Since the scheme will also be used as an ElGamal encryption scheme, it is imperative that P_B knows the secret decryption key x (626). The public parameters form a pair (g_0, g_2) of generators, such that the decryption key is the discrete log of the second generator g_2 base the first generator g_0 (627). This pair is known by both parties. In practice, they may be decided by P_B and sent to P_A along with the first message of an OT, or they may be part of a trusted setup that also gives to P_A the respective trapdoor.
- **Input.** The parties are activated to play an 1-out-of-2 OT. As input, P_B is activated with a bit of input (628). P_A is activated with two exponents (β_0, β_1) (629), i.e., two *multipliers* (i.e., exponents) (in the S2PC-with-BitComs protocol these exponents are called *multipliers* and will be used to generate respective input wire keys of a garbled circuit). This protocol is actually designed to "transmit" (a la OT) the result of raising the first generator to the power of the respective multipliers.
- **Step 1 ($P_B \rightarrow P_A$).** For each input bit of P_B, P_B (the receiver in the 1-out-of-2 OT) produces and sends a respective ElGamal BitCom of his bit b (630, 631), and then sends a respective ZKP of correctness (632) (e.g., see §A.3.2). On its own this ZKP is not intended

Setup.

$\mathrm{P_B} : x = \mathrm{DL}_{g_0}(g_\mathrm{B})$ (626)

$\mathrm{P_A}, \mathrm{P_B} : (g_0, g_\mathrm{B})$ (627)

Input.

$input_\mathrm{A} \to \mathrm{P_A} : (\beta_0, \beta_1)$ (628)

$input_\mathrm{B} \to \mathrm{P_B} : b$ (629)

Commit to bit.

$\mathrm{P_B} : \mu \leftarrow^{\$} \mathbb{Z}_q$ (630)

$\mathrm{P_B} \to \mathrm{P_A} : (c_1, c_2) \equiv \left({g_0}^r, {g_0}^b {g_\mathrm{B}}^\mu\right)$ (631)

$\mathrm{P_B} \to \mathrm{P_A} : \mathrm{NIZKP}_{\mathrm{ElGamalBitCom}}(c_1, c_2)$ (632)

Homomorphically encrypt value.

$\mathrm{P_B} : \beta_0, \beta_1, r \leftarrow^{\$} \mathbb{Z}_q$ (633)

$\mathrm{P_A} : c'_1 = {c_1}^{\beta_1 - \beta_0} {g_0}^r$ (634)

$\mathrm{P_A} : c'_2 = {c_2}^{\beta_1 - \beta_0} {g_0}^{\beta_0} {g_\mathrm{B}}^r$ (635)

$\mathrm{P_A} \to \mathrm{P_B} : \left(c'_1, c'_2\right)$ (636)

Output.

$\mathrm{P_A} \to output_\mathrm{A} : c_1, c_2, r$ (637)

$\mathrm{P_B} \to output_\mathrm{B} : \nu_b \equiv {g_0}^{\beta_b} \equiv c'_2/(c'_1)^x$ (638)

Figure B.13: **A 1-out-of-2 OT protocol based on an ElGamal BitCom scheme.** The output of the receiver ($\mathrm{P_B}$) is not one of the elements (β_0, β_1) of the input of the sender ($\mathrm{P_A}$), but rather the result of exponentiating the base generator g_0 to the power of one (β_b) of the input elements. The protocol is not secure against a malicious $\mathrm{P_A}$ — and it does not need to be for its application within a cut-and-choose structure (§3.3.2.2). Several components could be more generally described based on an underlying homomorphic encryption scheme: $(c_1, c_2) = \mathscr{E}[\mu](b)$; $(c'_1, c'_2) = \mathrm{RandLHT}[\beta_0, \beta_1, r]((c_1, c_2))$; $\nu_b \equiv {g_0}^{\beta_b} = \mathscr{E}^{-1}((c'_1, c'_2))$.

to provide extractability. If the public parameters (i.e., both generators g_0 and g_2) and the respective trapdoor are decided by a trusted setup (as implicitly assumed in Figure B.13), then extractability can be directly assumed possible, because the simulator himself would decide the trapdoor. If, instead, g_2 is selected by $\mathrm{P_B}$, then $\mathrm{P_B}$ may give a ZKPoK of the trapdoor, possibly a NIZKPoK (see §A.3.3), independent of the number of needed OTs. Alternatively, if more efficient for concrete instantiations, $\mathrm{P_B}$ could instead give a ZKPoK of each committed bit (i.e., one such ZKPoK per committed bit).

- **Step 2 ($\mathbf{P_A \to P_B}$).** Each ElGamal BitCom received by $\mathrm{P_A}$ is also an ElGamal encryption, of either the group identity ${g_0}^0$ (i.e., a commitment of bit 0) or of the first generator g_0 (i.e., a commitment of bit 1). $\mathrm{P_A}$ then prepares to compute a randomized version of an homomorphic transformation of the ElGamal ciphertext. Specifically, $\mathrm{P_A}$ selects as randomizer a new random exponent r' (633), and uses the pair (β_0, β_1) of multipliers (i.e., the linear-transformation exponents) to compute a randomized encryption of an exponentiation of a respective linear combination of the unknown committed value. This is done using the multiplicative homomorphic property of the ElGamal encryption scheme (respectively an additive homomorphic property of the committed exponents). The resulting committed exponent is the one (β_b) with index equal to the bit encrypted by $\mathrm{P_B}$. Specifically, the new first component c'_1 of the ElGamal encryption is computed from the

first component c_1 of the original encryption by exponentiating to the difference $\beta_1 - \beta_0$ of the two multipliers and then multiplying by the first generator g_0 exponentiated to the first multiplier β_0 (634). The second component c_1' is obtained similarly, but additionally multiplying the second generator g_2 to the power of the first multiplier β_0 (635). P_A sends the resulting transformed pair — an ElGamal ciphertext — to P_B (636). Even though the ciphertext has the form of an ElGamal commitment of one of the multipliers (β_b), the multiplier is not semantically hidden, because P_B to decrypt the ciphertext and find the result of raising the base generator g_0 to the power of the multiplier.

- **Output.** As output of the protocol, P_A outputs the bit commitment of P_B, and the randomization used in the homomorphic linear transformation (637). P_B outputs the decryption of the value received from P_A, which is the exponentiation ${g_0}^{\beta_b}$ by the multiplier with index equal to the input bit of P_B (638). Even though P_B cannot compute the discrete log (i.e., the multiplier), the exponentiation is not (and is not inteded to be) a commitment, because it is not semantically hiding.

Remark B.5 (Trading BitComs for pre-images). The IFC-based 2-out-of-1 requires communication of two pre-images per connector of each evaluation instance, but the pre-images are as large as BitComs. Conversely, in FFC the pre-images are much shorter, e.g., 256 bits instead of 3,248 bits, but the (single-round) 1-out-2 OT technique requires communication of one group element (instead of two pre-images) per connector. A conceivable improvement for FFC-based connectors may arise from a combination of ideas from the 2-out-of-1 and the 1-out-of-2 OT methods, to change the communication per connector to two pre-images.

B.5 More details on communication complexity

This section details better the communication complexity of the S2PC-with-Coms protocol.

B.5.1 Concrete parameters for 128 bits of security

Table B.1 proposes sizes for diverse parameters (row 1), for 128 bits of cryptographic security (row 2).

Group for BitComs and pre-images. The size of group elements supporting BitComs depends on the type of intractability assumption (row 3: IFC based on Blum integers requires $n = 3,248$ bits per group element [SBC+12, Table 7.2]; for DLC the sizes vary with the concrete type of group — FFC requires group elements as large as IFC (i.e., 3,248 bits), whereas ECC allows much shorter 256-bit elements (only twice the security parameter size). The size n' of pre-images (e.g., square-roots or exponents), under the respective one-way function (e.g., squaring for IFC or exponentiation for DLC), also varies with the type of instantiation: $3,248$ for IFC and 256 for DLC (both FFC and ECC) (row 4). Interestingly, the sizes associated with FFC lies between those of IFC and ECC, in the sense of having group elements as large as in IFC, but short pre-images as in ECC. In practice, this means larger size for outer BitComs (without much impact in the overall size), and low size for some types of connectors (input of P_A and output of P_B), where it matters most.

CR-Hash (row 5). A CR-Hash function can be instantiated by the SHA-256 function, with $\kappa_{\text{Hash}} = 256$ bits of output, for 128 bits of security under birthday attacks.

PRG (row 6). Pseudo-random generation of elements can be instantiated via pseudo-random number generators based on block-ciphers. For simplicity of security considerations, AES-256 is used instead of AES-128, ensuring a birthday bound of 128 bits in respect to ever repeating the same entropy across two uses. It requires a seed with at least $\kappa_{\text{PRG}} = 256$ bits of entropy. The actual internal seed may be larger (e.g., see [BK15, §10.2, Table 3] for bit-string generation based on AES-256, requiring a seed length of 384 bits). Nonetheless, in practice the seed can be derived from an initial random bit-string of 256 bits, and if useful also on additional input. The input of the PRG generations in the S2PC-with-Coms protocol also includes a counter and/or identifier, used to differentiate the specific procedures within the same protocol, and a specification of the intended output set (e.g., a bit-string length, or a the specification of an IFC or DLC group) (see Remark B.6 below).

Remark B.6 (Construction of PRGs and entropy of seeds). The PRG procedures include: the generation (PRGen_{GC}) of garbled circuits (517); the generation ($\text{PRGen}_{\text{InKey}}$) of input wire keys of P_A (506); the generation ($\text{PRGen}_{\text{BitString}}$) of "random" bit-strings (e.g., for permutation bits) (501); the generation ($\text{PRGen}_{\$\text{ForCom}}$) of "randomness" needed for some BitComs (502, 510) and for commitments of some input keys (505). In order to let all

Table B.1: Parameter sizes (e.g., for 128 bits of security)

Item	Description
$\boldsymbol{\kappa}$	long-term cryptographic security parameter (e.g., $1^{128} \equiv 128$ bits)
$\boldsymbol{n'}$	size of group-elements used for BitComs (e.g., for $\kappa = 1^{128}$: 3,248 bits for IFC and FFC; 264 bits for ECC)
$\boldsymbol{n}$	size of pre-images of group-elements, a.k.a. scalars, e.g., square-roots or exponents, under the respective one-way function, e.g., squaring or exponentiation (e.g., for $\kappa = 1^{128}$: 3,248 bits for IFC square-roots; 256 bits for FFC and ECC exponents)
$\boldsymbol{\kappa}_{\text{Hash}}$	output size (e.g., 256 bits) of CR-Hash (e.g., SHA-256), needed to ensure κ bits of security (e.g., 128 bits) under birthday attacks
$\boldsymbol{\kappa}_{\text{PRG}}$	size of PRG seed (e.g., 256 bits for AES-256 based PRGen procedure), needed to ensure κ bits of security (e.g., 128 bits) under birthday attacks.
$\boldsymbol{\lvert gg \rvert}$	size of a garbled (multiplicative Boolean) gate (e.g., 256 bits)
$\left(\lvert\mathscr{C}^{(m)}_{\text{Reg}}\rvert, \lvert\mathcal{O}^{(m)}_{\text{Reg}}\rvert\right)$	sizes, when using a regular (i.e., hiding and binding) minicrypt-based Com scheme, of a commitment ($\mathscr{C}$) and opening ($\mathcal{O}$) of an m-bit string (e.g., 256 bits for committing and $256 + m$ for opening, when using a String-Com based on SHA-256 with 256-bit salt; see possible optimization in Remark B.7)
$\left(\lvert\mathscr{C}^{(m)}_{\text{Ext}}\rvert, \lvert\mathcal{O}^{(m)}_{\text{Ext}}\rvert\right)$	sizes, when using an Ext-Com scheme non-malleable with respect to opening, of a commitment ($\mathscr{C}$) and opening ($\mathcal{O}$) of a string with m bits (e.g., $(3,248 + m;\ 256)$ bits for a RSA-OAEP-based scheme, or $(6,496 + m;\ 256)$ bits for an FFC-based ElGamal Com scheme, or $(512 + m;\ 256)$ bits for an ECC-based ElGamal Com scheme, all of which having randomness with a NPRO pre-image that includes the context of the execution)
$\left(\lvert\mathscr{C}^{(m)}_{\text{Equiv}}\rvert, \lvert\mathcal{O}^{(m)}_{\text{Equiv}}\rvert\right)$	when using an Equiv-Com scheme, sizes of a commitment ($\mathscr{C}$) and opening ($\mathcal{O}$) of an m-bit value (e.g., 256 bits of commitment and $3,248 + \min(256, m)$ bits for opening with IFC; same sizes are required by an FFC-based Pedersen Com scheme for DLC; smaller sizes of $(256; 256 + \max(256, m))$ bits are possible with an ECC-based Pedersen Com scheme for DLC)

Note. The following abbreviations will be used: $|\mathscr{C}\mathcal{O}_{..}| \equiv |\mathscr{C}_{..}| + |\mathcal{O}_{..}|$; $|\mathscr{C}_{\text{Equiv}}| \equiv |\mathscr{C}^{(\kappa_{\text{Hash}})}_{\text{Equiv}}|$; $|\mathscr{C}_{\text{Ext}}| \equiv |\mathscr{C}^{(\kappa_{\text{PRG}})}_{\text{Ext}}|$.

PRG procedures be based on the same block-cipher, it is assumed that, in order to prevent collisions, each procedure uses a distinctive identifier (e.g., 3 bits) as part of the plaintext input of the block-cipher, along with remaining indices that serve as a counter for repetition of the same PRGen procedure. In contrast to the above mentioned PRG procedures, there are two whose dependency on the RSC seeds is only implicit. These are: the generation of wire input keys of $\mathrm{P_B}$ (515), which are generated (via $\text{PRGen}_{\text{InKey}}$) using a (pseudo-random) group-element $\nu_{j,i,c}$ as seed; and the generation of wire output keys of $\mathrm{P_B}$ (519), which are generated (via $\text{PRGen}_{\$\text{ForCom}}$) using a (pseudo-random) wire key as seed. In order to explicitly ensure an entropy of 256 bits, the input of the two above-mentioned procedures also includes an auxiliary seed $\lambda^{(auxi)}_j$, itself generated (via $\text{PRGen}_{\text{AuxiSeed}}$) from the respective RSC seed

(500), and being disclosed in the RESPOND stage for evaluation instances. This is specially important for the procedure $\text{PRGen}_{\$\text{ForCom}}$ used to obtain output keys of P_B, because there the wire output key used as seed $k_{j,i}^{[c]}$ has less entropy, namely only 128 bits. This would not be as important for the inner group elements $\nu_{j,i,c}$ used for input keys of P_B, because there the actual group elements are generated from a pre-image with at least 256 bits of randomness. Nonetheless, the auxiliary seed also has the benefit of allowing that for each challenge instance (j) all pseudo-random generations of the two above-mentioned procedures can be computed using a block-cipher with the same key (in this case the auxiliary seed), which in practice may lead to better efficiency of PRG generation (e.g., by avoiding the key-scheduling needed in AES whenever changing the key).

Garbled gates. Each garbled gate requires 256 bits, corresponding to two AES ciphertexts each, based on the recent "two halves make a whole" construction [ZRE15] (row 7).

Reg-Com scheme (row 8). A regular (i.e., hiding and binding) standalone commitment scheme $(\mathscr{C}_{\text{Reg}}, \mathcal{O}_{\text{Reg}})$ used to commit a bit-string with m bits can be instantiated with 256 bits per commitment and 256+m bits per opening (row 8). Based on any regular non-interactive commitment scheme, The underlying randomness can be used as a PRG seed of any non-interactive regular commitment scheme, and the final commitment can be the CR-Hash of the commitment. In practice, if assuming that a concrete cryptographic hash function (e.g., SHA-256) has a pseudo-random output whenever the input is unpredictable, then it is sufficient to compute the commitment as the hash of the value being committed appended with a random 256-bit salt. Then, the randomness is 256 bits, the commitment is 256 bits and the opening is the randomness followed by the committed value.

Remark B.7 (Shorter openings for commitments of input garbled keys of P_A). An optimization is possible to amortize the size of openings of input keys of P_A, from 384 to 128 bits per opening, as explicitly considered in (505–507). Considering that several keys are opened for each evaluation instance, and that all of them are pseudo-random and already have the length of the security parameter (e.g., 128 bits), it is possible to let the randomness be equal for all keys of each instance, and let the inherent unpredictability be the basis of the hiding property of the commitment (no longer in a semantic sense). The commitment is no longer semantically hiding, because after the RSC open phase P_B is able to check if a

given tentative key is correct or not, but it still has 128 bits of security in respect to finding the key, i.e., finding a pre-image. In practice, assuming that a cryptographic hash has a pseudo-random output whenever the input is unpredictable, then each key commitment may be the hash of a pseudo-random bit-string (e.g., 256 bits) concatenated with the committed key. Overall this optimization allows reducing the size of the opening of each key from 384 bits to 128 bits, i.e., being enough to reveal the key, besides the randomness that is common across all keys for that challenge instance. Each commitment (of the complementary key in each wire) still remains with 256 bits, to have collision resistance up to 128 bits against a malicious P_A.

Ext-Com scheme (row 9). An extractable commitment scheme $(\mathscr{C}_{\text{Ext}}, \mathcal{O}_{\text{Ext}})$ can be obtained from a public-key encryption scheme. The commitment corresponds to encrypting a random seed using public-key encryption and then appending the string obtained by the one-time-pad of the value being committed XORed with the of the PRG expansion of the seed. For an IFC instantiation, RSA-OAEP [BR95, Sho01] allows semantically secure encryption of the random 256-bit seed, producing as cipher-text a group element (e.g., with 3,248 bits). Thus, the commitment of a bit-string of size m requires $3,248 + m$ bits in size, and the respective opening requires 512 bits. For a DLC instantiation, a hybrid encryption based on an initial ElGamal encryption of a random seed requires overall 512 bits of randomness, leading to a ciphertext with $6,496 + m$ bits of commitment and 512 bits for opening. An ECC instantiation reduce the size of the commitment to $512 + m$ bits.

