University of Connecticut

Abstract

Efficient and Adaptively Secure Cryptographic Primitives —Designing for Composability

Hong-Sheng Zhou
2010
Major Advisor:
Prof. Aggelos Kiayias
Department of Computer Science & Engineering

We study efficient protocol constructions against adaptive corruption in the universal composition framework. For standard cryptographic tasks, we propose a new framework to design efficiently two-party secure function evaluation protocols, and then apply it to oblivious transfer and obtain the first practical such constructions. Regarding reactive cryptographic tasks, we present a framework for designing blind signatures, and construct the first practical protocols for this task.

Efficient and Adaptively Secure Cryptographic Primitives —Designing for Composability

Hong-Sheng Zhou

B.E., Nanjing University of Posts & Telecommunications, 1998

M.E., Shanghai Jiaotong University, 2004

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Connecticut

2010

APPROVAL PAGE

Doctor of Philosophy Dissertation

Efficient and Adaptively Secure Cryptographic Primitives – Designing for Composability

Presented by

Hong-Sheng Zhou, B.E., M.E.

Major Advisor: [illegible]

Prof. Aggelos Kiayias

Associate Advisors: [illegible]

Dr. Juan A. Garay

[illegible]

Prof. Alexander C. Russell

[illegible]

Prof. Zhijie Jerry Shi

[illegible]

Prof. Alexander A. Shvartsman

University of Connecticut

2010

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Aggelos Kiayias for giving me the opportunity to learn cryptography and for contributing the most to my personal development as a researcher. During the years at UConn, Aggelos invested huge amount of energy to teach me the basics, and to reshape my independent thinking abilities. He shared with me many of his brilliant ideas, but more importantly, he taught me how to identify interesting problems and to provide the solutions, and how to pursue the true nature of things and the beauty. Guided by him, I found my own approach to cryptography research. Aggelos is the greatest person to work with; he has the amazing capability to understand my fuzzy intuitions (to be honest, sometime even I did not know what I was talking about but he could extract something meaningful), and he is always available to answer my questions. It was his persistent encouragement that motivated me through the ups and downs during my PhD studies. I will always miss the days under his guidance, and be indebted to him for all his help and for the greatest example he has set to me.

I would like to express my gratitude to the members of my dissertation committee, Juan Garay, Alex Russell, Jerry Shi, and Alex Shvartsman, for all their time and support, and for providing me many helpful technical suggestions. The technical parts in the dissertation are based on joint work with Juan Garay, Daniel Wichs, and my advisor, and I thank them for allowing me to use it here.

My special thanks go to Alex Russell for teaching me the great course Computational Complexity which supplied me a solid background for cryptography research, for providing me many individual and fun meetings when Aggelos was in the army or on sabbatical, and for running interesting reading groups together. Alex is always willing to teach me math or theory in general, and my interactions with him have significantly influenced my view on research. Thanks, Alex!

My special thanks go to Juan Garay. At TCC'06, Aggelos introduced me to Juan and we started to work together one year later. Juan together with Aggelos have given me critical help in my PhD studies. He threw at me several interesting research topics and provided me with many technical discussions, mixed with his rich knowledge about beers and coffee and colorful stories in crypto community — our meetings were never boring. In addition, I thank him for an interesting summer at Bell-Labs.

I thank Moti Yung for his collaboration, suggestions, encouragement, and help. I thank Daniel Wichs for sharing me his knowledge, broadening my view of cryptography research, and his collaboration.

I thank Ran Canetti, Stas Jarecki, Jon Katz, Vlad Kolesnikov, Jesper Nielsen, Manoj Prabhakaran, Abhi Shelat, and many researchers in the New York Metro area for many interesting and insightful conversations.

I thank all the members and visitors in our Crypto-DRM Lab, Sotiris Kentros, Michael Korman, Justin Neumann, Murat Osmanoglu, Serdar Pehlivanoglu, Yona Raekow, Qiang Tang, David Walluck, and Lei Fan, Maged Ibrahim, for giving me their help, and for students in the computer science department for maintaining great research atmosphere. I thank Charles Ju, Baikang Pei, Fan Zhang, Haimin Zhang, and many friends here for making me live at Storrs better.

Finally, I thank my mother, my bother and my sister and the extended family back in China, and my wife for their endless support and love.

My PhD studies were supported by Aggelos' NSF grants #0447808, #0831304, and #0831306.

DEDICATION

to my dear wife

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

Cryptography has been with us for many years, in its traditional role of protecting information transmission via encryption and authentication. However, with the more recent advancement of information and communication techniques and their deployment becoming ubiquitous and common-place, cryptography has broadened its scope with applications such as electronic voting, privacy preserving data-mining, and digital rights management, in conjunction to its role as a critical enabler of new ways of interaction. At the same time, the environments where those applications run, such as the Internet, are much more complex than yesteryear's networks for which secure communication protocols were designed and analyzed, resulting not only in new vulnerabilities, but also in more unpredictable situations and attacks. Protocols secure in the traditional stand-alone setting are unlikely to be secure in the Internet setting. The adversaries can corrupt parties at any moment during protocol execution. The corrupted parties can deviate from protocol specifications and share information among different protocols. The adversaries can delay the communication between parties, eavesdrop on the communication channel, and/or even inject malicious messages.

To date, universal composition security [Can01] is one of the most advanced notions for defining security in cryptography. Protocols satisfying UC security are secure within any environment. However, while Canetti's UC framework provides a feasible way to design multi-party computation (MPC) protocols in the Internet setting, it is still elusive to give efficient constructions. The main goal of this study is to design efficient and practical protocols in complex environment. We investigate two important privacy primitives, oblivious transfer and blind signatures, under more realistic attacks. Along the way, we identify several key techniques could be useful in efficient protocol design in general.

Protocol design of blind signature and oblivious transfer. Blind signature is a very useful privacy primitive, which allows a signer to interactively sign messages for multiple users such that the messages are not revealed to the signer. Since the initial introduction of the primitive [Cha82], it has been intensively used to design electronic cash schemes, electronic voting schemes and anonymous credential systems. In Section 4, we first defined the protocol specification (also called functionality) for blind signature. To implement the functionality efficiently, we disassembled the traditional security properties, "trimmed the fat", and then abstracted a lightweight core. We further assembled the lightweight core with tailored and highly efficient zero-knowledge proofs to achieve UC security. Our blind signature protocols are the first practical and adaptively secure constructions. Note that before our result there had not been practical blind signatures in the UC setting. These results are joint work with Aggelos Kiayias, and appeared in [KZ08, KZ07].

Oblivious transfer is a fundamental primitive in cryptography as the basis of a huge class of two-party and multi-party cryptographic tasks. OT allows a receiver to obtain exactly one of two (or more) messages from a sender where the receiver remains oblivious to the other message(s), and the sender is oblivious to which value was received. The OT protocols presented in Section 3 are the first practical and adaptively secure constructions. These results are joint work with Juan Garay and Daniel Wichs, and appeared in [GWZ09].

Adaptive security and efficiency. Currently many protocols are secure only against static adversaries, who corrupt parties only at the very beginning of protocol executions. However in a realistic setting, protocols must be designed to defend against adaptive adversaries, who have the power to corrupt parties at any moment during a protocol execution. For most tasks, adaptive security is significantly harder to achieve, and the corresponding constructions might be much less efficient. I am interested in designing adaptively secure protocols with higher efficiency for practical applications.

Generally an adaptively secure high-level protocol requires adaptively secure low-level building blocks. In [KZ07], we identified a family of protocols which are adaptively secure by using a zero-knowledge proof as a building block. Furthermore, the protocols are still adaptively secure even using a much weaker primitive called *leaking zero-knowledge proof.* The leaking zero-knowledge proof, unlike the zero-knowledge proof, can be realized much more efficiently. In addition, this technique was used in [KZ08] to make our blind signature constructions more efficient.

In [GWZ09], we made significant progresses in adaptive security. We introduced a new notion called *semi-adaptive security* which is stronger than static security but weaker than fully adaptive security. We then gave a simple, generic protocol compiler which transforms any semi-adaptively secure protocol into a fully adaptively secure one. The compilation effectively decomposes the problem of adaptive security into two simpler ones: the problem of semi-adaptive security and the problem of realizing a weaker variant of secure channels. We solved the latter problem by means of a new primitive that we call *somewhat non-committing encryption*, resulting in significant efficiency improvements over the standard method for realizing secure channels using fully non-committing encryption. The power of our methodology is demonstrated by transforming the recent efficient statically-secure oblivious transfer protocols [PVW08] to adaptively-secure ones.

Organization. In Chapter 2, we give a brief introduction of the universal composition framework, and the basic cryptographic primitives used in our efficient protocol design. In Chapter 3, we focus on the design of efficient oblivious transfer including two main parts: we first propose a framework for efficient two-party function evaluation construction, and then apply this framework to oblivious transfer and give concrete constructions under number theoretical assumptions; part of this chapter appears in CRYPTO'09. In Chapter 4, we propose a framework for efficient blind signature construction, and give concrete constructions under number theoretical assumptions; part of this chapter appears in TCC'08 and ICALP'07.

Chapter 2
PRELIMINARIES

2.1 Notations

All algorithms are in polynomial-time in the security parameter λ. By $(\langle a,b,c\rangle,\eta) \leftarrow A(x)$ we mean that algorithm A, with input x, outputs values a,b,c, where η is the randomness. If A is deterministic, η will be ignored, i.e., $\langle a,b,c\rangle \leftarrow A(x)$. By $(\langle a,b\rangle,\eta;c,\zeta) \leftarrow [\![A(x);B(y)]\!]_{\mathrm{LR}}$, we denote an execution between two interactive Turing machines A and B, where the left machine A on input x generates output a,b by using randomness η, while the right machine B on inputs y generates outputs c by randomness ζ. If only left side of the output is of concern, we write it as $(\langle a,b\rangle,\eta) \leftarrow [\![A(x);B(y)]\!]_{\mathrm{L}}$, and similarly for the right side. We use $\overset{c}{\approx}$ to denote computational indistinguishability, and $\overset{s}{\approx}$ for statistical indistinguishability.