Equiv-Com scheme (row 10). The equivocable commitment scheme $(\mathscr{C}_{\text{Equiv}}, \mathcal{O}_{\text{Equiv}})$ used for the RSC technique, not needing to be homomorphic, can be instantiated by the Blum-based Equiv BitString Com and a CR-Hash function. The commitment is a 256-bit CR-Hash of a group element; the opening requires 3,248+256 bits (from a group element and the committed CR-Hash). The same sizes are required by the Pedersen commitment scheme supported on FFC, but shorter sizes of (256; 512) bits can be obtained with ECC (row 10). Besides supporting the RSC technique in the overall S2PC-protocol, Equiv-BitComs are also relevant for simulatable coin-flipping and the transformation of interactive ZKP and ZKPoKs into non-interactive versions.

B.5.2 Communication complexity per component of the protocol

The cost of NIZKPs and NIZKPoKs and random BitCom permutations is either linear in the statistical parameter or in the number of input and output wires, respectively, but not in their product. Conversely, the communication cost of connectors is linear in the number of evaluation instances multiplied by the number of input and output wires.

Table B.2 lists the communication complexity required by the useful NIZKPs and NIZKPoKs, for IFC and DLC instantiations, and refers to their descriptions.

Table B.3 describes the communication complexity per type of connector, with a cost proportional to the number of input and output wires, multiplied by the number of evaluation GCs, but not the number of check circuits.

Clearly, the size of garbled circuits is a substantial cost. As the circuits increase in size (in proportion to their number of input and output wires) the garbled circuits necessarily become the main communication cost contribution.

For IFC and FFC instantiations, communication may be improved in respect to connectors of input bits of P_A by using additively bit-string Coms to commit to bit-string permutations, instead of one BitCom for each bit of the permutation. For FFC instantiations, communication may be improved in respect to connectors of input bits of P_B, by using a mixed OT construction that combines ideas from the 2-out-of-1 and the 1-out-of-2 constructions.

It is worth noticing that the communication associated with connectors of input of P_A (row 9) and connectors of output of P_B (row 11) requires communication of scalars, whereas the connectors of input of P_B (row 12) for an IFC instantiation (based on 2-out-of-1 OT) requires group elements. As a consequence, an instantiation using finite fields over the integers (FFC) will require more communication for input wires of P_B than for other types of connectors. Conversely, the communication when using ECC (with exponents of the same size as the size by which group elements can be represented) is indifferent to the type (group elements vs. exponents) of communicated elements. Interestingly, for the IFC instantiation based on Blum integers, no gain is obtained from the construction (based on 2-out-of-1 OT) requiring communication of pre-images (square-roots) instead of group-elements (squares), because they are all (group-elements) of the same size.

A variation of the protocol could also consider one RSC commitment per instance, instead of a single one, and in that case there would be a very slight difference associated with also

Table B.2: Communication sizes of NIZKPs, NIZKPoKs and coin-flipping

A	B	C	D	E	
Label	Crypto	Description	Reference	Size	1
$\text{NIZKPoK}_{\text{BI-trapdoor}}[n]$	IFC	NIZKPoK of a trapdoor of a Blum Integer of size n (also proves correctness)	§A.2.3 Fig. A.2	$2n + \sigma \times (2n+1) +$ $\sigma \times \left(\kappa_{\text{PRG}} + \lvert\mathscr{C}^{(n)}_{\text{Ext}}\rvert\right)$	2
$\text{NIZKPoK}_{\text{PseudoSqrts}}[n](\ell)$		NIZKPoK of ℓ pseudo square-roots of a vector of ℓ Blum BitComs, modulo a Blum integer of size n	—	(not specified here; not needed in PKI model)	3
$\text{NIZKPoK}_{\text{DL}}[n', n]$	DLC	NIZKPoK of a discrete log in a group with group elements of size n', and pre-images of size n	§A.3.3 Fig. A.5	$n + v \times \kappa_{\text{PRG}} +$ $s + e \times \left(n + \lvert\mathscr{C}^{(n)}_{\text{Ext}}\rvert\right)$	4
$\text{NIZKPoK}_{\text{ElGOpening}}[n', n]$		NIZKPoK of opening of an ElGamal Com (a pair of group elements of size n', with two exponents of size n)	§A.3.4 Fig. A.7	$n + s + v \times \kappa_{\text{PRG}} +$ $e \times \left(2n + \lvert\mathscr{C}^{(2n)}_{\text{Ext}}\rvert\right)$	5
$\text{NIZKP}_{\text{GBI}}[n]$	IFC	NIZKP of correct Blum integer of size n	§A.2.2 Fig. A.1	$\sigma \times (1 + 2n) + 2n$	6
$\text{NIZKP}_{\text{GM-All-0s}}[n](\ell)$		NIZKP that a vector with ℓ GM BitComs, each of size n, only commits to zeros	§A.2.4	$\sigma \times (2n) + n$ (does not depend on ℓ)	7
$\text{NIZKP}_{\text{GEB}}[n', n](\ell)$	DLC	NIZKP of ℓ good ElGamal BitComs, each being a pair of group elements of size n' with exponents of size n	§A.3.2 Fig. A.4	$\ell \times (4n' + 3n) + n$	8
$\text{NIZKP}_{\text{ElG-All-0s}}[n', n](\ell)$ (parallelization of ℓ instances of $\text{NIZKP}_{\text{Pair-with-Same-DL}}$)		NIZKP of ℓ ElGamal Coms of 0, all based on the same pair of generators (each Com is a pair of group elements of size n', with DLs of size n)	§A.3.1 Fig. A.8	$2n' + 3n$ (does not depend on ℓ)	9
$\text{NIZKP}_{\text{SameComBits}}$ $[(n'_1, ..., n'_m), (n_1, ..., n_m)](\ell)$ (e.g., with m equal to 2 or 3)	IFC	NIZKP of same committed bits across m vectors ($k \in [m]$), each with ℓ BitComs each of size n'_k and openings of size n_k, respectively	§A.4.1 Fig. A.9	$e \times (n'_1 + ... + n'_m) +$ $e \times (n_1 + ... + n_m) +$ $s + (v+1) \times \kappa_{\text{PRG}} +$ $e \times (1 + m \times n)$ (does not depend on ℓ)	10
$\text{GMCF1}_{\text{ElGCom0}}[n', n](\ell)$ (Protocol with three phases)	DLC	Generalized coin-flipping (type 1) of a vector of ℓ ElGamal Coms of 0 (P_A learns the ElGamal Coms of 0; P_B learns their openings)	§C.2.3 Fig. C.6	$\ell \times (4n' + n) +$ $\ell \times \lvert\text{NIZKPoK}_{\text{ElGOpening}}[n', n]\rvert$ $+\lvert\text{NIZKP}_{\text{ElG-All-0s}}[n', n](\mathbf{1})\rvert$	11
$\text{Ext-\&-Equiv-Com}[n](\ell)$ (Protocol with two phases)	DLC	Ext-and-Equiv Com and opening of ℓ exponents, each of size n (hidden in ElGamal Coms of size $2n'$)	§C.1.5	$\ell \times (2n') +$ $\ell \times \lvert\text{NIZKPoK}_{\text{ElGOpening}}[n', n]\rvert$ $+n + \lvert\text{NIZKP}_{\text{ElG-All-0s}}[n', n](0)\rvert$	12
$\text{GMCF1}_{\text{GMCom0}}$ (Protocol with three phases)	IFC	Generalized coin-flipping (type 1) of a vector of ℓ GM BitComs of 0 (P_A learns the GM BitComs of 0; P_B learns their openings)	§C.2.4 Fig. C.7	$\lvert\text{Ext-\&-Equiv-Com}[n](\ell)\rvert$ $+ \ell \times n' +$ $\lvert\text{NIZKP}_{\text{GM-All-0s}}[n](\ell)\rvert$	13
$\text{Ext-\&-Equiv-Com}[n](\ell)$ (Protocol with two phases)	IFC	Ext-and-Equiv Com and opening of ℓ group elements, each of size n	§C.1.4 Fig. C.2	$\ell \times n \times (1 + e/t) + \kappa_{\text{Hash}} \times (1+e) +$ $\kappa_{\text{PRG}} \times v + n(1 + 2e) + \sigma$	14

Legend: Table B.1 suggests parameter sizes. σ (number of bits of statistical security); (v, e) (number of evaluation and check instances in a cut-and-choose with RSC technique, selected to satisfy a statistical security goal — see Table 3.2 in §3.1.4, for concrete parameters, e.g., $(v_{\min}, e_{\max})$ equal to (65,64) or (188,32) for σ = 128 for non-interactive ZK sub-protocols). In DLC, each randomness for opening of an Equiv-Com is assumed to be one exponent of size n; in IFC it is one square-root of size n.

sending *check* seeds. The RSC technique could also be applied independently to garbled circuits and connectors. If not using RSC for garbled circuits, then the multiplicative factor

Table B.3: Communication of PKI-based S2PC-with-Coms (per protocol component)

A	B	C	D	E	F	
Instantiation / Intractability				IFC / DQR	DLC / DDH	1
Commun. of evaluation GCs			$P_A \to P_B$	$e \times \lvert gg \rvert \times (\lvert C \rvert + \ell_B)$	$e \times \lvert gg \rvert \times (\lvert C \rvert + \ell_B)$	2
Commun. of NIZKPoK of trapdoor of outer BitCom scheme ($\mathscr{B}_p$) of P_p			$P_p \to P_{\bar{p}}$	$\lvert \text{NIZKPoK}_{\text{BI-trapdoor}}[n', n] \rvert$	$\lvert \text{NIZKPoK}_{\text{DL}}[n', n] \rvert$	3
Commun. of initial outer BitComs of input and auxiliary output of each party			$P_p \to P_{\bar{p}}$	$(\ell_p + \ell'_p) \times n'$ (from GM BitComs)	$(\ell_p + \ell'_p) \times (2n')+$ (from ElGamal BitComs) $\lvert \text{NIZKP}_{\text{GEB}}[n', n](\ell_p + \ell'_p) \rvert$	4
Commun. of NIZKPoKs of openings of outer Coms of P_p			$P_p \to P_{\bar{p}}$	(not needed)	$(\bar{\ell}_p + \bar{\ell}'_p) \times \lvert \text{NIZKPoK}_{\text{ElGOpening}}[n', n] \rvert$	5
Commun. of extra intermediate BitComs and respective NIZKPs	Input of P_A (related to $\mathscr{B}_{\text{ConA}}$)		$P_A \to P_B$	$\ell_A \times n'$ (from Blum BitComs) + $\lvert \text{NIZKP}_{\text{SCB}}[(n', n'), (n, n)](\ell_A + \ell'_A) \rvert$	—	6
	Input of P_B (related to $\mathscr{B}_{\text{OT}}$)		$P_B \to P_A$	$\ell_B \times n$ (from Blum BitComs) + $\lvert \text{NIZKP}_{\text{SCB}}[(n', n'), (n, n)](\ell_B) \rvert$	$\ell_B \times (\mathbf{2}n)$ (from ElGamal BitComs) + $\lvert \text{NIZKP}_{\text{SCB}}[(\mathbf{2}n, \mathbf{2}n), (n', n')](\ell_B) \rvert$	7
	Output of P_B (related to $\mathscr{B}_{\text{FLB}}$)		$P_A \to P_B$	$\ell'_B \times n$ (from Blum BitComs)	— (outer Coms are enough)	8
Commun. of connectors InA	From pre-images		$P_A \to P_B$	$e \times (n + \ell_A + \ell'_A)$ (openings of Gen Blum Coms) (§B.4.1.1, §B.4.1.2)	$e \times (\bar{\ell}_A + \bar{\ell}'_A) \times (n + \ell_A + \ell'_A)$ (openings of ElGamal Coms) (§B.4.1.1)	9
	From GC Keys			$e \times \ell_A \times \lvert \mathscr{CO}_{\text{Reg}}(\text{Key}) \rvert$		10
Commun. of connectors InB	From group-elements and/or pre-images		$P_A \to P_B$	$e \times \ell_B \times (2n)$ (Two multipliers per $j \in J_E, i \in I_B$, complementing the **2-out-of-1 OT**)	$e \times \ell_B \times (2n')$ (One ElGamal encryption per $j \in J_E, i \in I_B$, constituting a **1-out-of-2 OT**)	11
Commun. of connectors OutB	From group-elements and/or pre-images		$P_A \to P_B$	$e \times \ell'_B \times (2n)$ (Two multipliers per $j \in J_E, i \in O_B$)	$e \times \ell'_B \times (2n)$ (Two exponents per $j \in J_E, i \in O_B$)	12
Commun. of cut-and-choose	If interactive		$P_B \to P_A$	$\lvert v \rvert + \lvert e \rvert$		13
	If non-interactive		$P_A \to P_B$	$\lvert \mathscr{CO}_{\text{Equiv}} \rvert$		14
Commun. of RSC seeds			$P_A \to P_B$	$v \times \kappa_{\text{PRG}}$		15
Commun. of permutations to outer BitComs	Adjust outer BitComs of output of P_B		$P_B \to P_A$	$\ell'_{B'} + \lvert \text{NIZKP}_{\text{SCB}}[(n', n'), (n, n)](\ell'_{B'}) \rvert$ (Blum BitComs vs. GM BitComs)	$\ell'_{B'} + \lvert \text{NIZKP}_{\text{SCB}}[(n', n'), (n, n)](\ell'_{B'}) \rvert$ (Pedersen BitComs vs. ElGamal BitComs)	16
	Adjust outer BitComs of output of P_A		$P_B \to P_A$	$\ell'_A +$ $\lvert \text{NIZKP}_{\text{GM-All-0s}}[n'](\ell'_B) \rvert$	$\ell'_A +$ $\lvert \text{GMCF1}_{\text{ElGCom0}}[n', n](\ell'_B) \rvert$	17
	Coin-flipping of random BitCom permutations		$P_B \to P_A$	$\lvert \text{Ext-\&-Equiv-Com}[n](L) \rvert +$ $\lvert \text{NIZKP}_{\text{GM-All-0s}}[n', n](0) \rvert$	$(\bar{\ell}_A + \bar{\ell}'_A) \times (2n') + (\bar{\ell}_B + \bar{\ell}'_B) \times (4n' + n)+$ $\lvert \text{NIZKP}_{\text{ElG-All-0s}}[n', n](0) \rvert +$ $(\bar{\ell}_A + \bar{\ell}'_A + \bar{\ell}_B + \bar{\ell}'_B) \times$ $\lvert \text{NIZKPoK}_{\text{ElGOpening}}[n', n] \rvert$	18
			$P_A \to P_B$	$(\ell_B + \ell'_B) \times n +$ $\lvert \text{NIZKP}_{\text{GM-All-0s}}[n', n](0) \rvert$	$(\bar{\ell}_B + \bar{\ell}'_B) \times n$ $+ \lvert \text{NIZKP}_{\text{ElG-All-0s}}[n', n](0) \rvert$	19

Legend: Commun. (communication); $\ell'_{B'}$ (in row 16 – number of *private* output wires of P_B, i.e., $i \in O_{B'}$); $\bar{\ell}_p$ (= $\lceil \ell_p / n \rceil$, for $p \in \{A, B\}$) and $\bar{\ell}'_p$ (= $\lceil \ell'_p / n \rceil$, for $p \in \{A, B\}$) (in rows 5, 9, 18 and 19 — number of outer-Coms of each type of wire set, after stringize operation). L ($\ell_A + \ell'_A + \ell_B + \ell'_B$ — total number of circuit input and output bits). See Table B.2 for sizes and description of NIZKPs, NIZKPoKs and coin-flipping sub-protocols. See Table 4.1 in Chapter 4 for parameters (v', e', t) used for coin-flipping in cell E18. See Table B.1 for sizes of other parameters (e.g., $n, n', n_2, n'_2, \kappa_{\text{Hash}}, \kappa_{\text{PRG}}, \lvert gg \rvert, \lvert \mathscr{CO}_{\text{Reg}}(\text{Key}) \rvert, \lvert \mathscr{CO}_{\text{Equiv}} \rvert, \lvert \mathscr{C}_{\text{Ext}} \rvert$).

e in row 2 would be replaced by $(v + e)$. In that case the C&C challenge would necessarily have to be simulatable for the case of malicious P^*_B, i.e., the respective simulator would need to be able to induce the outcome of the C&C partition. This induction can either be via the

Table B.4: Communication in S2PC-with-BitComs of AES-128

A	B	C	D	E	F	G	H	I	J	K	
Parametrization	Type of cryptographic instantiation		IFC	FFC	ECC	IFC	FFC	ECC	ECC		1
	Bits of Statistical security (σ)		40			40			**96**	**128**	2
	C&C parameters	Condition on # evaluation instances	$1 \le e \le \sigma/\mathbf{2}$			$1 \le e \le \sigma/\mathbf{5}$			$1 \le e \le \sigma/\mathbf{5}$		3
		$s \equiv$ # GCs	41			123			272	365	4
		$(v_{\min}; v_{\max})$ (# check circuits)	(21; 40)			(115; 122)			(253; 271)	(340; 364)	5
		$(e_{\min}; e_{\max})$ (# evaluation circuits)	(1; **20**)			(1; **8**)			(1; **19**)	(1; **25**)	6
Size of communicated elements	Set initial outer BitComs	NIZKPoK trapdoor (kB)	426	—		426	—		—		7
		Initial outer BitComs (kB)	156	312	25	156	312	25	25		8
		NIZK good BitComs (kB)	—	673	100	—	673	100	100		9
		NIZKoK of outer openings (kB)	—	187	44	—	187	44	44		10
	$\mathscr{B}_{\text{ConA}}$	Extra intermediate BitComs (kB)	52	—		52	—		—		11
		NIZKP of same committed bits (kB)	59	—		59	—		—		12
	$\mathscr{B}_{\text{OT}}$	Extra intermediate BitComs (kB)	52	—		52	—		—		13
		NIZKP of same committed bits (kB)	59	—		59	—		—		14
	$\mathscr{B}_{\text{FLB}}$	Extra intermediate BitComs (kB)	52		4.2	52		4.2	4.2		15
	†C&C of GCs	Connectors InA (kB)	131	124		53	50		118	155	16
		Connectors InB (kB)	2,079		169	831		68	161	205	17
		Connectors OutB (kB)	2,079	164		831	66		156	205	18
		Communicated GCs (kB)	4,434			1,774			4,212	5,542	19
		Communicated RSC seeds and RSC Equiv Com and opening (kB)	1.5	1.1	0.77	4.5	4.2	3.8	8.2	11	20
	Set final outer Coms	Coin-flip of random permutations (kB)	1,069	193	44	923	193	44	44		21
		Adjust output BitComs OutB (kB) (offset bits and NIZKP of same Com Bits)	59	48	11	59	48	11	11		22
†Analysis	Size of final state of P_A (kB)		209	3.3	0.34	209	3.3	0.34	0.34		23
	Size of final state of P_B(kB)		261	3.4	0.39	261	3.4	0.39	0.39		24
	Total communication (kB)		10,709	8,265	5,478	5,332	4,188	**2,189**	4,883	6,353	25
	(Not GCs) / Total		59%	46%	13%	68%	58%	19%	14%	13%	26
	Total commun. if local-PKI and simple S2PC		8,984	7,734	5,012	3,753	3,657	2,081	4,775	6,245	27

Circuit parameters: $|C|_\wedge = 6,800$; $\ell_A = \ell_B = \ell'_B = 128$. †See Remark B.8 on the variable communication of GCs and connectors, depending on the number of evaluation instances. The NIZKPS, NIZKPoKs and coin-flipping use cut-and-choose parameters for 128 bits of statistical security.