2.2 The Universal Composability Framework

The Universal Composability (UC) framework proposed by Canetti [Can01, Can05], culminating a long sequence of simulation-based security definitions (cf. [Yao82, GMW87, GL90, MR91, Bea91, Can00, HM00, DM00, PW00]; see also [PW01, BPW07, DKM$^+$04, KDMR08, PS04, Küs06, CDPW07, LPV09] for alternative/extended frameworks), allows for arguing the security of cryptographic protocols in arbitrary settings where executions can be concurrent and adversarially interleaved. The framework is particularly attractive for the design of secure systems as it supports modularity, provides non-malleability across sessions [DDN00], and preserves security under composition.

In this framework one first defines an "ideal functionality" of a protocol, and then proves that a particular implementation of this protocol operating in a given computational environment securely realizes this ideal functionality. The basic entities involved are n players $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The real execution of a protocol π, run by the players in the presence of $\mathcal{A}$ and an environment machine $\mathcal{Z}$, with input z, is modeled as a sequence of *activations* of the entities. The environment $\mathcal{Z}$ is activated first, generating in particular the inputs to the other players. Then the protocol proceeds by having $\mathcal{A}$ exchange messages with the players and the environment. Finally, the environment outputs one bit, which is the output of the protocol. Please refer to Figure 2.1 for a high-level pictorial description.

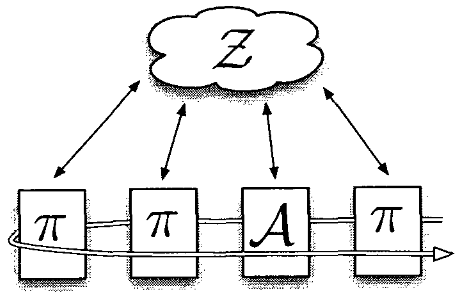


Figure 2.1: The real world. Here protocol parties are running π. The adversary $\mathcal{A}$ observes protocol communication between parties (denoted by double stroke arrow), and it might corrupt some parties. For example here the third party is corrupt.

The security of the protocols is defined by comparing the real execution of the protocol

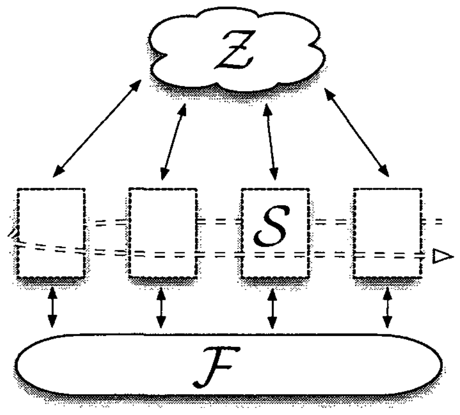


Figure 2.2: The ideal world. Here dummy parties (denoted by dotted boxes) directly forward the inputs/outputs between environment $\mathcal{Z}$ and functionality $\mathcal{F}$. The ideal adversary $\mathcal{S}$ might corrupt some parties, and on behalf these parties send inputs to or receive outputs from $\mathcal{F}$. In addition $\mathcal{S}$ might generate simulated protocol communication (denoted by double dotted stroke arrow), and delay the outputs from $\mathcal{F}$ to the dummy parties.

to an ideal process in which an additional entity, the ideal functionality $\mathcal{F}$, is introduced; essentially, $\mathcal{F}$ is an incorruptible trusted party that is programmed to produce the desired functionality of the given task. The players are replaced by dummy players, who do not communicate with each other; whenever a dummy player is activated, it forwards its input to $\mathcal{F}$. Let $\mathcal{S}$ denote the adversary in this idealized execution. As in the real-life execution, the output of the protocol execution is the one-bit output of $\mathcal{Z}$. Please refer to Figure 2.2 for a pictorial description of the ideal execution.

Now a protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-execution adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π in the real-life execution, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal execution. More precisely, if the two binary distribution ensembles, $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{A}$ and players running protocol π (resp., adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$), are computationally indistinguishable (denoted $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$). For further details on the UC framework refer to [Can05].

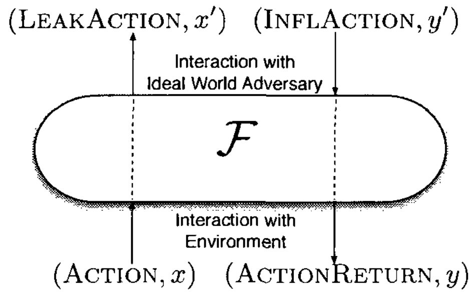


Figure 2.3: An ideal functionality template. Here a single action Action is considered.

In this thesis, several ideal functionalities will be introduced. We adapt the presentation style from [GKZ10]. In Figure 2.3, we present a pictorial version of a functionality where a single action is considered. For reactive tasks, multiple actions will be introduced. The functionality $\mathcal{F}$ (denoted by elliptic circle) operates based on some internal instructions or codes. See examples shortly.

2.3 *Other Frameworks Related to UC*

In all previously mentioned simulation-based security frameworks, the activations are scheduled *sequentially*, in the sense that, in a system of processes executing in parallel, only one process is allowed to be activated at any point, and when the active process produces a message, the intended recipient is the next active one. Here, if the chain of activation is broken, then a designated master process will be triggered. The underlying computational entities include probabilistic interactive Turing machine [Can01, Küs06], probabilistic I/O automata [PW01, BPW07] and the sequential probabilistic process calculus [DKM$^+$04, KDMR08].

However, in the physical world of distributed systems, computations are carried out in many different places *concurrently*. In the above frameworks with sequential activations, this issue is resolved by providing "pseudo-concurrency" to approximate the "true concurrency".

In the literature, there are frameworks that represent concurrency aspects of distributed systems directly by considering non-deterministic scheduling and then quantifying over all possible schedules, including the probabilistic polynomial-time process calculus of [LMMS98, MMS03, DKM$^+$04, MRST06] by using non-sequential scheduling with state-dependent functions (or Markov chains).

Recently Canetti et al. [CCK$^+$05, CCK$^+$07a, CCK$^+$08, CCK$^+$07b] extend the Probabilistic I/O Automata (PIOA) of Lynch, Segala and Vaandrager [LSV03] to a security framework for analyzing cryptographic protocols along the lines of the UC framework. Here a special mechanism, called the task schedule, is introduced for curbing the power of non-sequential scheduling. The resulting framework represents the concurrent nature of distributed systems in a direct way. It also allows for analyzing partially-specified protocols, and allows some scheduling choices to be determined non-deterministically during run-time. Canetti et al. [CCLP07] also show that the security models with sequential scheduling are incomparable with the ones with non-sequential scheduling; please refer to their paper for interesting separation examples.

2.4 *Typical Setup: Common Reference String*

As was observed in [CKL03], most functionalities cannot be realized in the UC framework without some setup. One common form of setup is the common reference string (CRS). We model a CRS as an ideal functionality $\mathcal{F}^{D}_{\text{CRS}}$ which is shown in Figure 2.4. Please refer to [Can07] for an excellent survey about variant trust setups.

Functionality $\mathcal{F}^{D}_{\text{CRS}}$

$\mathcal{F}_{\text{CRS}}$, parameterized by a distribution D, interacts with a set $\mathcal{P}$ of parties and adversary $\mathcal{S}$. For each input or message, the functionality verifies that $sid = (\mathcal{P}, sid')$ for some sid', and ignores it otherwise.

CRS generation:

- On input (CRS, P, sid) from $P \in \mathcal{P}$, if there is no value crs recorded then choose $crs \overset{\$}{\leftarrow} D$ and record it. Send $(\text{LeakCRS}, P, sid, crs)$ to $\mathcal{S}$.
- On receiving $(\text{InflCRS}, P, sid)$ from $\mathcal{S}$, send $(\text{CRSReturn}, P, sid, crs)$ to P.

Figure 2.4: The common reference string ideal functionality, $\mathcal{F}_{\text{CRS}}$.

2.5 *Digital Signatures*

Digital signature schemes allow a distinguished party (i.e., the signer), who has some secret information (i.e., secret key), to generate tag strings (i.e., signatures) for plaintexts, so that all parties (i.e., the verifiers) can make sure that the signatures were generated by the signer but not someone else. Digital signature schemes were first proposed by Diffie and Hellman [DH76], and basic security requirements were formally defined by Goldwasser, Micali and Rivest [GMR88]. Ideal functionality of digital signatures was first studied in [Can01], and later several revisions were made. The formulation presented in Figure 2.5 is from [Can05].

Functionality $\mathcal{F}_{\mathrm{SIG}}$

$\mathcal{F}_{\mathrm{SIG}}$ interacts with adversary $\mathcal{S}$, and a distinguished signer S and a set $\mathcal{V}$ of verifiers V. For each input or message, the functionality verifies that $sid = (S, sid')$ for some sid', and ignores it otherwise.

Key generation:

- Upon receiving ($\textsc{KeyGen}, S, sid$) from party S, forward ($\textsc{LeakKeyGen}, S, sid$) to $\mathcal{S}$.
- Upon receiving ($\textsc{InflKeyGen}, S, sid, \langle \mathsf{s}, \mathsf{v}\rangle$) from the adversary $\mathcal{S}$, record $\langle \mathsf{s}, \mathsf{v}\rangle$ in $\mathsf{history}_S$ and output ($\textsc{KeyGenReturn}, S, sid, \mathsf{v}$) to party S, where s is a probabilistic poly-time algorithm, and v is a deterministic poly-time algorithm.

Signature generation:

- Upon receiving ($\textsc{Sign}, S, sid, m$) from party S, compute $(\sigma, \zeta) \leftarrow \mathsf{s}(m)$ and verify that $\mathsf{v}(m, \sigma) = 1$. If so, then output ($\textsc{SignReturn}, S, sid, \sigma$) to party S, and record $\langle m, \sigma, \zeta\rangle$ into $\mathsf{history}_S$. Else, halt.