NPRO and Equiv-Com technique, where P_A decides the challenges non-interactively, or in an interactive decision of cut-and-choose, e.g., via a coin-flipping.

The communication of RSC seeds is mostly irrelevant.

B.5.3 Comparison of AES-128 and SHA-256

Tables B.4 and B.5 show the communication required for different types of elements.

Table B.5: Communication in S2PC-with-BitComs of SHA-256

			IFC	FFC	ECC	IFC	FFC	ECC	ECC	
Parametrization	Type of cryptographic instantiation									
	Bits of Statistical security (σ)		40			40			**96**	**128**
	C&C param-eters	Condition on # eval instances	$1 \le e \le \sigma/\mathbf{2}$			$1 \le e \le \sigma/\mathbf{5}$			$1 \le e \le \sigma/\mathbf{5}$	
		$s \equiv$ # GCs	41			123			272	365
		$(v_{\min}; v_{\max})$ (# check circuits)	(21; 40)			(115; 122)			(253; 271)	(340; 364)
		$(e_{\min}; e_{\max})$ (# evaluation circuits)	(1; **20**)			(1; **8**)			(1; **19**)	(1; **25**)
Size of communicated elements	Set initial outer BitComs	ZKPoK/NIZKPoK of trapdoors (kB)	426	—		426	—		—	
		Initial outer BitComs (kB)	312	624	51	312	624	51	51	
		NIZK good BitComs (kB)	—	1346	200	—	1346	200	200	
		NIZKoK of outer openings (kB)	—	187	44	—	187	44	44	
	$\mathscr{B}_{\text{ConA}}$	Extra intermediate BitComs (kB)	104	—		104	—		—	
		NIZKP of same committed bits (kB)	59	—		59	—		—	
	$\mathscr{B}_{\text{OT}}$	Extra intermediate BitComs (kB)	104	—		104	—		—	
		NIZKP of same committed bits (kB)	59	—		59	—		—	
	$\mathscr{B}_{\text{FLB}}$	Extra intermediate BitComs (kB)	104		8.5	104		8.5	8.5	
	†C&C of GCs	Connectors InA (kB)	255	247		102	99		235	309
		Connectors InB (kB)	4,157		338	1663		135	321	422
		Connectors OutB (kB)	4,157	328		1,663	131		311	410
		Communicated GCs (kB)	58,292			23,317			55,377	72,865
		Communicated RSC seeds and RSC Equiv Com and opening (kB)	1.5	1.1	0.77	4.5	4.2	3.8	8.2	11
	Set final outer Coms	Coin-flip of random permutations (kB)	1,797	193	44	1,797	193	44	44	
		Adjust output BitComs OutB (kB) (offset bits and NIZKP of same Com Bits)	59	48	11	59	48	11	11	
†Analysis	Size of final state of P_A (kB)		417	3.3	0.36	417	3.3	0.36	0.36	
	Size of final state of P_B(kB)		521	3.4	0.42	521	3.4	0.42	0.42	
	Total communication (kB)		69,887	65,525	59,563	29,773	27,714	**24,044**	56,611	74,375
	(Not GCs) / Total		17%	11%	2.1%	22%	16%	3.0%	2.2%	2.0%
	Total commun. if local-PKI and simple S2PC		67,382	64,890	59,447	27,269	27,079	23,928	56,494	74,259

Circuit parameters: $|C|_\wedge = 90,825$; $\ell_A = \ell_B = \ell'_B = 256$. Notes from Table B.4 also apply.

An initial NIZKPoK is produced in case of an IFC instantiation, to support the simulator in its subsequent extraction of square-roots, but is not necessary for the DLC instantiation (row 7). The communication size for initial outer BitComs is shown in row 8, with BitComs being more expensive for the IFC and FFC instantiations, which use 3,248-bit group elements and with the ElGamal BitComs in DLC instantiations using 2 group elements per BitCom. The ECC instantiation is much shorter, because each group element has only 264 bits.

The DLC instantiations further require a ZKP of correctness of the ElGamal BitComs (row 9). All BitComs are parsed into a BitStringCom, which in IFC is a simple vector of GM BitComs but in DLC is an homomorphic combination into a single ElGamal BitStringCom

per wire type. A ZKPoK of the randomness of each of these BitStringComs is performed to provide the randomness to the simulator (row 10).

The pre-condition of connectors requires the parties to agree on intermediate BitComs. If they are different from the outer BitComs, then they need to be prepared. For example, for the input bits of P_A the bit-string Com optimization can be implemented with generalized Blum Coms (with trapdoor known by P_B). P_A still needs to build one new Blum BitCom per input bit (row 11) and send a NIZKP that they commit to the same bits as her outer GM BitComs (row 12).

In the case of input bits of P_B, the use of a 2-out-of-1 OT technique with IFC also requires new intermediate BitComs for the input bits of P_B (row 13), along with a standalone NIZKoK of openings corresponding to the bits that are committed by the outer BitComs (row 14).

For the output bits of P_B, the forge-and-lose technique requires P_A to send to P_B Equiv-Coms (Blum or Pedersen) associated with each possible output bit (row 15).

Cut-and-choose parameters for sub-protocols. The number of rounds of interactions is reduced by using NIZKPs and NIZKPoKs instead of interactive ZKPs and ZKPoKs. For those sub-protocols the statistical security parameter is instead equal to the computational security parameter, because the prover learns the inherent proof challenge before sending a message to the prover. In the sizes reported in Tables B.4 and Table B.5: the NIZKPs of same committed bits (rows 12, 14 and 22) is performed with 128 bits of statistical security, using $(v_{\min}, e_{\max})$ equal $(188, 32)$ (allowing better communication than $(65, 64)$). In row 21: the coin-flipping for the IFC case is based on an Ext-and-Equiv Com scheme with (v, e, t) equal to $(245, 148, 74)$, achieving 128 bits of statistical security with an expansion rate equal to 2 in the commit phase (see Table 4.1); the coin-flipping for the DLC case uses a specialized protocol (§C.2.3) based on another NIZKP and NIZKPoK.

Cut-and-choose of garbled circuits. To sustain the cut-and-choose of garbled circuits, P_A sends to P_B the elements of connectors, GCs associated with evaluation instances.

- **Connectors InA.** For connectors of input of P_A, the communication includes group elements associated with the permutation bit, and includes commitments and openings of the wire keys (row 16). In respect to group elements, in IFC the technique requires communicating one BitCom opening per wire per evaluation circuit, whereas in DLC it

requires communicating one BitStringCom opening per evaluation circuit (i.e., without repetition across wires). This is thus much shorter for DLC, because it requires fewer openings and each such opening is smaller. However, the communication component associated to regular commitments (hiding and binding) of the wire input keys (for check instances) and openings (for evaluation instances) is the same for both IFC and DLC instantiation.

- **Connectors InB.** For connectors of input of P_B, the communication varies again with the type of cryptographic instantiation, requiring two group elements per wire per circuit row 17. By coincidence, the size is the same for IFC and FFC. In the former the connectors require one BitCom and one opening, whereas in the later it requires one ElGamal BitCom. ECC is smaller because of the smaller group elements.
- **Connectors OutB.** The connector scheme is essentially the same for connectors of the output of P_B, but using different BitCom schemes (Blum for IFC and Pedersen for DLC). The construction requires two pre-images per wire per circuit, and thus the sizes vary accordingly to the size of pre-images (IFC vs. DLC) (row 18).
- **GCs.** Only the garbled circuits selected for evaluation need to be communicated. Their number and size is independent of the cryptographic instantiation of connectors (IFC or DLC) and account for a very significant portion of the overall communication, specially for low values of evaluation instances and high values of circuit size (row 19).
- **RSC elements.** The avoidance of communication of check circuits is made possible by the sending of one RSC PRG seed for each check instance and also by implementing the RSC Equiv-Com and opening it (row 20).

Remark B.8 (On the size of GCs and connectors). The actual communicated sizes of GCs and connectors varies with the number of evaluation instances. The sizes reported in rows 16–20 (for the C&C of garbled circuits), with respective effect on the communication analysis, consider a cut-and-choose partition whose numbers (e, v) of check and evaluation instances is equal to the respective minimum and maximum $(e_{\min}, v_{\max})$ associated with the chosen interval of variation. An execution where the number of evaluation instances is smaller incurs a lower communication overhead.

Set final outer Coms. The final outer commitments are randomized based on random permutations decided via a simulatable coin-flipping. The parties agree on random "randomnesses" for committing 0 for each final outer commitment (DLC) or BitCom (IFC), which can

then be applied as a permutation to the initial outer commitments (row 21). Furthermore, the final BitComs of output of P_B, which initially had just been prepared with BitComs of random bitS, need to be offset to incorporate the actually obtained output bits. P_B transmits to P_A the needed information: the offset bits and a ZKP that they are the correct offset bits (reducible to a ZKP of same committed bits between Blum and GM BitComs, or Pedersen and ElGamal BitComs) (row 22).

Analysis.

- **Final state of each party.** The final state of each honest party then includes the private input and output bits, the private randomnesses, all final outer Coms, the PKI private trapdoor and the PKI public-parameters of both parties (rows 23 and 24). In the case of DLC the size of the final state is even smaller than the communication associated with the initial outer BitComs because the final outer Coms corresponding to a parsing that compresses several BitComs into a single BitStringCom per wire type. In linked executions the parties may also retain information about the outer BitComs and/or some intermediate BitComs, to enable a more efficient linkage.
- **Total communication.** The overall estimated communication is shown in row 25. It does not account with some metadata elements that would be present in actual implementations, e.g., session identifiers, identifiers of the parties, and other communication protocol overheads, which would nonetheless be insignificant in comparison with the overall communication.
- **"(Not GCs)/ GCs."** Row 26 shows the overhead proportion required by non-GCs. While this gives a an intuition about the communication cost inherent to the BitCom approach, it is not good enough on its own. The overhead shown is minimal in the sense that a varying number of evaluation instances in each execution may lead to fewer GCs (and also connectors). Thus, when a particular execution has better (lower) "total size" than shown, the respective overhead proportion may be higher. It is also noteworthy that in a traditional cut-and-choose approach, just the additional GCs account for a significant overhead. In comparison with the forge-and-lose case, in a traditional cut-and-choose the additional GCs would represent a 200% overhead (e.g., 123 vs. 41 GCs) if not using RSC technique, or about 140% (e.g., 49 vs. 20 evaluation GCs) if using RSC. On top of this, there would still be the overhead related with ensuring the consistency of input and output across different GCs, depending on the S2PC protocol.

- **Communication in simpler S2PC.** In a setting where the output BitComs can be ignored and where a local-PKI can be assumed (i.e., decidable by a simulator for the purpose of simulatability), the overall communication requirements are smaller, as represented in row 27. This total is calculated by discounting from row 25 the values of rows 7, 10, 12, 14, 20 and 22, and by considering the values in rows 11 and 13 instead of the contribution of respective initial outer BitComs in row 8.

Pipelining tradeoff. If the parties *pipeline* computation and communication, namely the generation and sending of garbled gates (P_A), or the receiving and evaluation of garbled gates (P_B), the memory required by each party can be lower than the required communication [HEKM11]. When generating a GC, P_A discards each garbled gate as soon as it uses it in the computation of the (compressive) RSC commitment (in the Commit stage), or as soon as it sends it to P_B (in the Respond stage, for evaluation GCs), and discards the keys of each intermediate wire as soon as all related garbled-gates have been generated. (This requires a trivial re-ordering of some steps, e.g., in the Commit stage and/or on how to compute the global-hash, which for simplicity is not shown.) For each *evaluation* GC, P_B discards each garbled gate as soon as it obtains its output wire key, and discards each (intermediate) wire key as soon as all respective garbled-gates have been evaluated. For each *check* GC, P_B simply generates the GC, from the respective random seed, as described for P_A. This technique allows a significant reduction in memory (i.e., the amount of information that needs to be stored simultaneously). As noted in [KSS12], a down-side of implementing pipelining in a C&C protocol is that P_A needs to generate twice each evaluation GC.

Appendix C

Commitments and coin-flipping

This chapter gives details about concrete protocols for simulatable (Ext&Equiv) Com schemes (Section C.1) and coin-flipping (Section C.2).

C.1 Simulatable commitments

§C.1.1 specifies a variation of the interactive Ext&Equiv Com scheme described at higher level in §4.3, with $\text{Ext}_{\text{ExcIfAb}}$ type of extractability, and defined in a hybrid model with access to an ideal Ext Com functionality and an ideal Equiv Com functionality. §C.1.2 makes the security analysis, describing the simulators and showing that they lead the joint distribution of outputs in the ideal world to be indistinguishable to the respective distributions of real world executions. §C.1.3 shows possible adjustments to achieve post and pre verifiable extractability and comments on non-malleability aspect. §C.1.4 adjusts the protocol to a non-interactive instantiation, in the CRS-and-NPRO model. §C.1.5 comments on a specialized protocol for exponents in DLC.

C.1.1 Interactive Ext&Equiv Com scheme

This section gives with succinct notation in Figure C.1 a description of an $\text{Ext}_{\text{ExcIfAb}}$&Equiv scheme, constituting an adjustment of the respective scheme described in Section §4.3, here adjusted to have a single commitment of a (global) hash, instead of one per instance of the cut-and-choose. An adjustment to Ext_{Post}&Equiv can be made by simply augmenting the

underlying Equiv Com of the global hash into an Ext_{Post}&Equiv Com.

Setup parameters. The parties implicitly agree on several common-input parameters: the family $\mathbb{S}$ of committable domains, for simplicity assumed hereafter to be the family of bit-strings, indexed by bit-string length ℓ (639); the security parameters (computational and statistical) (640); the cut-and-choose parameters, with a fixed number of *check* and *evaluation* instances (641), and the IDA threshold t (642), consistent with the statistical security goal (643), defining the number t of fragments needed to decode a string that has been split with a t-out-of-e erasure encoding; the IDA scheme (644), containing a *split* algorithm and a *recover* algorithm, the former one to be used by both parties, the later one only needed for simulation; an authenticator mode (645) (in Figure C.1, the STRICT mode corresponds to the description given in §4.3.1) and other respective parameters (a function specification *Auth*, a respective output length ℓ_a, and the nonce length ℓ_z) (646); a pair of globally known PRG and CR-Hash functions and respective parameters (seed length and hash output), adequate to the cryptographic security parameter (647). There is thus an auxiliary tuple *auxi* that can be defined with the implicit common input (in practice it might be proposed by P_S and accepted by P_R, or agreed upon in a higher level of a larger protocol), containing: security parameters, cut-and-choose parameters, IDA parameters,authenticator parameters, cryptographic primitives and respective parameters (648). For the purpose of the subsequent specification, the auxiliary parameters are considered implicit in the protocol specification.

Abbreviations. Each execution defines a particular execution context *ctx*, composed of a session and sub-session identifiers, and an identifier of the sender and receiver parties (649), to be used as common context of messages communicated between the two parties. Each call to an underlying ideal commitment functionality will use an adjusted message context that will have the same session identifier but an adjusted (and unique) sub-session identifier (650).

Extractable commit phase (P_S commits a string to P_R).

- **Activation.** The sender (P_S) is activated to initialize the protocol, by receiving from its upper environment a message with identifier `commit`, contextualized, and indicating the length ℓ of the string to commit and the respective string m (651).
- **1.a. Commit instances.** P_S selects n random seeds (652) (e.g., 119) and uses $\mathcal{F}_\text{X}$ to commit individually to each of them (653,654). P_S uses the PRG to expand each seed s_j

Setup parameters.

$\mathbb{S} = \left\{\{0,1\}^{\ell} : \ell \in \mathbb{N}\right\}$ (committable domains) (639)

$\kappa \equiv 1^{\kappa}, \sigma \equiv 1^{\sigma}$ (security parameters) (640)

(v, e) (C&C parameters) — let $n \equiv e + v$ (641)

t (IDA threshold) — let $b \equiv e - t + 1$ (642)

(with $((n-b)!e!) / ((e-b)!n!) > 2^{\sigma}$) (643)

$\text{IDA} \equiv (\text{IDA}_{\text{Split}}, \text{IDA}_{\text{Recover}})$ (algorithms) (644)

$\text{AuthMode} \in \{\text{STRICT}, \text{LOOSE}\}$ (645)

$\text{Auth}_{\text{Params}} = (\text{Auth}_{\text{Mode}}, Auth, \ell_a, \ell_z)$ (646)

$\text{Prims} = ((\text{PRG}, \kappa_{\text{PRG}}), (\text{CR-Hash}, \kappa_{\text{Hash}}))$ (647)

$auxi = ((\kappa, \sigma), (v, e, t), \text{IDA}, \text{AuthParams}, \text{Prims})$ (648)

Abbreviations.