Signature verification:

- Upon receiving ($\textsc{Verify}, V, sid, \langle m, \sigma, \mathsf{v}'\rangle$) from party $V \in \mathcal{V}$, do: if $\mathsf{v}' = \mathsf{v}$, the signer S is not corrupted, $\mathsf{v}(m, \sigma) = 1$, and m is not recorded, then halt. Else, output ($\textsc{VerifyReturn}, V, sid, \mathsf{v}'(m, \sigma)$) to party V.

Corruption:

- Upon receiving ($\textsc{Corrupt}, J, sid$) from $\mathcal{S}$, return ($\textsc{CorruptReturn}, J, sid, \mathsf{history}_J$) to $\mathcal{S}$.

Figure 2.5: Signature functionality $\mathcal{F}_{\mathrm{SIG}}$.

Definition 2.5.1 (Secure Signature Schemes [GMR88]). A signature scheme $\Sigma(\mathsf{SIG}) = \mathsf{SIG}.\{\mathsf{KG}, \mathsf{SG}, \mathsf{Vrf}\}$ is <u>secure</u> if the following properties hold:

– Completeness: For all PPT $\mathcal{A}$,

$$\Pr\left[\ (vk,sk) \leftarrow \mathsf{KG}(1^\lambda); m \leftarrow \mathcal{A}(vk); (\sigma,\zeta) \leftarrow \mathsf{SG}(vk,sk,m); \phi \leftarrow \mathsf{Vrf}(vk,m,\sigma)\ :\ \phi = 0\ \right] \overset{c}{\approx} 0.$$

– Consistency: For any PPT $\mathcal{A}$,

$$\Pr\left[\ (vk,m,\sigma) \leftarrow \mathcal{A}(1^\lambda); \phi_1 \leftarrow \mathsf{Vrf}(vk,m,\sigma); \phi_2 \leftarrow \mathsf{Vrf}(vk,m,\sigma)\ :\ \phi_1 \neq \phi_2\ \right] \overset{c}{\approx} 0.$$

– Unforgeability: *(Existential Unforgeability against Chosen Message Attacks (EU-CMA))*

For any PPT forger $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} (vk,sk) \leftarrow \mathsf{KG}(1^\lambda); (m,\sigma) \leftarrow \mathcal{A}^{\mathsf{SG}(vk,sk,\cdot)}(vk); \phi \leftarrow \mathsf{Vrf}(vk,m,\sigma) \\ : \phi = 1 \text{ and } \mathcal{A} \text{ never submitted } m \text{ to the } \mathsf{Sign}(vk,sk,\cdot) \text{ oracle} \end{array}\right] \overset{c}{\approx} 0.$$

□

Remark 2.5.2. Consistency was first explicitly investigated by Canetti [Can04]. Here we adopt the corrected version of consistency by Garay et al [GKZ10].

Based on a digital signature scheme $\Sigma(\mathsf{SIG})$, we can obtain the corresponding protocol $\pi_{\Sigma(\mathsf{SIG})}$ as presented in Figure 2.6. In [Can05], Canetti proves the following theorem.

Theorem 2.5.3. $\Sigma(\mathsf{SIG})$ *is GMR-secure* $\Leftrightarrow$ $\pi_{\Sigma(\mathsf{SIG})}$ *securely realizes* $\mathcal{F}_{\mathsf{SIG}}$.

2.6 Okamoto Signature

In our construction, we will use the signature recently proposed in section 5 in [Oka06], which is based on bilinear groups, and is EU-CMA secure under q-2SDH assumption.

Protocol $\pi_{\Sigma(\mathsf{SIG})}$

Key generation: When party S is invoked with (KeyGen, sid) by $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \mathsf{KG}(1^\lambda)$, stores $\langle vk, sk \rangle$, defines the verification algorithm $\mathsf{v} := \mathsf{Vrf}(vk, \cdot, \cdot)$, and outputs (KeyGenReturn, sid, v) to $\mathcal{Z}$.

Signature generation: When party S is invoked with (Sign, sid, m) by $\mathcal{Z}$ where $sid = (S, sid')$, it computes $(\sigma, \zeta) \leftarrow \mathsf{s}(m)$ and outputs (SignReturn, sid, σ) to $\mathcal{Z}$.

Signature verification: When party V is invoked with (Verify, sid, m, σ, v') by $\mathcal{Z}$ where $sid = (S, sid')$, it outputs (VerifyReturn, sid, $\mathsf{v}'(m, \sigma)$) to $\mathcal{Z}$.

Corruption: When party J is invoked with (Corrupt, sid, J) by $\mathcal{Z}$, it returns (CorruptReturn, sid, $\mathsf{history}_J$) to $\mathcal{Z}$.

Figure 2.6: Signature protocol $\pi_{\Sigma(\mathsf{SIG})}$.

Bilinear Groups. Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order p so that (i) $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$; (ii) $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is an isomorphism with $\psi(g_2) = g_1$ and (iii) $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. We remark that in some cases it can be that $\mathbb{G}_1 = \mathbb{G}_2$ (and in this case ψ_2 would be the identity mapping). Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ groups as above with $|\mathbb{G}_1| = |\mathbb{G}_2| = p$; a bilinear map is a map $\hat{e}$ s.t. for all $(u, v) \in \mathbb{G}_1 \times \mathbb{G}_2$ it holds that $\hat{e}(a^x, b^y) = \hat{e}(a, b)^{xy}$ and $\hat{e}(g_1, g_2) \neq 1$.

Definition 2.6.1 (2-Variable Strong Diffie-Hellman (2SDH) Assumption)**.** Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups defined as above. The q-2SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given a $(3q+4)$-tuple $\langle g_1, g_2, w_2 \leftarrow g_2^x, u_2 \leftarrow g_2^y, g_2^{\frac{y+b_1}{x+a_1}}, \ldots, g_2^{\frac{y+b_q}{x+a_q}}, a_1, \ldots, a_q, b_1, \ldots, b_q \rangle$ as input, output $\langle \varsigma \leftarrow g_1^{\frac{y+d}{fx+r}}, \alpha \leftarrow g_2^{fx+r}, d, V_1, V_2 \rangle$ where $a_1, \ldots, a_q, b_1, \ldots, b_q, d, f, r \in \mathbb{Z}_p$; $w_1 \leftarrow \psi(w_2), \varsigma, V_1 \in \mathbb{G}_1$; $\alpha, V_2 \in \mathbb{G}_2$; and $\hat{e}(\varsigma, \alpha) = \hat{e}(g_1, u_2 g_2^d)$, $\hat{e}(V_1, \alpha) = \hat{e}(w_1, w_2) \cdot \hat{e}(g_1, V_2)$, $d \notin \{b_1, \ldots, b_q\}$. The q-2SDH assumption suggests that any PPT algorithm $\mathcal{A}$ solving the q-2SDH problem has negligible success probability, i.e.

$$\Pr\left[\begin{array}{l} x,y \in \mathbb{Z}_p; g_2 \in \mathbb{G}_2; g_1 \leftarrow \psi(g_2); w_2 \leftarrow g_2^x; u_2 \leftarrow g_2^y; \\ a_1,\ldots,a_q,b_1,\ldots,b_q \in \mathbb{Z}_p; c_1 \leftarrow g_2^{\frac{y+b_1}{x+a_1}},\ldots,c_q \leftarrow g_2^{\frac{y+b_q}{x+a_q}}; \\ (\varsigma,\alpha,d,V_1,V_2) \leftarrow \mathcal{A}(g_1,g_2,w_2,u_2,a_1,\ldots,a_q,b_1,\ldots,b_q,c_1,\ldots,c_q): \\ \qquad \hat{e}(\varsigma,\alpha) = \hat{e}(g_1,u_2g_2^d) \wedge \hat{e}(V_1,\alpha) = \hat{e}(w_1,w_2)\cdot\hat{e}(g_1,V_2) \wedge d \notin \{b_1,\ldots,b_q\} \end{array}\right] \overset{c}{\approx} 0.$$

□

Next we briefly introduce Okamoto signature.

- Key generation: Randomly select $g_2,u_2,v_2 \xleftarrow{\$} \mathbb{G}_2$ and set $g_1 \leftarrow \psi(g_2)$, $u_1 \leftarrow \psi(u_2)$ and $v_1 \leftarrow \psi(v_2)$. Randomly select $x \xleftarrow{\$} \mathbb{Z}_p$ and compute $X \leftarrow g_2^x \in \mathbb{G}_2$. Set $vk = \langle g_1,g_2,u_2,v_2,X\rangle$ and secret key $sk = \langle x\rangle$.
- Signature generation: Let $m \in \mathbb{Z}_p$ be the message to be signed. Signer S randomly selects r and s from $\mathbb{Z}_p$ s.t. $x + r \not\equiv 0 \bmod p$; and compute $\varsigma \leftarrow (g_1^m u_1 v_1^s)^{\frac{1}{fx+r}}$, $\alpha \leftarrow g_2^{fx+r}$, $V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$, $V_2 \leftarrow X^{fh+\frac{r}{f}} g_2^{rh}$, where $f,r,s,h \xleftarrow{\$} \mathbb{Z}_p$. The signature for m is $\sigma = \langle \varsigma,\alpha,s,V_1,V_2\rangle$.
- Signature verification: Given verification key $vk = \langle g_1,g_2,u_2,v_2,X\rangle$, message m, and signature $\sigma = \langle \varsigma,\alpha,s,V_1,V_2\rangle$, check that $m,s \in \mathbb{Z}_p$, $\varsigma, V_1 \in \mathbb{G}_1$, $\alpha, V_2 \in \mathbb{G}_2$, $\varsigma \neq 1$, $\alpha \neq 1$ and $\hat{e}(\varsigma,\alpha) = \hat{e}(g_1, g_2^m u_2 v_2^s)$, $\hat{e}(V_1,\alpha) = \hat{e}(\psi(X),X)\cdot\hat{e}(g_1,V_2)$. If they hold, then the verification is valid; otherwise invalid.