$ctx \equiv (sid, cid, \text{P}_\text{S}, \text{P}_\text{R})$ (649)

$ctx_{(\ldots)} \equiv (sid, (cid, \cdots), \text{P}_\text{S}, \text{P}_\text{R})$ (650)

Phase 1. Ext-Commit phase.

$input_{\text{S}} \to \text{P}_\text{S} : (\texttt{commit}, ctx, \ell, m \in \{0,1\}^{\ell})$ (651)

(1.a. Commit instances.)

For $j \in [n]$:

$\text{P}_\text{S} : s_j \leftarrow^{\$} \{0,1\}^{\kappa_{\text{PRG}}}$ (seed) (652)

$\text{P}_\text{S} \to \mathcal{F}_\text{X} : (\texttt{commit}, ctx_{(\text{X},j)}, \kappa_{\text{PRG}}, s_j)$ (653)

$\mathcal{F}_\text{X} \to \text{P}_\text{R} : (\texttt{receipt}, ctx_{(\text{X},j)}, \kappa_{\text{PRG}})$ (654)

$\text{P}_\text{S} : s'_j = \text{PRG}[s_j]([\ell/t] + \ell_a)$ (655)

$\text{P}_\text{S} : h = \text{CR-Hash}\left(\langle s'_j : j \in [n]\rangle\right)$ (global hash) (656)

$\text{P}_\text{S} \to \mathcal{F}_\text{Q} : (\texttt{commit}, ctx_{(\text{Q},0)}, \kappa_{\text{Hash}}, h)$ (657)

$\mathcal{F}_\text{Q} \to \text{P}_\text{R} : (\texttt{receipt}, ctx_{(\text{Q},0)}, \kappa_{\text{Hash}})$ (658)

If $\text{AuthMode} =^{?} \text{STRICT}$, then:

$\text{P}_\text{S} : h' = \text{CR-Hash}(m)$ (message hash) (659)

$\text{P}_\text{S} \to \mathcal{F}_\text{Q} : (\texttt{commit}, ctx_{(\text{Q},1)}, \kappa_{\text{Hash}}, h')$ (660)

$\mathcal{F}_\text{Q} \to \text{P}_\text{R} : (\texttt{receipt}, ctx_{(\text{Q},1)}, \kappa_{\text{Hash}})$ (661)

(1.b. Cut-and-choose.)

$\text{P}_\text{R} : (J_V, J_E) \leftarrow^{\$} \text{PARTITIONS}[v, e](n)$ (662)

$\text{P}_\text{R} : z \leftarrow^{\$} \{0,1\}^{\ell_z}$ (nonce) (663)

$\text{P}_\text{R} \to \text{P}_\text{S} : (\texttt{c\&c-partition}, ctx, (J_V, J_E, z))$ (664)

(1.c. Open check seeds.)

$\text{P}_\text{S} \to \mathcal{F}_\text{X} : (\texttt{open-ask}, ctx_{(\text{X},j)}) : j \in J_V$ (665)

$\mathcal{F}_\text{X} \to \text{P}_\text{R} : (\texttt{open-send}, ctx_{(\text{X},j)}), s_j) : j \in J_V$ (666)

(1.d. Send eval maskings.)

$\text{P}_\text{S} : \langle m'_j : j \in J_E\rangle \leftarrow \text{IDA}_{\text{Split}}[t, J_E](m)$ (667)

$\text{P}_\text{S} : a_j = Auth_z(m'_j) : j \in J_E$ (authenticators) (668)

$\text{P}_\text{S} : t_j = (m'_j \| a_j) \oplus s'_j : j \in J_E$ (maskings) (669)

$\text{P}_\text{S} \to \text{P}_\text{R} : (\texttt{maskings}, ctx, \ell, \langle t_j : j \in J_E\rangle)$ (670)

$\text{P}_\text{R} : \downarrow (\texttt{receipt}, ctx, \ell)$ (671)

Phase 2. Equiv-Open phase.

$input_{\text{S}} \to \text{P}_\text{S} : (\texttt{open-ask}, ctx)$ (672)

(2.a. Reveal message and open global hash.)

$\text{P}_\text{S} \to \text{P}_\text{R} : (\texttt{reveal}, ctx, m)$ (673)

$\text{P}_\text{S} \to \mathcal{F}_\text{Q} : (\texttt{open-ask}, ctx_{(\text{Q},0)})$ (674)

$\mathcal{F}_\text{Q} \to \text{P}_\text{R} : (\texttt{open-send}, ctx_{(\text{Q},0)}, h)$ (675)

If $\text{AuthMode} =^{?} \text{STRICT}$, then:

$\text{P}_\text{S} \to \mathcal{F}_\text{Q} : (\texttt{open-ask}, ctx_{(\text{Q},1)})$ (676)

$\mathcal{F}_\text{Q} \to \text{P}_\text{R} : (\texttt{open-send}, ctx_{(\text{Q},1)}, h')$ (677)

$\text{P}_\text{R} : \text{If CR-Hash}(m) \neq h' \text{ then } \downarrow (\texttt{abort}, ctx)$ (678)

(2.b. Obtain check masks.)

$\text{P}_\text{R} : s'_j = \text{PRG}[s_j]([|m|/t] + \ell_a) : j \in J_V$ (679)

(2.c. Obtain eval masks.)

$\text{P}_\text{R} : \langle m'_j : j \in J_E\rangle \leftarrow \text{IDA}_{\text{Split}}[t, J_E](m)$ (680)

$\text{P}_\text{R} : a_j = Auth_z(m'_j) : j \in J_E$ (authenticators) (681)

$\text{P}_\text{R} : s'_j = t_j \oplus (m'_j \| a_j) : j \in J_E$ (tentative masks) (682)

(2.d. Verify global hash.)

$\text{P}_\text{R} : \text{If CR-Hash}(\langle s'_j : j \in [n]\rangle) \neq h :$ (683)

then $\text{P}_\text{R} \to output_{\text{R}} : (\texttt{abort}, ctx)$ (684)

else $\text{P}_\text{R} \to output_{\text{R}} : (\texttt{open-send}, ctx, m)$ (685)

Figure C.1: **$\text{Ext}_{\text{ExcIfAb}}$&Equiv BitStringCom scheme in $(\mathcal{F}_\text{X}, \mathcal{F}_\text{Q})$-hybrid world.**

into a string s'_j with a *reduced-length* (equal to the target length ℓ divided by the IDA recovery-*threshold* t) extended by the *authenticator-length* ℓ_a (655). P_S calculates the *global hash* h as the CR-hash of the concatenation of all seed-expansions (656). P_S then uses $\mathcal{F}_\text{Q}$ to commit to the hash h (657,658). If in the STRICT mode, P_S also computes the

hash of the string m (659) and then uses $\mathcal{F}_{\text{Q}}$ to commit to said hash (660,661).
The reduced-length mentioned above assumes for simplicity that the IDA split operation produces optimal size fragments. In practice it is possible to use an IDA that produces slightly larger fragments, though not significantly affecting the communication rate. For simplicity this eventual contribution is here ignored.

- **1.b. Cut-and-choose.** P_{R} samples a random cut-and-choose partition (662), with the parametrized number v of *check* instances (e.g., 73) and *eval* instances (e.g., 46), and also samples a random nonce z (663) (of appropriate length ℓ_z) and sends them both to P_{S} (664).
- **1.c. Open check seeds.** P_{S} asks $\mathcal{F}_{\text{X}}$ to *open* to P_{R} the seeds of *check* instances (but not those of *evaluation* instances) (665,666).
- **1.d. Send eval maskings.** P_{S} uses the threshold IDA to *split* the string m being committed into as many fragments m'_j as the number of *evaluation* instances (667), each with a *reduced length*. Then, P_{S} computes the authenticator a_j of each fragment m'_j as an appropriate function $Auth_z$ indexed by the nonce (668); P_{S} then uses the extended mask s'_j to compute the masking t_j of the fragment concatenated with the authenticator (669). Finally, P_{S} sends to P_{R} the maskings associated with all *evaluation* instances (670), and also informing the committed string length ℓ. For simplicity the syntactical checks required by P_{R} are left implicit (e.g., confirming the lengths of all elements in received messages). Finalizing the commit phase, P_{R} outputs a contextualized `receipt` message to her upper environment, also including the length ℓ of the committed string (671). P_{R} memorizes the eval maskings and the check seeds needed for the open phase of the same context *ctx*.

Equivocable open phase (P_{S} opens a string to P_{R}).

- **2.a. Reveal committed string.** Upon being initialized to the open phase (672), P_{S} sends the string m to P_{R} (673). P_{S} *asks* $\mathcal{F}_{\text{Q}}$ to *open* to P_{R} the previously committed *global hash* h (674,675). If using the STRICT authenticator mode, then P_{S} also asks $\mathcal{F}_{\text{Q}}$ to open to P_{R} the hash h' of the string (676,677), and P_{R} verifies that the received hash h' is consistent with the hash of the received string m (678). If not, P_{R} aborts; otherwise it proceeds.
- **2.b. Obtain *check* masks.** P_{R} locally computes the PRG-expansion (with the appropriate length) of each *check* seed (679).
- **2.c. Obtain *evaluation* masks.** P_{R} uses the IDA to obtain the same fragments that an honest P_{S} would (680). For each fragment m'_j, P_{R} computes an authenticator a_j in the same

way that an honest P_S would have, based on the fragment and the nonce (681). Then, P_R concatenates the tentative fragment m'_j and the tentative authenticator a_j, and computes the XOR combination of the resulting string with the extended masking t_j, thus obtaining a tentative extended mask s'_j, supposedly used by P_S to produce the masking (682).

- **2.d. Verify global hash.** Then, P_R verifies that the global hash of all concatenated masks is equal to the global hash h opened by P_S (683). If some verification fails, then P_R aborts (684), otherwise it accepts the string of P_S as a correct *opening* (685).

Complexity of the protocol. Asymptotically as the target length of the strings being committed increases: the open phase of the protocol has communication rate 1; for each fixed value of maximum allowed communication expansion rate in the commit phase, there is a cut-and-choose (n, e) and erasure-code parametrization (e, t) that achieves a respective allowed communication rate e/t, requiring each phase of $\mathcal{F}_Q$ and $\mathcal{F}_X$ to be invoked for short bit-strings only a number of times that is independent of the polynomial target length.

C.1.2 Simulatability analysis

Proving security amounts to show, without rewinding in the simulation, that the new commitment scheme is Ext&Equiv, i.e., the *commit* phase is extractable (Ext) and the *open* phase is equivocable (Equiv). The analysis assumes that the PRG and CR-Hash are cryptographically secure and that the underlying commitments of seeds and hashes are mediated (in a hybrid model) by respective ideal functionalities ($\mathcal{F}_X$, $\mathcal{F}_Q$). The proof of security is accomplished by defining simulators for the cases of each corrupted party and showing that with overwhelming probability in the statistical security parameter they induced a probability distribution of outputs that is computationally indistinguishable (with respect to the computational security parameter) from the one in the ideal world.

Theorem 2 (security of protocol). Assuming a cryptographically secure PRG, CR-Hash and authenticator, the protocol described in Figure C.1 securely emulates, with extractability of type $\text{Ext}_{\text{ExcIfAb}}$ and with error negligible in the statistical security parameter σ (Table 4.1), the ideal functionality $\mathcal{F}_{\text{MCom}}$ (with abort) of *long* bit-string commitments in the ($\mathcal{F}_X$,$\mathcal{F}_Q$)-hybrid model, in the presence of static and computationally active adversaries.

Adjustments to Ext_{Post} and and Ext_{Pre} extractability are discussed in §C.1.3.

C.1.2.1 Extractability — simulatability with corrupted P^*_S.

The extractor-simulator $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ initiates a simulation, with black-box access to the real adversary $\mathcal{A}$, letting it believe that it is in the real world controlling P^*_S.

Simulation of the *commit* phase. Once the protocol starts, $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ (impersonating the honest P_R and also the ideal Ext-Com $\mathcal{F}_\mathrm{X}$ in the simulated execution) extracts the eval seeds (or all seeds) committed by P^*_S (653) and later receives from P^*_S the check seeds and the maskings of authenticated fragments of the string being committed (670). $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ then unmasks each eval masking, using an XOR with the PRG-expansion of the respective extracted eval seed, obtaining from each a respective *tentative* authenticated fragment μ'_j. $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ verifies whether the authentication is correct or not, thus identifying which instances are *good* (i.e., with respect to the seed-based unmasking). (The security of the described authenticator is statistically derived from the properties of a universal hash family.)

If the number of good evaluation fragments is at least t (the recovery threshold) then $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ uses them (any subset of t *good* fragments) as input (along with respective indices) to the IDA recovery algorithm to reconstruct a tentative committed string and accepts it as the assumed extracted string. Otherwise, if there are fewer than t good fragments, then $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ assumes that this is a case of delayed abort by P^*_S, and decides an arbitrary string (e.g., all zeros, or a random string) of appropriate length as the (fake) extracted string. For example, in the trivial case where P^*_S would build all check and evaluation instances as bad, $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ in the ideal world would still commit to an arbitrary valid value, but later in the open phase it would never let the ideal commitment functionality $\mathcal{F}_{\mathrm{MCom}}$ open the value to the honest P_R. Finally, in either of the two above cases (delayed abort or not), $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ (in the role of the ideal $\widehat{\mathrm{P}}^*_\mathrm{S}$) commits in the ideal world to the extracted string, by sending a respective `commit` message to the ideal functionality $\mathcal{F}_{\mathrm{MCom}}$ in the ideal world, thus committing to it.

For the purpose of the $\mathrm{Ext}_{\mathrm{ExcIfAb}}$ property, it is not problematic that $\mathcal{S}$ uses any subset of eval masks labeled as good. Even if for any of those instances the malicious P^*_S has used a difference mask, the authenticator mechanism guarantees that P^*_S will then fail to induce a successful opening, because the respective instance will lead P_R to calculate in the open phase a different mask that will not be correctly authenticated. As described ahead in §C.1.3, the simulation procedure changes in case of intending to achieve $\mathrm{Ext}_{\mathrm{Post}}$, and if using an $\mathrm{Ext}_{\mathrm{Post}}$&Equiv Com for the global hash.

Simulation of the *open* phase. Once $\mathrm{P}_{\mathrm{S}}^{*}$ opens the committed string to P_{R} in the simulated execution (673–666), $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ checks that the opening is successful (685) and that it corresponds to the previously extracted string. If the opening is unsuccessful, e.g., if the global hash verification fails (683), then $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ emulates an abort, leading $\mathcal{F}_{\mathrm{MCom}}$ to send a contextualized `abort` message to the ideal P_{R}. If (with negligible probability) the opening is successful but different from the string previously extracted by $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$, then $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ outputs `Fail` (i.e., in this case the simulation fails). Otherwise, if the opening of the expected string is successful in the simulated execution, then $\mathcal{S}$ sends a respective `open-ask` message to the ideal $\mathcal{F}_{\mathrm{MCom}}$ in the ideal world, thus leading the ideal $\widehat{\mathrm{P}}_{\mathrm{R}}$ to receive said string.

Analysis of the simulation (statistical security). In the commit phase, $\mathcal{S}$ makes a perfect emulation of the abort distribution, since it only aborts early in the ideal world (leading the ideal P_{R} to abort) if and only if $\mathrm{P}_{\mathrm{S}}^{*}$ also leads $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}[\mathrm{P}_{\mathrm{R}}]$ to abort in the simulated execution. Thus, distinguishability (by the environment) between real and ideal world might only happen if $\mathrm{P}_{\mathrm{S}}^{*}$ is able to successfully open a string different from the one $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ has extracted. However, the probability of such event can be made negligible small in the statistical security parameter, for appropriate cut-and-choose and IDA parameters.

Based on the security of the authenticator mechanism, $\mathrm{P}_{\mathrm{S}}^{*}$ cannot lead $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ to believe that, with respect to consistency between committed seed and openable fragment, a *bad* fragment is *good*, except with negligible probability in the length of the nonce. Also, based on the default binding property of all underlying commitments (actually, on the binding of the respective ideal functionalities), $\mathrm{P}_{\mathrm{S}}^{*}$ is not able to equivocate any of the Ext-Com or Equiv-Com. Now, a malicious successful opening by $\mathrm{P}_{\mathrm{S}}^{*}$ requires that all check instances are good, and that the number of bad evaluation instances is (at least $e - t + 1$) such that the number of good evaluation instances is less than the required threshold t of recovery. Examples of parameters for the negligible probability of this event are shown in Table C.1.

Nuances of extractability. The extractability of the described Com scheme is of type $\mathrm{Ext}_{\mathrm{ExcIfAb}}$, as defined in §4.2.2. This is because $\mathcal{S}_{\mathrm{Ext}}^{\mathrm{S}^{*}}$ can with noticeable probability miss detection of a delayed abort, namely always if a malicious $\mathrm{P}_{\mathrm{S}}^{*}$ commits to an incorrect global hash but otherwise builds correct maskings of a string based on the PRG-expansion of the eval seeds. The overall scheme can be transformed into an $\mathrm{Ext}_{\mathrm{Post}}$-and-Equiv Com scheme if the Equiv-Com of the hash is replaced by an $\mathrm{Ext}_{\mathrm{Post}}$-and-Equiv (or $\mathrm{Ext}_{\mathrm{Pre}}$-and-Equiv) Com, and

assuming that the Ext-Coms of seeds are also $\mathrm{Ext}_{\mathrm{Post}}$ (or $\mathrm{Ext}_{\mathrm{Pre}}$). In particular, this is happens if the hybrid model is adjusted from $(\mathcal{F}_{\mathrm{X}},\mathcal{F}_{\mathrm{Q}})$ to $(\mathcal{F}_{\mathrm{X}},\mathcal{F}_{\mathrm{XQ}})$, i.e., allowing $\mathcal{S}$ to impersonate $\mathcal{F}_{\mathrm{XQ}}$ when P_{S} uses it to commit to the global hash, thus allowing $\mathcal{S}$ to extract the global hash during the commit phase. The global hash can then be used to validate in advance whether a tentative extracted string is correct or is in fact a delayed abort. Such simulation is described in more detail in §C.1.3. A further adjustment to allow pre-verifiable extraction is also detailed therein.