2.7 Commitments

A commitment scheme is a protocol between two parties (i.e., the committer C and the receiver V) that allows the committer C to commit to a message m so that the receiver cannot learn such m. Later the committer opens the commitment and reveals to the receiver that she committed to m, and at this moment the committer cannot change her mind and the commitment cannot be opened to a different message m' where $m' \neq m$. In this thesis, we use non-interactive commitment scheme, where the committer can compute the commitment by herself and then send it to the receiver; in the opening stage, the committer sends m and the random coins used to the receiver at once, and the receiver simply run the commitment verification algorithm by himself to make sure if m was the message the committer had committed to.

Definition 2.7.1 (Secure Commitment Schemes). A (non-interactive) commitment scheme $\Sigma(\mathsf{COM}) = \mathsf{COM}.\{\mathsf{CRS}, \mathsf{Com}, \mathsf{Vrf}\}$ is secure if the following properties hold:

– Completeness: For all PPT $\mathcal{A}$,

$$\Pr\left[\ crs \leftarrow \mathsf{CRS}(1^\lambda); m \leftarrow \mathcal{A}(crs); (c,\zeta) \leftarrow \mathsf{Com}(crs,m); \phi \leftarrow \mathsf{Vrf}(crs,c,m,\zeta) : \phi = 0\ \right] \overset{c}{\approx} 0.$$

– Binding: For all PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} crs \leftarrow \mathsf{CRS}(1^\lambda); (c, m_0, \zeta_0, m_1, \zeta_1) \leftarrow \mathcal{A}(crs); \phi_0 \leftarrow \mathsf{Vrf}(crs, c, m_0, \zeta_0); \\ \phi_1 \leftarrow \mathsf{Vrf}(crs, c, m_1, \zeta_1) = 1 \ : \ \phi_0 = \phi_1 = 1 \wedge m_0 \neq m_1 \end{array}\right] \overset{c}{\approx} 0.$$

– (Indistinguishable) Hiding: For all PPT $\mathcal{A}$, $\Pr[\mathbf{IndHid}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{IndHid}_{\mathcal{A}}(\lambda, 1)]$, where $\mathbf{IndHid}_{\mathcal{A}}(\lambda, b)$ is defined as below:

```
$\mathbf{IndHid}_{\mathcal{A}}(\lambda, b)$
$crs \leftarrow \mathsf{CRS}(1^\lambda)$;
$(m_0, m_1) \leftarrow \mathcal{A}(crs)$;
$(c_b, \zeta_b) \leftarrow \mathsf{Com}(crs, m_b)$;
Return $b' \leftarrow \mathcal{A}(c_b)$.
```

□

Above we give an game-based formulation of the hiding property. Below we give a formulation of such property in the simulation paradigm.

- (Simulatable) Hiding: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}$, $\widetilde{\mathsf{Com}}$ such that $\Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)]$, where $\mathbf{SimHid}_{\mathcal{A}}(\lambda, b)$ is defined as

$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)$	$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)$
$crs \leftarrow \mathsf{CRS}(1^\lambda)$;	$(crs, \tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
$b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot)}(crs)$;	$b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot)}(crs)$;
Return b'.	Return b'.

```
$\mathcal{O}_1(crs, m)$                          $\mathcal{O}_0(crs, \tau, m)$
    $(c, \zeta) \leftarrow \mathsf{Com}(crs, m)$;          $(c, \xi) \leftarrow \widetilde{\mathsf{Com}}(crs, \tau)$;
    Output $c$.                              Output $c$.
```

We note that secure commitment schemes can be constructed in the plain model. In that case, $\widetilde{\mathsf{CRS}} := \mathsf{CRS}$ and only security parameter will be included in *crs*, and the trapdoor τ is empty. Here we introduce two additional security properties of commitment, equivocality and extractability. We also note that such properties cannot be achieved in the plain model, and we formulate them under the trust setup of CRS.

Definition 2.7.2 (Equivocal Commitments). We say a secure commitment scheme $\Sigma(\mathsf{COM}) = \mathsf{COM}.\{\mathsf{CRS}, \mathsf{Com}, \mathsf{Vrf}\}$ is equivocal if the following property holds:

- Equivocality: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Com}}$, $\widetilde{\mathsf{Eqiv}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot)}(crs)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)$
    $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs,m)$
    $(c,\zeta) \leftarrow \mathsf{Com}(crs,m)$;
    Output $(c,\zeta)$.