C.1.2.2 Equivocability — simulatability with corrupted $\mathrm{P}^*_{\mathrm{R}}$.

The equivocator-simulator $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ initiates a simulation, with black-box access to $\mathcal{A}$, letting it believe that it is in the real world controlling $\mathrm{P}^*_{\mathrm{R}}$.

Simulation of the *commit* phase. In the ideal world, $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ in the role of $\widehat{\mathrm{P}}^*_{\mathrm{R}}$ waits to receive from $\mathcal{F}_{\mathrm{MCom}}$ a receipt of commitment initiated by the ideal $\widehat{\mathrm{P}}_{\mathrm{S}}$. Then, in the role of P_{S} in the simulated execution, $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ plays the whole commit phase to commit an arbitrary string to $\mathrm{P}^*_{\mathrm{R}}$. This involves keeping state about the the Equiv-Com of the global hash of masks (658), and possibly (i.e., in the STRICT authenticator mode) about the Equiv-Com of the hash of the committed string (661), about the cut-and-choose partition and the nonce, and about the maskings of authenticated fragments (670). If $\mathrm{P}^*_{\mathrm{R}}$ aborts at any point before the end of the overall *commit* phase, then $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ emulates an abort, i.e., in the role of $\widehat{\mathrm{P}}^*_{\mathrm{R}}$ in the ideal world sends an `abort` message to $\mathcal{F}_{\mathrm{MCom}}$, thus making it ignore further actions related with this commitment sub-session.

Simulation of the *open* phase. $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ waits in the ideal world to receive from $\mathcal{F}_{\mathrm{MCom}}$ the *opening* of the target string (i.e., the one initially committed by the ideal $\widehat{\mathrm{P}}_{\mathrm{S}}$). Then, $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$, in the role of P_{S} and also in the role of $\mathcal{F}_{\mathrm{Q}}$ in the simulated execution, sends to $\mathrm{P}^*_{\mathrm{R}}$ the target string (673), instead of the previously committed arbitrary string. If in the STRICT mode, then $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ in the role of $\mathcal{F}_{\mathrm{Q}}$ equivocates the opening of the needed hash of the string (677). Then, $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ computes what are the needed *alternative* eval masks s'_j to obtain (via unmasking of the previously sent maskings t_j) the target string received from $\mathcal{F}_{\mathrm{MCom}}$. This is done in the exact same way that P_{R} does, as follows: $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ computes the string fragments (680), then their authenticators (681), and then takes the XOR with the maskings t_j (682) that were transmitted in the commit phase. Finally, $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ computes the global hash (as in (656), but now using the

updated masks), and then impersonates $\mathcal{F}_{\text{Q}}$ and equivocates the opening of said global hash (675). This allows P_{R}^{*} to perform all verifications as if $\mathcal{S}_{\text{Equiv}}^{\text{R}^{*}}$ was in fact an honest P_{S}. Finally, $\mathcal{S}_{\text{Equiv}}^{\text{R}^{*}}$ outputs in the ideal world whatever P_{R}^{*} outputs in the simulated execution (685).

Analysis of the simulation. The only difference between a real protocol execution and the simulated execution is that $\mathcal{S}_{\text{Equiv}}^{\text{R}^{*}}$ commits to an arbitrary string and later equivocates it. However, detection by P_{R}^{*} of equivocation would require differentiating the random masks from seed-expansions, which is contrary to the pseudo-randomness assumption of the PRG. Thus, in case of corrupted P_{R}^{*} the distributions between real and ideal world are computationally indistinguishable.

C.1.3 Achieving post- and pre-verifiable extraction

The main difference between $\text{Ext}_{\text{ExcIfAb}}$ and Ext_{Post} extractability is that in the former the simulator is not guaranteed an overwhelming probability of distinguishing a delayed-abort commitment from an openable commitment. Interestingly, the difference does not affect the emulatability of the ideal two-party commitment functionality in the static corruption model. First, $\mathcal{S}$ extracts with overwhelming probability any correct value that after a commit phase can be opened successfully with noticeable probability. Second, when no valid value can be opened, the revealing of a delayed abort by a malicious P_{S} leads an honest P_{R} to an `abort` message identical to the case of a regular abort in the open phase. Correspondingly, after a delayed abort by P_{S} in a simulated execution, the ideal $\widehat{\text{P}}_{\text{R}}$ in the ideal world will still perform a regular abort, regardless of the value committed by $\mathcal{S}$ in the ideal world. This dissertation does not explore the adaptive corruption model, where a secure commitment scheme would have to satisfy a more stringent notion of simulatability.

Enabling post-verifiable extractability. As already mentioned, the protocol from §C.1.1 (Fig. C.1) can be easily adapted to allow extractability of the post-verifiable type (Ext_{Post}), by replacing the Equiv-Com of the global hash by an Ext_{Post}-and-Equiv Com. Even though this involves using an Ext&Equiv Com for a short string to enable another Ext&Equiv Com for a larger string, it allows increasing the size of the committable domain, i.e., what may be called a *commitment extension*. In other words, an Ext_{Post}-and-Equiv Com of an arbitrarily (polynomially) large string can be obtained from an Ext_{Post}-and-Equiv Com

of a short hash and a *few* Ext_{Post} Coms of short seeds.

In the simulated commit phase, $\mathcal{S}$ extracts the eval seeds s_j (653) and the global hash h (657) (and possibly also the hash h' of the committed string (660)). If any of those Coms (of a seed or of the global hash) is detected as a case of delayed abort, then $\mathcal{S}$ immediately decides that the overall commit phase is in the state of delayed abort. Also in the commit phase, $\mathcal{S}$ receives from P_S^* the nonce z (664) needed to calculate authenticators, the check seeds s'_j (666) and the maskings t_j (670) (of authenticated fragments $(m'_j||a_j)$ of the string m being committed). $\mathcal{S}$ uses the PRG-expansion s'_j of each eval seed to unmask each eval masking t_j and obtain a respective tentative authenticated fragment $\mu'_j||\alpha_j$. $\mathcal{S}$ verifies whether the tentative authenticator α_j is valid; if yes, then the instance is marked as *good* (i.e., with respect to the extracted seed).

If there are fewer than t good fragments, then $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ decides that this is a case of delayed abort by P_S^*. Otherwise, if the number of good evaluation fragments is at least t (the recovery threshold), $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ uses the tentative fragments μ'_j (any subset with t tentative fragments marked as good) as input (along with respective indices) to the IDA recovery algorithm, to calculate a tentative committed string μ. $\mathcal{S}$ uses the tentative string as input to the (deterministic) IDA split algorithm, thus obtaining one new tentative fragment μ''_j for each eval instance. $\mathcal{S}$ then calculates the authenticator α''_j of each IDA-outputted fragment μ''_j (namely for the instances that had not been used to recover the string), and uses the respective authenticated fragments (i.e., concatenation of fragment and authenticator) to calculate the tentative mask σ''_j corresponding to the respective masking t_j. Finally, in possession of all these new eval tentative masks σ''_j, and also using the check masks s'_j obtained as the PRG expansion of the seeds learned in the commit phase, $\mathcal{S}$ concatenates (with an appropriate ordering) all check and tentative eval masks and calculates a respective tentative global hash η. In the STRICT authenticator mode, $\mathcal{S}_{\text{Ext}}^{\text{S}^*}$ also computes the hash η' of the tentative extracted string μ. If the calculated tentative hash(es) is(are both) the same as the extracted hash(es) (i.e., if $h = \mu$ and $h' = \mu'$), then $\mathcal{S}$ accepts the extracted string μ as the correctly committed string m. Otherwise P_S decides that this is a delayed abort, because any successful opening by P_S would require that all eval masks used as pre-image of the global hash are inconsistent with the respective eval seeds.

If $\mathcal{S}$ detected a delayed abort during the described simulation, then it decides an arbitrary string (e.g., all zeros, or a random string) of appropriate length as the (fake) extracted string.

For example, in the trivial case where P_S^* would build all check and evaluation instances as bad, $\mathcal{S}_{\text{Ext}}^{S^*}$ in the ideal world would still commit to a fake value, but later in the open phase it would never let the ideal functionality open the value to the honest P_R.

In summary, the commitment is decided as a delayed abort case if any of the following happens: (i) $\mathcal{S}_{\text{Ext}}^{S^*}$ detects any of the underlying commitments $(\overline{s}_j)$ of seeds or $(\overline{h}, \overline{h}')$ of hash(es) as a delayed abort; (ii) $\mathcal{S}_{\text{Ext}}^{S^*}$ is not able to calculate a valid pre-image $(s'_1||...||s'_n)$ of the extracted global hash h, either because it does not find at least t well authenticated fragments $(\mu'_j||\alpha_j)$ with respect to the extracted seeds, or because the concatenation of eval masks s''_j required to successfully unmask the calculated tentative string μ (recovered from enough well authenticated fragments) and check masks s'_j obtained as PRG expansion of the extracted seeds is not a valid pre-image of the global hash h; (iii) (if in the STRICT authenticator mode), the tentative string μ calculated by $\mathcal{S}_{\text{Ext}}^{S^*}$ is not a pre-image of the respective extracted hash h'.

If $\mathcal{S}_{\text{Ext}}^{S^*}$ decides that P_S has performed a delayed abort, then any successful opening by P_S would either require breaking collision resistance of the CR-Hash (or the binding of commitments), or be lucky to have anticipated the position of enough $(n-t+1)$ evaluation instances, to allow undetected construction of a sufficient number of bad instances to prevent reconstruction of the correct string. The two first conditions are assumed infeasible by cryptographic assumptions; the third condition is statistically prevented (i.e., except with negligible probability) with appropriate cut-and-choose and erasure-code threshold.

Remark C.1 (Authenticators vs. Ext_{Post}&Equiv-Coms of hashes). In the scheme just described, based on an Ext_{Post}&Equiv Com of a global hash (§C.1.3), the authenticators are essential to enable an Ext_{Post} type of extractability, i.e., to ensure detection of delayed abort cases. Conversely, the scheme hinted in Section 4.3, with procedure depicted in Fig. 4.3 and with simulation depicted in Fig. 4.4 (case B), achieves correct Ext_{Post} extraction without using authenticators, and rather using one Ext_{Post}&Equiv-Com (of the hash of each mask) per evaluation instance. Indeed there is a tradeoff: when using one Ext_{Post}&Equiv-Com per mask (to commit its hash), the Com serves as an authenticator.

Enabling pre-verifiable extractability. A post-verifiable extractable (Ext_{Post}) Com scheme can be transformed into a respective pre-verifiable (Ext_{Post}) scheme by defining that

a delayed abort is equal to a conventionalized value in the domain of committable values, e.g., the all-zeros string in the case of a scheme for bit-strings of a given length, thus making $\mathrm{P_R}$ accept said value, instead of aborting, in case $\mathrm{P_S}$ proves that a delayed abort took place. The opening of the conventionalized value via the "delayed abort" can be performed by opening the hash(es) and the eval seeds, thus giving to $\mathrm{P_R}$ the same power as $\mathcal{S}^{\mathrm{S}^*}_{\mathrm{Ext}}$ had during the commit phase, i.e., with respect to deciding whether or not the commitment was a delayed abort. While the described opening procedure is not equivocable, this commitment would never be produced by a simulator $\mathcal{S}^{\mathrm{R}^*}_{\mathrm{Equiv}}$ impersonating an honest $\mathrm{P_S}$, in a simulated execution with access to a possibly malicious $\mathrm{P}^*_{\mathrm{R}}$. Complementary, upon the opening of the conventionalized string, the output of an honest $\mathrm{P_R}$ does not reveal whether the opening was regular or via the delayed abort path, and thus the emulation in the ideal world works well by asking the ideal commitment functionality to simply commit the conventionalized value.

Remark C.2 (Comparison with non-malleability). A notion somewhat similar to post-verifiable extraction can be found in an "a-posteriori verifiable proof of knowledge," e.g., used by Fischlin in a rewinding setting to devise Ext-Coms of sub-linear size, non-malleable with respect to opening (NM-wrt-opening) [FF09] — there, the proof is only validated in the open phase, i.e., allowing an incorrect proof to be given in the commit phase. However, as a possible property of a real commitment scheme (not necessarily simulatable), NM-wrt-opening is incomparable to $\mathrm{Ext_{Post}}$ extractability, in the sense that neither implies the other. For example, if the $\mathrm{Ext_{Post}}$ scheme described in §C.1.3 is instantiated by replacing the underlying ideal Com functionalities (applied to the seeds and the hash) by (i) malleable (e.g., including removing the dependency on the context *ctx*) or (ii) NM-wrt-opening real Com schemes (not necessarily simulatable) — $\mathrm{Ext_{Post}}$-Coms for seeds and $\mathrm{Ext_{Post}}$&Equiv-Coms for hashes — then the overall Com scheme respectively becomes (i) malleable or (ii) NM-wrt-opening. Complementary, an $\mathrm{Ext_{ExcIfAb}}$ (i.e., not $\mathrm{Ext_{Post}}$) Com scheme may also be malleable (obvious) or NM-wrt-opening (e.g., a Com scheme defined as the nested composition of a NM-wrt-opening Com of an $\mathrm{Ext_{ExcIfAb}}$ Com).

In the constructions in this dissertation, it is the inclusion of the execution context *ctx* (which includes the identities of sender and receiver, the session identifier and an unique sub-session identifier), as part of the input of the commit and open phases, that ensures NM-wrt-to opening and/or NM-wrt-commitment, namely preventing the success of replay attacks. For example, a simple replay of a real commitment, by the same sender or a different sender,

would not be accepted by an honest receiver — the ideal underlying functionalities confirm the source and destination of each message and ignore the repetition of execution contexts; the defined real commitment scheme assumes (often implicitly) a way to ensure uniqueness of the execution context, and relates the execution context with the actual commitment value in a non-trivial way (e.g, with the help of a NPRP, as in the construction in §C.1.4).

C.1.4 Non-interactive Ext&Equiv-Com

The non-interactive Com scheme alluded in Remark 4.4 is described in Figure C.2. Interestingly, the use of real non-interactive Com schemes allows avoiding the authenticator mechanism, by taking advantage of the explicit value of commitments, which did not exist when commitments were mediated through ideal commitment functionalities. In the new scheme, which uses one Equiv-Com and one Ext-Com per instance of the cut-and-choose, there is as usual one random seed for each instance, and a respective Ext-Com of the seed, but then (as an innovation in comparison with the protocol in the hybrid model), the Equiv-Com of the respective hash of the mask is produced using the same seed as "randomness." This ensures that for a good instance (i.e., one that is validated if selected for check) the extraction of the seed actually also allows extraction of the Equiv-Com of the hash of the mask, thus providing the needed verifiability that would otherwise be provided by an authenticator mechanism.

Remark C.3 (). **On simultaneous extractability and equivocability.** An interesting feature of this non-interactive instantiation is that the actual commitment sent in the commit phase is simultaneously extractable and equivocable, for a simulator that knows the trapdoor. This is in contrast with other approaches (e.g., mixed commitments [DN02]) where the actual CRS changes (though indistinguishable to a real party) with the type of corruption being simulated, in order to either allow extraction or allow equivocation. The property of a CRS allowing simultaneous extraction and equivocation of the same commitment intuitively appears to be interesting to consider for protocols designed for adaptive security. However, such analysis is outside of the scope of the static model considered in this dissertation, and it is worth recalling that the protocols herein are not proven secure in the adaptive model.

Setup parameters. As common setup input, the parties agree on the family $\mathbb{S}$ of committable domains, hereafter assumed to be the family of sets of bit-strings of indexed length (686),

Setup parameters.

$\mathbb{S} = \left\{\{0,1\}^\ell : \ell \in \mathbb{N}\right\}$ (committable domains) (686)

$\kappa \equiv 1^\kappa, \sigma \equiv 1^\sigma$ (security parameters) (687)

(v, e) (C&C parameters) — let $n \equiv v + e$ (688)

$\Pi \equiv \text{PARTITIONS}[v, e]([n])$ (689)

t (IDA threshold) — let $b = e - t + 1$ (690)

(with $((n-b)!e!) / ((e-b)!n!) > 2^\sigma$) (691)

$\text{IDA} \equiv (\text{IDA}_{\text{Split}}, \text{IDA}_{\text{Recover}})$ (algorithms) (692)

$\text{Prims} = (\text{PRG}, \kappa_{\text{PRG}}), (\text{CR-Hash}, \kappa_{\text{Hash}})$ (693)

CRS (common reference string) (694)

Abbreviations.

$auxi = ((\kappa, \sigma), (v, e, t), \text{IDA}, \text{Prims}, \text{CRS})$ (695)

$ctx = (sid, cid, \text{P}_\text{S}, \text{P}_\text{R})$ (696)

$ctx' = (ctx, auxi)$ (697)

Phase 1. Ext-Commit phase.

$input_\text{S} \to \text{P}_\text{S} : (\texttt{commit}, ctx, auxi, \ell, m \in \{0,1\}^\ell)$ (698)

(1.a. Commit instances.) P_S: For $j \in [n]$:

$s_j \leftarrow^\$ \{0,1\}^{\kappa_{\text{PRG}}}$ (seed) (699)

$\underline{s}_j \leftarrow^\$ \text{Gen}_{\$\text{ForCom}}\left(\mathscr{C}_{\text{Ext}}^{(\text{CRS})}; s_j\right)$ (700)

$\overline{s}_j = \mathscr{C}_{\text{Ext}}^{(\text{CRS})}(s_j;\ \underline{s}_j)$ (Ext-Com of seed) (701)

$s'_j = \text{PRGen}_{\text{BitString}}[s_j][1](\lceil \ell/t \rceil)$ (mask) (702)

$h_j = \text{CR-Hash}\left(ctx', s'_j\right)$ (hash of mask) (703)

$\underline{h}_j = \text{PRGen}_{\$\text{ForCom}}[s_j][2]\left(\mathscr{C}_{\text{Equiv}}^{(\text{CRS})}; h_j\right)$ (704)

$\overline{h}_j = \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(h_j;\ \underline{h}_j)$ (Equiv-Com of hash) (705)

(1.b. Cut-and-choose.) P_S:

$H = \text{CR-Hash}\left(\left\langle(\overline{s}_j, \overline{h}_j) : j \in [n]\right\rangle\right)$ (global hash) (706)

$(J_V, J_E) = \text{NPRO}[\text{CRS}, \overline{H}](\Pi)$ (707)

(1.c. String masking.)

$\text{P}_\text{S} : \left\langle m'_j : j \in J_E\right\rangle \leftarrow \text{IDA}_{\text{Split}}[t, J_E]\,(m)$ (708)

$\text{P}_\text{S} : t_j = m'_j \oplus s'_j : j \in J_E$ (maskings) (709)

$\text{P}_\text{S} : R_V = \left\langle(s_j, \underline{s}_j) : j \in J_V\right\rangle$ (check seeds) (710)

$\text{P}_\text{S} : R_E = \left\langle(\overline{s}_j, \overline{h}_j, t_j) : j \in J_E\right\rangle$ (eval commits) (711)

$\text{P}_\text{S} : \overline{m} \equiv ((J_V, J_E), (R_V, R_E))$ (712)

$\text{P}_\text{S} \to \text{P}_\text{R} : (\texttt{commit}, ctx, \ell, \overline{m})$ (713)

(1.d. Commitment verification.) P_R:

$((J_V, J_E), (R_V, R_E)) \equiv \overline{m}$ (as (712)) (714)

$\left\langle(\overline{s}_j, \overline{h}_j, t_j) : j \in J_E\right\rangle \equiv R_E$ (as (711)) (715)

$\left\langle(s_j, \underline{s}_j) : j \in J_V\right\rangle \equiv R_V$ (as (710)) (716)

Get $(\overline{s}_j, \overline{h}_j) : j \in J_V$ (steps (701–705)) (717)

$H = \text{CR-Hash}\left(\left\langle(\overline{s}_j, \overline{h}_j) : j \in [n]\right\rangle\right)$ (as (706)) (718)

If $\text{NPRO}\,[H]\,(\Pi) \neq (J_V, J_E)$, (719)

then $\downarrow (\texttt{abort}, ctx)$ (720)

else $\downarrow (\texttt{receipt}, ctx, \ell)$ (721)

Phase 2. Equiv-Open phase.

$input_\text{S} \to \text{P}_\text{S} : (\texttt{open-ask}, ctx)$ (722)

(2.a. Reveal message and open hashes.)

$\text{P}_\text{S} \to \text{P}_\text{R} : \left(\texttt{open}, ctx, \left(m, \left\langle\underline{h}_j : j \in J_E\right\rangle\right)\right)$ (723)

(2.b. Verify opening.)

$\text{P}_\text{R} : \left\langle m'_j : j \in J_E\right\rangle \leftarrow \text{IDA}_{\text{Split}}[t, J_E]\,(m)$ (724)

For $j \in J_E$:

$\text{P}_\text{R} : s'_j = t_j \oplus m'_j$ (tentative masks) (725)

$\text{P}_\text{R} : h_j = \text{CR-Hash}\left(ctx', s'_j\right)$ (tentative hashes) (726)

P_R : If $\bigvee_{j \in J_E} \mathscr{C}_{\text{Equiv}}^{(\text{CRS})}(h_j;\ \underline{h}_j) \neq \overline{h}_j$, (see (711,723)) (727)

then $\downarrow (\texttt{abort}, ctx)$ (728)

else $\downarrow (\texttt{open-send}, ctx, m)$ (729)

Figure C.2: **Non-interactive rate-e/t $\text{Ext}_{\text{ExcIfAb}}$&Equiv BitStringCom scheme.**

the computational and statistical security parameters (κ, σ) (687), the cut-and-choose parameters (v, e) (688), which define the set Π of possible cut-and-choose partitions (689), an IDA threshold t (690), which defines the number b of bad instances in an optimal attack, and which overall must be consistent with the statistical security goal (691). The IDA scheme is defined by a *Split* algorithm that splits strings into fragments, and a *Recover* algorithm that allows

decoding (indexed) sets with enough number of correct fragments into the respective original strings (692) (the parties will not need to use the Recover algorithm, but it needs to be defined to allow extractability by a simulator). The needed PRG and CR-Hash primitives are globally defined, along with respective input and output lengths consistent with the computations security parameter κ (693). As a fundamental difference from the interactive protocol described in a ($\mathcal{F}_{\mathrm{X}}$,$\mathcal{F}_{\mathrm{Q}}$)-hybrid setting with ideal commitments (see Figure C.1), the current non-interactive protocol is based on a common reference string (CRS) that defines the parameters of the underlying real non-interactive Ext-Coms and non-interactive Equiv-Coms (694), and allowing respective Ext and Equiv properties to a simulator that is able to decide the CRS. This current CRS-based non-interactive protocol does not need to use the authenticator mechanism.

Abbreviations. The parties are aware of all auxiliary parameters mentioned above, knowing how to parse it deterministically into a tuple *auxi* (695). Each execution is contextualized with a tuple *ctx* that contains the session and subsession identifiers, and the identity of the sender and receiver (696). By definition, the *extended context* is defined as the pair composed of the basic context *ctx* and the tuple *auxi* of auxiliary parameters (697).

Commit phase. The protocol is initialized when the environment activates $\mathrm{P_S}$ with an input tuple composed of a message identifier `commit` for initiating a commitment, an adequate execution context *ctx*, a length ℓ (i.e., the index that defines the committable domain), and a respective bit-string m to be committed (698). (All other required parameters — *auxi* — remain implicit and assumed known by both parties; otherwise they would be transmitted or negotiated in a first step.)

- **1.a. Commit instances.** For each instance $j \in [n]$ of the cut-and-choose, $\mathrm{P_S}$ proceeds as follows: selects a random PRG seed s_j (699), uses it to pseudo-randomly generate "randomness" $\underline{s}_j$ suitable to produce an Ext-Com of the seed (700), and then uses the generated randomnesses to generate an Ext-Com $\overline{s}_j$ of the seed (701). Then, $\mathrm{P_S}$ calculates the pseudo-random expansion s'_j of the seed into a *mask* with a *reduced length* equal to the least integer not less than the string length ℓ divided by the IDA threshold t, i.e., the length necessary to mask a fragment (702). $\mathrm{P_S}$ calculates the CR-Hash h_j of the pair composed of the extended context *ctx'* and the mask s'_j (703). (The use of the extended context in the pre-image induces non-malleability of the overall scheme, i.e., preventing a successful reuse in executions with different contexts.) Then, $\mathrm{P_S}$ uses the initial seed s_j

to **pseudo-randomly** generate "randomness" $\underline{h}_j$ suitable for an Equiv-Com of the hash (704), and uses it indeed to produce an Equiv-Com $\overline{h}_j$ of the hash (705).

As already mentioned when describing the interactive scheme, for simplicity the *length* of a mask is being defined for the case of using a perfectly ... erasure code. In practice the use of an erasure code with a slight expansion in size of fragments is trivially possible, by correspondingly increasing slight the size of masks. Here, "slightly" means an overhead that is not relevant vs. the communication savings brought by the use of fragmentation.

It is crucial that the seed s_j committed by the Ext-Com $\overline{s}_j$ is also the seed used to pseudo-randomly generate the randomness $\underline{h}_j$ used to produce the Equiv-Com $\overline{h}_j$ of the hash h_j, so that the extraction of the seed allows the simulator to reconstruct the Equiv-Com $\overline{h}_j$ of the hash once having a correct guess of the committed hash h_j.

- **1.b. Cut-and-choose.** $\mathrm{P_S}$ computes a CR-Hash H, hereafter denoted global hash, of the vector of pairs of instances, i.e., with each pair being composed of an Ext-Com $\overline{s}_j$ of a seed and an Equiv-Com $\overline{h}_j$ of a hash (706). $\mathrm{P_S}$ then feeds the global hash into the NPRO, obtaining as output a random cut-and-choose partition from the set Π of allowed partitions (707).
- **1.c. String masking.** $\mathrm{P_S}$ uses the IDA Split algorithm to split the committing value m into a set of fragments m_j, one for each evaluation instance, with the agreed recovery threshold t (708). $\mathrm{P_S}$ computes the masking t_j of each fragment m'_j by XORing it with the respective mask s'_j (709). $\mathrm{P_S}$ prepares a "reply" tuple R_V for check instances, as a vector of pairs, with one pair per check instance, and each pair being composed of the seed s_j and the randomness $\underline{s}_j$ used to commit it (710). $\mathrm{P_S}$ also prepares a "reply" tuple R_E for eval commit-related elements, as a sequence of triplets, with each triplet containing the Ext-Com $\overline{s}$ of the seed $\overline{s}_j$, the Equiv-Com $\overline{h}_j$ of the hash and the masking t_j of the authenticated fragment (711). $\mathrm{P_S}$ then prepares the string commitment $\overline{m}$ as the tuple containing the pair (J_V, J_E) of subsets that defines the cut-and-choose partition, and the pair (R_V, R_E) of "reply" tuples (712). Finally, $\mathrm{P_S}$ sends to $\mathrm{P_R}$ a message with identifier `commit`, containing the execution context *ctx*, the string length ℓ and the string commitment $\overline{m}$ (713).
- **1.d. Commitment verification.** $\mathrm{P_R}$ starts by parsing the received commitment $\overline{m}$ into the respective expected elements (714, 715, 716). It is left implicit that the necessary syntactic verifications are performed, and if any one of them fails then $\mathrm{P_R}$ would output an abort (as in (720)). Then, for each check instance ($j \in J_V$), $\mathrm{P_R}$ repeats the initial procedure

of an honest P_S to obtain a pair with an Ext-Com $\overline{s}_j$ of a seed and an Equiv-Com $\overline{h}_j$ (717). P_R joins these elements along with the ones received in the commit message with respect to the eval instances, to reproduce a vector of pairs (i.e., a pair for each instance), and use it as input to recompute the tentative global hash H (718). Then, P_R uses this hash as input to the NPRO, to obtain the cut-and-choose partition from the CRS and the Equiv-Com of the global hash (719). If the partition is not the one in the commitment message then P_R aborts (720); otherwise it accepts the commitment (721).

Open phase. Upon activation to open the commitment (722), P_S sends a contextualized `open-ask` message to P_R, containing the committed string m and a tuple with the randomnesses $\underline{h}_j$ used to produce the Equiv-Coms $\overline{h}_j$ of the hashes h_j of eval instances (723). P_S applies the IDA Split algorithm to obtain from the committed string a tuple of fragments m'_j, one for each eval instance (724). For each eval instance, P_R uses the obtained fragment m'_j to unmask the respective masking t_j and thus obtain a tentative mask s'_j (725), which a honest P_S would have used,and then P_R calculates the CR-Hash h_j of the extended context followed by the tentative mask (726). Finally, using the randomnesses $\underline{h}_j$ received from P_S, P_R recomputes the Equiv-Coms $\overline{h}_j$ of the hashes h_j of masks (727), and verifies that they are equal to the Equiv-Coms received from P_S in the commit phase. If some verification fails, then P_R outputs a contextualized `abort` message (728); otherwise it outputs a contextualized `open-send` message containing the revealed string m (729).

Analysis. The analysis of simulatability is look-alike to the case of the interactive protocol, except for some differences mentioned in this paragraph.

- **Extractability.** Instead of the previous ad-hoc authenticator mechanism, in this protocol the verifiability of each tentative fragment is achieved by checking whether or not the seed s_j extracted from the Ext-Com $\overline{s}_j$ enables recomputing the Equiv-Com $\overline{h}_j$ of the hash h_j. By the binding property of commitment schemes, a correctly verified Equiv-Com $\overline{h}_j$ informs the extractor-simulator ($\mathcal{S}^{S^*}_{Ext}$) that the respective committed hash h_j is the only one that even a malicious P_S may be able to open later in the open phase. Thus, from the maskings t_j of fragments of instances for which $\mathcal{S}^{S^*}_{Ext}$ has a confirmation of the only possible mask that may lead to a successful opening, $\mathcal{S}^{S^*}_{Ext}$ may compute the respective unmasking, thus obtaining tentative fragments. Once $\mathcal{S}^{S^*}_{Ext}$ obtains enough good tentative fragments, i.e., in number equal to the IDA threshold, $\mathcal{S}^{S^*}_{Ext}$ can reconstructs a tentative

string μ, using the IDA Recover algorithm. For the purpose of $\text{Ext}_{\text{ExcIfAb}}$ extractability, this tentative extracted string is enough, because it means that P_S will in the open phase either open such message or abort. The statistical security is related in the same way as before to the cut-and-choose and IDA threshold parameters.

- **Equivocability.** Equivocability is directly related to the ability of equivocating the underlying Equiv-Com of hashes. Interestingly, in this protocol the pre-image of the NPRO can directly be an actual hash, instead of having to use a respective Equiv-Com (as in the NIZKPs and NIZKPoKs in Appendix A). This is because here the NPRO is only used to prove that the commit phase was appropriately generated, and in fact the equivocation power of a simulator ($\mathcal{S}^{\text{R}^*}_{\text{Equiv}}$, impersonating an honest P_S, in a simulation with access to a black-box P_R) does not require building incorrect instances in the commit phase. (Conversely, in NIZKPs and NIZKPoKs the simulator needs the ability to produce "fake" proofs.) To equivocate, $\mathcal{S}^{\text{R}^*}_{\text{Equiv}}$ reveals the intended string being equivocated, then calculates what are the hashes h_j of the masks that would be consistent with such string (even though those masks might not be obtainable as PRG expansion of any seeds), and then sends the "randomnesses" $\underline{h_j}$ needed to equivocate those hashes from the respective hashes.
- **Non-malleability.** Non-malleability with respect to executions with other contexts is ensured by the NPRO use, whose pre-image can be traced back, still during the commit phase, to the particular execution context. Basically, the verification that P_R makes for check instances requires using the extended context *ctx'* as pre-image to several CR-Hashes. Even though the Ext-Coms of seeds do not use the extended context (apart from the CRS that defines the Ext-Com scheme), they are committed and opened in the same message, and so the malleability aspect does not apply.
- **Communication complexity.** The communication of the commit phase is (except message contexts and the length of the specification of the string length): one PRG seed, one randomness of Ext-Com, and one bit, per check instance; one Equiv-Com, one Ext-Com, one fragment, and one bit, per eval instance. The communication of the open phase is (except message contexts) the committed string and one randomness of Equiv-Com per eval instance. In summary, the communication is amortized to

Table 4.1 in §4.3.4 has already shown cut-and-choose and IDA threshold parameters for several goals of statistical security and asymptotic communication rate — these parameters are applicable to the non-interactive instantiation just defined.

Table C.1 complements the analaysis by comparing the parameters for 40 bits of statistical

Table C.1: UC commitment scheme parameters for 40 bits of statistical security

A	B	C	D	E	F	
Maximum allowed expansion rate	**This work**		[GIKW14] (original)	Variations of [GIKW14]		1
	$r = e/t \le r_{\max}$	$r' = n/t \le r_{\max}$	$\delta = t_0/(2n')$ $r = n'/n$	Optimal δ $r = n'/n$	t_0-out-of-n' OT $r = n'/n$	2
$r_{\max} \le 2$	$n = 119$ $v = 73$ $e = 46$ $t = 23$ $r' \approx 5.17$ $r = 2$	$n = 324$ $v = 87$ $e = 237$ $t = 162$ $r' = 2$ $r \approx 1.46$	$n = 826$ $n' = 1652$ $t_0 = 428$ $t_{\text{error}} = \lfloor 399/2 \rfloor$ $\delta = 107/826 \approx 0.1295$ $r = 2$	$n = 577$ $n' = 1154$ $t_0 = 339$ $t_{\text{error}} = \lfloor 239/2 \rfloor$ $\delta \approx 0.2064$ $r = 2$	$n = 352$ $n' = 704$ $t_0 = 186$ $t_{\text{error}} = \lfloor 167/2 \rfloor$ $r = 2$	3
$r_{\max} \le 3/2$	$n = 193$ $v = 121$ $e = 72$ $t = 48$ $r' \approx 4.02$ $r = 1.5$	$n = 822$ $v = 144$ $e = 678$ $t = 548$ $r' = 1.5$ $r \approx 1.237$	$n = 2540$ $n' = 3810$ $t_0 = 650$ $t_{\text{error}} = \lfloor 621/2 \rfloor$ $\delta = 65/762 \approx 0.0853$ $r = 1.5$	$n = 1706$ $n' = 2559$ $t_0 = 481$ $t_{\text{error}} = \lfloor 373/2 \rfloor$ $\delta \approx 0.1379$ $r = 1.5$	$n = 1152$ $n' = 1728$ $t_0 = 296$ $t_{\text{error}} = \lfloor 281/2 \rfloor$ $r = 1.5$	4
$r_{\max} \le 11/10$	$n = 775$ $v = 500$ $e = 275$ $t = 250$ $r' = 3.1$ $r = 1.1$	$n = 12,793$ $v = 598$ $e = 12,195$ $t = 11,630$ $r' = 1.1$ $r \approx 1.0489$	$n = 48,200$ $n' = 53,020$ $t_0 = 2424$ $t_{\text{error}} = \lfloor 2397/2 \rfloor$ $\delta = \frac{303}{13255} \approx 0.0229$ $r = 1.1$	$n = 28,740$ $n' = 31,614$ $t_0 = 1498$ $t_{\text{error}} = \lfloor 1377/2 \rfloor$ $\delta \approx 0.03945$ $r = 1.1$	$n = 23,530$ $n' = 25,883$ $t_0 = 1185$ $t_{\text{error}} = \lfloor 1169/2 \rfloor$ $r = 1.1$	5
$r_{\max} \le 101/100$	$n = 7310$ $v = 4684$ $e = 2626$ $t = 2600$ $r' = 2.81$ $r = 1.01$	$n = 1,125,645$ $v = 5631$ $e = 1,120,014$ $t = 1,114,500$ $r' = 1.01$ $r \approx 1.00495$	$n = 4,474,600$ $n' = 4,519,346$ $t_0 = 22,388$ $t_{\text{error}} = \lfloor 22,359/2 \rfloor$ $\delta = \frac{5597}{2,259,673} \approx 0.00248$ $r = 1.01$	$n = 2,384,200$ $n' = 2,408,042$ $t_0 = 12,166$ $t_{\text{error}} = \lfloor 11,677/2 \rfloor$ $\delta \approx 0.004737$ $r = 1.01$	$n = 2,231,600$ $n' = 2,253,916$ $t_0 = 11,166$ $t_{\text{error}} = \lfloor 11,151/2 \rfloor$ $r = 1.01$	6

Common legend for columns B-F. r (communication expansion rate in the commit phase, relative to the target length, i.e., to the length of the value being committed — it is asymptotic in that it does not account with the base short commitments (columns B-C) or the OT implementation (columns D–F).

Legend for columns B-C ("This work"). r' (overall length of PRG output, divided by the target length (at $\mathrm{P_A}$ — it is smaller at $\mathrm{P_B}$, because $\mathrm{P_B}$ does not evaluate the PRG for *evaluation* instances); also the overall length of CR-Hash input, divided by the target length); n (total number of instances in the cut-and-choose); e (number of evaluation instances = number of fragments); t (recovery threshold = number of fragments necessary to recover message). The parameters were chosen to minimize the total number of instances n, while satisfying the *maximum allowed rate* ($r_{\max}$, identified in column A), as follows: in column B ("$r = e/t \le r_{\max}$"), the communication expansion rate r is limited to $r_{\max}$ (in this case the PRG and the CR-Hash can be applied to bigger lengths — see r'); in column C ($t = \lceil n/r \rceil$), the computation expansion rate r' determined by the length of PRG output and CR-Hash input are limited to $r_{\max}$ (and in this case the overall communication rate r is smaller). After minimizing n, the remaining parameters were chosen to minimize e.

Legend for columns D–F ("[GIKW14]" and variations). n (number of blocks before encoding, i.e., number of symbols in which the target message is partitioned); t_0 (0-info threshold (the original notation was t), i.e., number of blocks whose knowledge does not reveal anything about the original message); t_{error} (error-recovery threshold — the original notation is $\Delta/2$); δ (probability of message passing through the δ-Rabin-OT — the original version uses $t_0 = 2\delta n'$); n' (total number of blocks after encoding, satisfying $n' = t + n + \Delta - 1$). For each value $r = n'/n$, the values of other parameters were chosen to minimize n. In column F, where the equivocator-simulator can always equivocate, statistical security depends only on the probability that a malicious $\mathrm{P_A}$ can guess $t_{\text{error}} + 1$ positions that $\mathrm{P_B}$ will not select in the OT.

security with the respective parameters required by the protocol from [GIKW14]. For example, for a goal of asymptotic communication rate of 1.1 their protocol would require encoding the committing value m into 53,020 blocks, and using an error correcting code capable of correcting more than 1198 semantic errors. The table also describes parameters for optimizations of [GIKW14], namely by using k-out-of-n OT instead of δ-Rabin OT, reducing the number of instances by up to a factor slightly larger than two. In the protocol in this dissertation it would be enough to encode the committing string into 275 blocks, and allowing recovery from 250 blocks using an erasure code. The table also shows in column C the parameters required to reduce the overall length of PRG output and CR-Hash input into a corresponding proportion (r') of the length of the string being committed.

C.1.5 Specialized Ext&Equiv Com of an exponent in DLC

In spite of the above defined Ext&Equiv Com schemes with asymptotically low communication complexity, it is useful to define a specialized scheme for committing exponents in DLC, which may be more practical whenever the number of exponents is very small. In the proof of knowledge and proof of correctness approach, the commit phase is followed by a ZKPoK of the opening, and the open phase corresponds to revealing the committed value followed by a ZKP that it is the value committed by the commitment (e.g., see [Lin03] in a rewinding setting).

More specifically, to commit an exponent: in the commit phase, the sender sends an ElGamal commitment of the exponent and gives a NIZKPoK of the ElGamal opening. in the open phase, the sender reveals the committed exponent and gives a NIZKP that it is the correct value — this can be trivially reduced to a NIZKP of ElGamal Com of 0. A set of NIZKPs of ElGamal Com of 0 based on the same pair of generators can be blended into a single NIZKP of a vector of ElGamal BitComs of 0 (§A.3.5), at the cost of a single NIZKP of ElGamal Com of 0.

C.2 Generalized multi coin-flipping

C.2.1 Ideal generalized multi coin-flipping

Figure C.3 describes a successful protocol flow of generalized coin-flipping in the ideal world.

Initial activation:

$\mathcal{Z} : ctx = (sid, cfid, \mathrm{P}_1, \mathrm{P}_2)$ (730)

$\mathcal{Z} \to \mathrm{P}_1 : (\texttt{gcf-in-1}, ctx, \mathbb{G}, (f_1, f_2))$ (731)

$\mathcal{Z} \to \mathrm{P}_2 : (\texttt{gcf-in-2}, ctx, \mathbb{G}, (f_1, f_2))$ (732)

Phase 1:

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{MCF}} : (\texttt{in-1}, ctx, \mathbb{G}, f_1)$ (733)

$\mathcal{F}_{\mathrm{MCF}} \to \mathrm{P}_2 : (\texttt{req-1}, ctx, \mathbb{G}, f_1)$ (734)

Phase 2:

$\mathrm{P}_2 \to \mathcal{F}_{\mathrm{MCF}} : (\texttt{in-2}, ctx, f_2)$ (735)

$\mathcal{F}_{\mathrm{MCF}} : \gamma \leftarrow^{\$} \mathbb{G}$ (736)

$\mathcal{F}_{\mathrm{MCF}} : res_1 = f_2(\gamma)$ (737)

$\mathcal{F}_{\mathrm{MCF}} \to \mathrm{P}_1 : (\texttt{out-1}, ctx, f_2, res_1)$ (738)

Phase 3:

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{MCF}} : (\texttt{OK}, ctx)$ (739)

$\mathcal{F}_{\mathrm{MCF}} : res_2 = f_1(\gamma)$ (740)

$\mathcal{F}_{\mathrm{MCF}} \to \mathrm{P}_2 : (\texttt{out-2}, ctx, res_2)$ (741)

Final local outputs

$\mathrm{P}_1 \to \mathcal{Z} : (\texttt{gcf-out-1}, ctx, res_1)$ (742)

$\mathrm{P}_2 \to \mathcal{Z} : (\texttt{gcf-out-2}, ctx, res_2)$ (743)

Figure C.3: **Generalized coin-flipping (into-a-well) — successful flow of ideal protocol**. Legend: f_1, f_2 (polynomial size specifications of efficiently computable functions); $\mathbb{S}$ (polynomial size specification of a set, efficiently samplable in a uniform way and with elements representable in size polynomial in the implicit computational security parameter κ.) Example: $\mathbb{S} = \{0,1\}^{\ell}$ for some positive integer $\ell \in poly(\kappa)$; or $\mathbb{S} = \mathbb{Z}_q$, with q being the order of a finite field (i.e., a power of a prime), with $|q| \in poly(\kappa)$, e.g., with f_1 or f_2 being the function that computes an ElGamal Com γ' of 0 using as input a respective randomness γ).

Remark C.4 (On coin-flipping images of trivial functions). If f_1 and f_2 are bijections, i.e., one-to-one from $\mathbb{G}$ to its f-image, and if they can be efficiently inverted, then it is sufficient to perform a regular coin-flipping (using the simple $\mathcal{F}_{\mathrm{MCom}}$ in the traditional template), and then let each party output the image under the function that the other party chose. Specifically, this is applicable to the identity function.

C.2.2 GMCF based on ideal commitments

When the full generality of $\mathcal{F}_{\mathrm{GMCF}}$ is not needed, simpler versions can be considered, with more efficient instantiations. This subsection considers several specialized cases: one ($\mathcal{F}_{\mathrm{GMCF\text{-}1}}$) where only the first function (f_1) is not the trivial identity function; one ($\mathcal{F}_{\mathrm{GMCF\text{-}2}}$) where only the second function (f_2) is not the trivial identity function; one ($\mathcal{F}_{\mathrm{MCF}}$) where both functions ($f_1, f_2$) are the trivial identity function. The next paragraphs show how to emulate these less general versions, in a hybrid model with access to some type of ideal multi Com functionality, and assuming some special structure (e.g., homomorphism) in the underlying set $\mathbb{G}$.

Setup:

$\mathcal{Z} : ctx = (sid, cfid, \mathrm{P}_1, \mathrm{P}_2)$ (744)

$\mathcal{Z} : f_2(\cdot) \equiv \cdot$ (identify function) (745)

$\mathcal{Z} \to \mathrm{P}_1 : (\texttt{cfg-in-1}, ctx, \mathbb{G}, (f_1, f_2))$ (746)

$\mathcal{Z} \to \mathrm{P}_2 : (\texttt{cfg-in-2}, ctx, \mathbb{G}, (f_1, f_2))$ (747)

$\mathrm{P}_1, \mathrm{P}_2 : ctx' = (sid, (\texttt{com}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (748)

Phase 1:

$\mathrm{P}_1 : \gamma^{(1)} \leftarrow^{\$} \mathbb{G}$ (749)

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{GMCom}} : (\texttt{commit}, ctx', \mathbb{G}, \gamma^{(1)})$ (750)

$\mathcal{F}_{\mathrm{GMCom}} \to \mathrm{P}_2 : (\texttt{receipt}, ctx', \mathbb{G})$ (751)

Phase 2:

$\mathrm{P}_2 \to \mathrm{P}_1 : (\texttt{contrib-2}, ctx, \gamma^{(2)} \leftarrow^{\$} \mathbb{G})$ (752)

Phase 3:

$\mathrm{P}_1 : f_1'(\cdot) \equiv f_1(\cdot \oplus \gamma^{(2)})$ (753)

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{GMCom}} : (\texttt{open}, ctx', f_1')$ (754)

$\mathcal{F}_{\mathrm{GMCom}} : \gamma' = f_1'(\gamma^{(1)})$ (755)

$\mathcal{F}_{\mathrm{GMCom}} \to \mathrm{P}_2 : (\texttt{open}, ctx', (f_1', \gamma'))$ (756)

Final local outputs

$\mathrm{P}_1 \to \mathcal{Z} : (\texttt{cf-out-1}, ctx, \gamma \equiv \gamma^{(1)} \oplus \gamma^{(2)})$ (757)

$\mathrm{P}_2 \to \mathcal{Z} : (\texttt{cf-out-2}, ctx, \gamma')$ (758)

Figure C.4: **Hybrid protocol for GMCF-1 ($\mathcal{F}_{\text{GMCF-1}}$).** Legend: f (efficiently-computable function — typically a one-way function, but may also be a trivial function, such as the identity function (see Remark C.4)); $\mathbb{G}$ (specification of a group set, with elements representable in polynomial size in the implicit computational security parameter κ; e.g., $\mathbb{G} = \{0,1\}^{\ell}$ for some positive integer $\ell \in poly(\kappa)$; or $\mathbb{G} = \mathbb{Z}_q$, with q being the order of a finite field (i.e., a power of a prime), with $|q| \in poly(\kappa)$, for example (in a DLC instantiation) with f being the function that computes an ElGamal Com γ' of 0 using as input a respective randomness γ); $\oplus$ (efficient group operation).

Generalized Coin-flipping Type-1 ($\mathcal{F}_{\text{GMCF-1}}$). If only the first party decides a non-identity function f_1 to be applied, and if the underlying set $\mathbb{G}$ has an associated group operation $\oplus$, then the functionality $\mathcal{F}_{\mathrm{GMCF}}$ can be emulated in the $\mathcal{F}_{\mathrm{GMCom}}$-hybrid model with access to an ideal generalized multi-com functionality $\mathcal{F}_{\mathrm{GMCom}}$. Figure C.4 describes a protocol in the respective hybrid world.

An alternative to the generalized MCom is possible, as follows. In phase 1, P_1 would start by committing, using an ideal simple MCom $\mathcal{F}_{\mathrm{MCom}}$, to the f_2-image $\gamma'^{(1)}$ of a random element $\gamma^{(1)}$, and then also commit to a respective NIZKPoK of the f_2-pre-image element $\gamma^{(1)}$ of the committed element $\gamma'^{(1)}$. This would enable P_2 to extract the pre-image of the contribution of P_1, if the NIZKPoK is correct and associated with the committed element. Then, in phase 3, P_1 would directly open the f_2-image $\gamma'^{(1)}$ and the respective NIZKPoK. If P_2 would find something wrong with the NIZKPoK (which the simulator would have also found during extraction in the phase 1), then it would not accept the opening and would abort. In a simulated execution, the simulator would have to wait for phase 3 to also abort, instead of aborting directly in phase 1.

Setup:

$\mathcal{Z} : ctx = (sid, cfid, \mathrm{P}_1, \mathrm{P}_2)$ (759)

$\mathcal{Z} : f_1(\cdot) \equiv \cdot$ (identify function) (760)

$\mathcal{Z} : f_2$ is a group homomorphism, i.e., satisfies $f_2(\cdot \oplus \cdot\cdot) = f_2(\cdot) \otimes f_2(\cdot\cdot)$ (761)

$\mathcal{Z} \to \mathrm{P}_1 : (\texttt{cfg-in-1}, ctx, \mathbb{G}, (f_1, f_2))$ (762)

$\mathcal{Z} \to \mathrm{P}_2 : (\texttt{cfg-in-2}, ctx, \mathbb{G}, (f_1, f_2))$ (763)

$\mathrm{P}_1, \mathrm{P}_2 : ctx' = (sid, (\texttt{com}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (764)

$\mathrm{P}_1, \mathrm{P}_2 : ctx'' = (sid, (\texttt{zkpok}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (765)

Phase 1:

$\mathrm{P}_1 : \gamma^{(1)} \leftarrow^{\$} \mathbb{G}$ (766)

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{MCom}} : \left(\texttt{commit}, ctx', \mathbb{G}, \gamma^{(1)}\right)$ (767)

$\mathcal{F}_{\mathrm{MCom}} \to \mathrm{P}_2 : (\texttt{receipt}, ctx', \mathbb{G})$ (768)

Phase 2:

$\mathrm{P}_2 : \gamma^{(2)} \leftarrow^{\$} \mathbb{G}$ (769)

$\mathrm{P}_2 \to \mathrm{P}_1 : \left(\texttt{contrib-2}, ctx, \gamma'^{(2)} = f_2\left(\gamma^{(2)}\right)\right)$ (770)

$\mathrm{P}_2 \to \mathcal{F}_{\mathrm{ZKPoK}}^{\text{Func-Inv}} : \left(\text{send}, ctx'', f_2, \gamma^{(2)}\right)$ (771)

$\mathcal{F}_{\mathrm{ZKPoK}}^{\text{Func-Inv}} \to \mathrm{P}_1 : \left(\text{OK}, ctx'', f_2, \gamma'^{(2)} = f_2\left(\gamma^{(2)}\right)\right)$ (772)

Phase 3:

$\mathrm{P}_1 : \gamma' = f_2\left(\gamma^{(1)}\right) \otimes \gamma'^{(2)}$ (773)

$\mathrm{P}_1 \to \mathcal{F}_{\mathrm{MCom}} : (\texttt{open}, ctx')$ (774)

$\mathcal{F}_{\mathrm{MCom}} \to \mathrm{P}_2 : \left(\texttt{open}, ctx', \gamma^{(1)}\right)$ (775)

$\mathrm{P}_2 : \gamma = \gamma^{(1)} \oplus \gamma^{(2)}$ (776)

Final local outputs

$\mathrm{P}_1 \to \mathcal{Z} : \left(\texttt{cf-out-1}, ctx, \gamma'\right)$ (777)

$\mathrm{P}_2 \to \mathcal{Z} : (\texttt{cf-out-2}, ctx, \gamma)$ (778)

Figure C.5: **Hybrid protocol for GMCF-2 ($\mathcal{F}_{\mathbf{GMCF\text{-}2}}$).** Notes in Figure C.4 also apply.

Generalized Coin-flipping Type-2 ($\mathcal{F}_{\mathbf{GMCF\text{-}2}}$). If only the second party decides a non-identity function f_2 to be applied, and if the function is a group homomorphism from the underlying group $(\mathbb{G}, \oplus)$ into a new group $(f_2(\mathbb{G}), \otimes)$, then the functionality $\mathcal{F}_{\mathrm{GMCF}}$ can be emulated in the $(\mathcal{F}_{\mathrm{MCom}}, \mathcal{F}_{\mathrm{ZKPoK}}^{\text{Func-Inv}})$-hybrid model, with access to an ideal functionality $\mathcal{F}_{\mathrm{MCom}}$ simple multi-Com, and an ideal functionality for ZKPoK of a function inverse $\mathcal{F}_{\mathrm{ZKPoK}}^{\text{Func-Inv}}$. Figure C.5 describes a protocol in the respective hybrid world.

C.2.3 GMCF-1 (in DLC) of an ElGamal Com of 0

This subsection describes a concrete instantiation of GMCF type 1, for the specific function of ElGamal Com of 0 from a respective randomness. The protocol takes advantage of the ElGamal Com structure, using it to commit to part of its own function image. In phase 1, the commit phase of $\mathcal{F}_{\mathrm{GMCom}}$ is equivalent to the commit phase of a simple $\mathcal{F}_{\mathrm{MCom}}$, which can be instantiated as a simple commitment followed by a ZKPoK of the committed element. In phase 3, the open phase of $\mathcal{F}_{\mathrm{GMCom}}$ is obtained with the revealing of the final value (the image obtained from applying the ElGamal-Com-of-0 function to the previously committed randomness) followed by a NIZKP that it is consistent with the function and the commitment. Based on the structure of ElGamal, this can be reduced to just two NIZKPs of same DL, i.e., twice as expen-

Setup Parameters:

Generators: g, h (779)

Order: $q = \#(\langle g \rangle) = \#(\langle h \rangle)$ (780)

$f_1(\cdot) = \langle g^{\cdot}, h^{\cdot} \rangle$ (ElGamal Com of 0) (781)

$f_2(\cdot) = \cdot$ (Identity function) (782)

Initial activation:

$\mathcal{Z} : ctx = (sid, cfid, \mathrm{P}_1, \mathrm{P}_2)$ (783)

$\mathcal{Z} \to \mathrm{P}_1 : (\texttt{cfg-in-1}, ctx, \mathbb{Z}_q, (f_1, f_2))$ (784)

$\mathcal{Z} \to \mathrm{P}_2 : (\texttt{cfg-in-2}, ctx, \mathbb{Z}_q, (f_1, f_2))$ (785)

Phase 1:

$\mathrm{P}_1 : \gamma^{(1)} \leftarrow^{\$} \mathbb{Z}_q$ (786)

$\mathrm{P}_1 : \gamma'^{(1)} = f_1\left(\gamma^{(1)}\right) = \left(g^{\gamma^{(1)}}, h^{\gamma^{(1)}}\right)$ (787)

$\mathrm{P}_1 : r \leftarrow^{\$} \mathbb{Z}_q$ (788)

$\mathrm{P}_1 : c = (c_1, c_2) \equiv \left(g^r, g^{\gamma^{(1)}} h^r\right)$ (789)

$\mathrm{P}_1 : z_1 = \mathrm{NIZKPoK}^{\mathrm{ElGCom}[g,h]}_{\mathrm{Opening}}(c)$ (790)

$\mathrm{P}_1 \to \mathrm{P}_2 : (\texttt{msg-1}, ctx, (c, z_1))$ (791)

$\mathrm{P}_2 : \mathrm{Ver}^{\mathrm{Opening}}_{\mathrm{NIZKPoK}}[\mathrm{ElGCom}[g,h]][c](z_1)$ (792)

Phase 2:

$\mathrm{P}_2 : \gamma^{(2)} \leftarrow^{\$} \mathbb{Z}_q$ (793)

$\mathrm{P}_2 \to \mathrm{P}_1 : \left(\texttt{msg-2}, ctx, \gamma^{(2)}\right)$ (794)

$\mathrm{P}_1 : \gamma = \gamma^{(2)} + \gamma^{(1)} (\mathrm{mod}\ q)$ (795)

Phase 3:

$\mathrm{P}_1 : C_1 = \left(c_1, c_2 / g^{\gamma^{(1)}}\right)$ (796)

$\mathrm{P}_1 : C_2 = ((g^{\gamma}, h^{\gamma}))$ (797)

$\mathrm{P}_1 : z_2 = \mathrm{NIZKP}^{[g,h]}_{\mathrm{VecElGCom0s}}((C_1, C_2))$ (798)

$\mathrm{P}_1 \to \mathrm{P}_2 : \left(\texttt{msg-3}, ctx, \left(\gamma'^{(1)}, z_2\right)\right)$ (799)

$\mathrm{P}_2 : \gamma'^{(2)} = f_1\left(\gamma^{(2)}\right) = \left(g^{\gamma^{(2)}}, h^{\gamma^{(2)}}\right)$ (800)

$\mathrm{P}_2 : \gamma' = \gamma'^{(2)} * \gamma'^{(1)}$ (801)

$\mathrm{P}_2 : C_1 \equiv \left(c_1, c_2 / g^{\gamma^{(1)}}\right)$ (802)

$\mathrm{P}_2 : C_2 \equiv \gamma'$ (803)

$\mathrm{P}_2 : \mathrm{Ver}^{\mathrm{VecElGCom0s}}_{\mathrm{NIZKP}}[g,h][(C_1, C_2)](z_2)$ (804)

Final outputs:

$\mathrm{P}_1 \to \mathcal{Z} : (\texttt{cfg-out-1}, ctx, \gamma)$ (805)

$\mathrm{P}_2 \to \mathcal{Z} : \left(\texttt{cfg-out-1}, ctx, \gamma'\right)$ (806)

Figure C.6: **Generalized coin-flipping type-1 of ElGamal Com of 0**

sive as the NIZKP that would be required to equivocably open an actual ElGamal Com of 0.

Procedure. The procedure is described with succinct notation in Figure C.6.

- **Setup parameters.** The setup considers two generators (g_0, g_1) known by both parties (779), each defining the same group of known order q (780). The first function of the generalized coin-flipping is the ElGamal function f_1 for committing to bit zero, using as bases the two generators. The function image is a pair of exponentiations of the input (the "randomness"), each with the respective generator as base (781). The second function of the generalized coin-flipping is set to be the identity function (782).
- **Initial activation.** For the initial activation of an execution, the environment sets the session and subsession identifiers, as well as the involved parties (783). Then, the environment activates the two parties, which party is in the role of first and second to

receive the output (784, 785).

- **Phase 1.** The first party (P_1) selects a random exponent $\gamma^{(B)}$ (786) and computes its image under the first function, i.e., it computes a respective ElGamal BitCom of 0 (787). Then, P_1 generates a new random exponent r, (788), and uses it as randomness to produce an ElGamal Com c (hereinafter denoted the second commitment) of the first randomness $\gamma^{(B)}$ used to produce the BitCom of 0. (789). Then, P_1 produces a NIZKPoK of the ElGamal opening of the second commitment (790), and sends to the second party (P_2) a contextualized message with the second commitment c and the respective NIZKPoK (791). P_2 verifies the correctness of the NIZKPoK (792).
- **Phase 2.** The second party selects a random exponent $\gamma^{(A)}$ (793), and sends it in a contextualized message to P_1 (794). At this point, P_1 is able to compute the combination γ of the contributions of the two parties (this will later be the output of P_1). (795)
- **Phase 3.** Before revealing the contribution of P_1 to the coin-flipping (a BitCom $\gamma'^{(1)}$ of 0, P_1 elaborates the needed NIZKP, namely that it is consistently related to the randomness to which P_1 was bound in phase 1 and in respect to which it proved knowledge. Conceptually, this can be split in two assertions, one for each component of the ElGamal BitCom $\gamma'^{(1)}$ of 0 to be revealed.

  One assertion to prove is that the initial ElGamal Com c revealed in the first phase was an encryption of the first component of the BitCom to be revealed. This can be reduced to a proof of ElGamal BitCom of 0 after dividing the second component c_2 of the initial commitment by the first component of the BitCom to reveal (796). The second assertion to prove is that the ElGamal BitCom to reveal is indeed an ElGamal BitCom of 0, i.e., that both components have the same discrete log, in respect to the respective base generators. Equivalently, this second assertion can be performed directly in respect to the final ElGamal BitCom of the coin-flipping (i.e., after combining the two contributions) (797). P_1 elaborates a NIZKP transcript of the two assertions, i.e., using parallelization to obtain one transcript as large as if it was just a single assertion (798).

  P_1 sends a contextualized message with her contribution $\gamma'^{(1)}$ — a BitCom of 0 — and with the NIZKP transcript (799). P_2 then computes the result of the coin-flipping, first by calculating the BitCom $\gamma'^{(2)}$ associated with his own contribution $\gamma^{(2)}$ (800), then combining the result with the contribution $\gamma^{(1)}$ of P_1 (801), then calculating the elements relative to the received NIZKP, namely two ElGamal BitComs (802,803) and then verifying the respective NIZKP (804). If the verification returns `false`, then P_2 outputs `abort`.

- **Final outputs.** As final output, P_1 outputs the combined randomness γ (805), whereas P_2 outputs the respectively combined BitCom of 0 γ'. (806).

Complexity. The overall communication complexity of the three messages is four group elements (from two ElGamal Coms), one exponent, one NIZKPoK of ElGamal Opening and one NIZKP of a vector (of length two) of ElGamal Coms of 0 (i.e., of pairs with the same DL in respect to the pair of generators). Actually, the NIZKPs of same DL can be parallelized and cost the same as just one.

C.2.4 GMCF-1 (in IFC) of a vector of GM BitComs of 0

This subsection describes a concrete instantiation of GMCF type 1, for the specific function of GM BitCom of 0 from a respective randomness. Instead of taking advantage of a concrete structure of GM BitComs for the initial Commitments, the protocol follows a different approach. The initial commitment is directly made for the image that needs to be later opened. However, in general this would not allow extraction of the pre-image, e.g., if the simulator did not possess the GM trapdoor. Thus, P_1 also commits to a NIZKPoK transcript of the GM openings (i.e., of their pseudo-square-roots). Later, in the open phase it is enough to open the GM BitComs and prove that they are BitComs of 0. Something that can go wrong here is that in the open stage (phase 3) the receiver P_2 may find that the ZKPoK was incorrect. This means that a simulator would have actually not been able to extract the pre-images of the contribution of P_1, but could not abort in time. Nonetheless, once in phase 3 detecting an incorrect opening, it is defined that P_2 aborts in the real world. Thus, in the simulated execution the simulator in the ideal world sends an abort message to the TTP, which will induce $\widehat{P}_2$ to also receive an abort form the TTP and then output abort to $\mathcal{Z}$.

Procedure. The procedure is described with succinct notation in Figure C.7.

- **Setup.** The protocol goal is in respect to a Blum integer modulus N (807), namely to allow the seconds party P_2 to learn GM BitComs and the first party to learn theur openings (pseudo-square-roots). In the notation of generalized coin-flipping, the first function is thus defined as modular squaring (808), whereas the second function is the identity (809).
- **Initial activation.** The initial activation is similar to the case of DLC, i.e., $\mathcal{Z}$ prepares the execution context (810) and then activates each party with the context, the information

Setup Parameters:

Blum integer modulus: N (807)

$f_1(\cdot) = \cdot^2 \pmod N$ (vectorizable) (808)

Vector of BitComs of 0

$f_2(\cdot) = \cdot$ (Identity function) (809)

Initial activation:

$\mathcal{Z} : ctx = (sid, cfid, \mathrm{P}_1, \mathrm{P}_2)$ (810)

$\mathcal{Z} \to \mathrm{P}_1 : (\texttt{cfg-in-1}, ctx, \mathbb{Z}_N^*, (f_1, f_2))$ (811)

$\mathcal{Z} \to \mathrm{P}_2 : (\texttt{cfg-in-2}, ctx, \mathbb{Z}_N^*, (f_1, f_2))$ (812)

$\mathrm{P}_1, \mathrm{P}_2 : ctx' = (sid, (\texttt{z1}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (813)

$\mathrm{P}_1, \mathrm{P}_2 : ctx'' = (sid, (\texttt{com}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (814)

$\mathrm{P}_1, \mathrm{P}_2 : ctx''' = (sid, (\texttt{z2}, cfid), \mathrm{P}_1, \mathrm{P}_2)$ (815)

Phase 1:

$\mathrm{P}_1 : \gamma_{[\ell]}^{(1)} \leftarrow^{\$} (\mathbb{Z}_N^*)^\ell$ (816)

$\mathrm{P}_1 : {\gamma'_{[\ell]}}^{(1)} = f_1\left(\gamma_{[\ell]}^{(1)}\right) = \left(\gamma_{[\ell]}^{(1)}\right)^2 \pmod N$ (817)

$\mathrm{P}_1 : \gamma'^{(1)}_{[\ell]} = \left\langle \gamma'^{(1)}_i : i \in [\ell] \right\rangle$ (818)

$\mathrm{P}_1 : z_1 = \mathrm{NIZKPoK}^{(ctx')}_{\mathrm{Sqrts}(N)}\left(\gamma'^{(1)}_{[\ell]}\right)$ (819)

$\mathrm{P}_1 : \underline{c} = \mathrm{Gen}_{\$\mathrm{ForCom}}\left[\mathscr{C}_{\mathrm{Ext\&Equiv}}\right]\left(\gamma'^{(1)}_{[\ell]}, z_1\right)$ (820)

$\mathrm{P}_1 : \overline{c} = \mathscr{C}^{(ctx'',\mathrm{CRS})}_{\mathrm{Ext\&Equiv}}\left(\left(\gamma'^{(1)}_{[\ell]}, z_1\right); \underline{c}\right)$ (821)

$\mathrm{P}_1 \to \mathrm{P}_2 : (\texttt{msg-1}, ctx, \overline{c})$ (822)

Phase 2:

$\mathrm{P}_2 : \gamma_{[\ell]}^{(2)} \leftarrow^{\$} (\mathbb{Z}_N^*)^\ell$ (823)

$\mathrm{P}_2 \to \mathrm{P}_1 : \left(\texttt{msg-2}, ctx, \gamma_{[\ell]}^{(2)}\right)$ (824)

$\mathrm{P}_1 : \gamma_{[\ell]} = \gamma_{[\ell]}^{(2)} * \gamma_{[\ell]}^{(1)} \pmod N$ (825)

Phase 3:

$\mathrm{P}_1 : z_2 = \mathrm{NIZKP}^{(ctx'',\mathrm{CRS})}_{\mathrm{Squares}}(N)\left(\gamma'^{(1)}_{[\ell]}\right)$ (826)

$\mathrm{P}_1 : \underline{c} = \mathcal{O}^{ctx'',\mathrm{CRS}}_{\mathrm{Ext\&Equiv}}(c, \underline{c}, \overline{c})$ (827)

$\mathrm{P}_1 \to \mathrm{P}_2 : \left(\texttt{msg-3}, ctx, \left(\gamma'^{(1)}_{[\ell]}, z_1, \underline{c}, z_2\right)\right)$ (828)

$\mathrm{P}_2 : \mathrm{Ver}^{\mathrm{Sqrts}(N)}_{\mathrm{NIZKPoK}(ctx'',\mathrm{CRS})}\left[\gamma'^{(1)}_{[\ell]}\right](z_1)$ (829)

$\mathrm{P}_2 : \mathrm{Ver}^{\mathrm{Squares}(N)}_{\mathrm{NIZKP}(ctx''',\mathrm{CRS})}\left[\gamma'^{(1)}_{[\ell]}\right](z_2)$ (830)

$\mathrm{P}_2 : \gamma'^{(2)}_{[\ell]} = f_1\left(\gamma^{(2)}_{[\ell]}\right) = \left(\gamma^{(2)}_{[\ell]}\right)^2 \pmod N$ (831)

$\mathrm{P}_2 : \gamma'_{[\ell]} = \gamma'^{(2)}_{[\ell]} * \gamma'^{(1)}_{[\ell]}$ (832)

Final outputs:

$\mathrm{P}_1 \to \mathcal{Z} : \left(\texttt{cfg-out-1}, ctx, \gamma_{[\ell]}\right)$ (833)

$\mathrm{P}_2 \to \mathcal{Z} : \left(\texttt{cfg-out-1}, ctx, \gamma'_{[\ell]}\right)$ (834)

Figure C.7: **Generalized coin-flipping type-1 of squares (GM BitComs of 0)**

about who learns the output first and who learns it in the second place (811,812). Upon receiving the context, each party knows the adjustment in needs to perform to use a slightly different (and unique) context in subsequent sub-routines, namely one for a NIZKPoK (813), another for an Ext&Equiv Com (814), and another for a NIZKP (815).

- **Phase 1.** For each index $i \in [len]$, P_1 selects a random residue (816) and then computes its square, i.e., a respective BitCom of zero (817). Then, P_1 aggregates all squares into a vector $\gamma_{[\ell]}^{(1)}$ (818). P_1 elaborates a NIZKPoK z_1 of a vector of square-roots of the vector of squares (819). (This is avoidable if for P_1 has already provided a NIZKPoK of trapdoor in a setup phase.) Then, P_1 prepares randomness $\underline{c}$ for an Ext&Equiv Com (820) of the pair composed of the vector of random squares and the respective NIZKPoK of square-roots. It uses the randomness to produce the commitment $\overline{c}$ (821), and sends it in a contextualized message to P_1 (822).
- **Phase 2.** P_2 selects a random group element $\gamma^{(2)}$ as her contribution (823), and sends it

in a contextualized message to P_1 (824). P_1 compute his final output γ, as a combination of the pre-image $\gamma^{(1)}$ of his contribution and the contribution $\gamma^{(2)}$ received from P_2 (825).

- **Phase 3.** P_1 prepares a NIZKP that his contribution is a vector of squares, i.e., that they are correct BitComs of 0 (826). Then, P_1 prepares the auxiliary information $\underline{\underline{c}}$ needed to open the previous commitment (827). Since the commitment is equivocable (besides extractable), the auxiliary information may have a sintax different from the actual randomness used by P_2 to build the commitment. P_1 sends a contextualized message to P_2, containing the committed values, the remaining auxiliary info need to verify it, and the new NIZKP transcript (828). P_2 verifies the correctness of the NIZKPoK (829), and of the NIZKP (830). If something is wrong then it aborts, otherwise it continues. P_2 calculates the square (i.e., the f_1-image) $\gamma'^{(2)}$ of her contribution (831), and then combines it with the previous square contribution $\gamma'^{(1)}$ received from P_1, thus obtaining her final output square γ' (832).
- **Final outputs** P_1 outputs the product of square-roots (833), whereas P_2 outputs the product of squares (834).

Complexity. The overall communication complexity of the three messages is ℓ group elements from P_2, one NIZKP of squares (i.e., a NIZKP of GM BitComs of 0), one Ext&Equiv-Com and opening of a vector with ℓ group elements and also of a respective NIZKPoK of square-roots. In a PKI setting where the simulator has access to the trapdoor (e.g., via another NIZKPoK), the NIZKPoK of square-roots is not necessary, because the trapdoor is enough to extract square-roots. However, in a global-CRS setting an explicit NIZKPoK would be needed. The NIZKP of GM BitComs of 0 only requires communication of group elements in number twice the statistical security parameter (in this non-interactive case the statistical parameters is equal to the computational security parameter), and one Equiv-Com (see row 7 in Table B.2). The communication of the commit and open phases of the Ext-and-Equiv-Com depend on the size of the committed value and also on the cut-and-choose and erasure code parameters (e.g., see Table C.1).