$\mathcal{O}_0(crs,\tau,m)$
    $(c,\xi) \leftarrow \widetilde{\mathsf{Com}}(crs,\tau)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(c,\xi,m)$;
    Output $(c,\zeta)$.
```

□

Definition 2.7.3 (Extractable Commitments). We say a secure commitment $\Sigma(\mathsf{COM}) = \mathsf{COM}.\{\mathsf{CRS},\mathsf{Com},\mathsf{Vrf}\}$ is <u>extractable</u> if the following property holds.

- Extractability: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Ext}}$ such that

$$\Pr\left[\begin{array}{r} (crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda); (c,m,\zeta) \leftarrow \mathcal{A}(crs); \phi \leftarrow \mathsf{Vrf}(crs,c,m,\zeta); \\ m' \leftarrow \widetilde{\mathsf{Ext}}(crs,\tau,c) \;:\; \phi = 1 \wedge m \neq m' \end{array}\right] \overset{c}{\approx} 0.$$

□

2.8 *Non-Interactive Zero-Knowledge*

Definition 2.8.1 (Secure Non-Interactive Zero-Knowledge (NIZK) Schemes). An NIZK scheme $\Sigma(\mathsf{NIZK}) = \mathsf{NIZK}.\{\mathsf{CRS},\mathsf{Prv},\mathsf{Vrf}\}$ is <u>secure</u> for relation R if the properties hold:

- Completeness: For any PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{r} crs \leftarrow \mathsf{CRS}(1^\lambda); (x,w) \leftarrow \mathcal{A}(crs); (\varsigma,\zeta) \leftarrow \mathsf{Prv}(crs,x,w); \\ \phi \leftarrow \mathsf{Vrf}(crs,x,\varsigma) : (x,w) \in R \land \phi = 0 \end{array}\right] \overset{c}{\approx} 0.$$

- Soundness: For all PPT $\mathcal{A}$,

$$\Pr\left[crs \leftarrow \mathsf{CRS}(1^\lambda); (x,\varsigma) \leftarrow \mathcal{A}(crs); \phi \leftarrow \mathsf{Vrf}(crs,x,\varsigma) : x \notin \mathcal{L}_R \land \phi = 1\right] \overset{c}{\approx} 0.$$

- Zero-Knowledge: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Prv}}$ such that $\Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{SimZK}_{\mathcal{A}}(\lambda,b)$ is defined as

```
$\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)$                                $\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;                             $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;              $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.                                   Return $b'$.

$\mathcal{O}_1(crs,x,w)$                                  $\mathcal{O}_0(crs,\tau,x,w)$
    $(\varsigma,\zeta) \leftarrow \mathsf{Prv}(crs,x,w)$;                 $(\varsigma,\xi) \leftarrow \widetilde{\mathsf{Prv}}(crs,\tau,x)$;
    Output $\varsigma$.                                   Output $\varsigma$.
```

□

Definition 2.8.2 (Non-Erasure Zero-Knowledge NIZKs). We say a secure NIZK scheme $\Sigma(\mathsf{NIZK}) = \mathsf{NIZK}.\{\mathsf{CRS},\mathsf{Prv},\mathsf{Vrf}\}$ is non-erasure zero-knowledge for relation R, if the following property holds:

- Non-Erasure Zero-Knowledge: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Prv}}$,

$\widetilde{\mathsf{Eqiv}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda,b)$ is defined as

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)$
    $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs,x,w)$
    $(\varsigma,\zeta) \leftarrow \mathsf{Prv}(crs,x,w)$;
    Output $(\varsigma,\zeta)$.

$\mathcal{O}_0(crs,\tau,x,w)$
    $(\varsigma,\xi) \leftarrow \widetilde{\mathsf{Prv}}(crs,\tau,x)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(\varsigma,\xi,w)$;
    Output $(\varsigma,\zeta)$.
```

□

Definition 2.8.3 (Knowledge-Extractable NIZKs). We say a secure NIZK scheme $\Sigma(\mathsf{NIZK}) = \mathsf{NIZK}.\{\mathsf{CRS},\mathsf{Prv},\mathsf{Vrf}\}$ is <u>knowledge-extractable</u> for relation R, if the following property holds:

– Knowledge-Extraction: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}$, $\widetilde{\mathsf{Ext}}$ such that

$$\Pr\left[\begin{array}{r} (crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda); (x,\varsigma) \leftarrow \mathcal{A}(crs); \phi \leftarrow \mathsf{Vrf}(crs,x,\varsigma); \\ m \leftarrow \widetilde{\mathsf{Ext}}(crs,\tau,x,\varsigma) \,:\, \phi = 1 \wedge (x,w) \notin R \end{array}\right] \overset{c}{\approx} 0.$$

□

2.9 *Interactive Zero-Knowledge Proofs*

Definition 2.9.1 (Secure Zero-Knowledge (ZK) Schemes). An (interactive) ZK scheme $\Sigma(\mathsf{ZK}) = \mathsf{ZK}.\{\mathsf{CRS}, P, V\}$ is <u>secure</u> for relation R if the following properties hold:

- Completeness: For any PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{r} crs \leftarrow \mathsf{CRS}(1^\lambda); (x,w) \leftarrow \mathcal{A}(crs); (\beta,\eta) \leftarrow [\![P(crs,x,w); V(crs,x)]\!]_{\mathrm{R}} \\ : (x,w) \in R \wedge \beta = 0 \end{array}\right] \overset{c}{\approx} 0.$$

- Soundness: For all PPT $\mathcal{A}$,

$$\Pr\left[crs \leftarrow \mathsf{CRS}(1^\lambda); x \leftarrow \mathcal{A}(crs); (\beta,\eta) \leftarrow [\![\mathcal{A}; V(crs,x)]\!]_{\mathrm{R}} : x \notin \mathcal{L}_R \wedge \beta = 1\right] \overset{c}{\approx} 0.$$

- Zero-Knowledge: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{P}$ such that $\Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{SimZK}_{\mathcal{A}}(\lambda,b)$ is defined as

$\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)$	$\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)$
$crs \leftarrow \mathsf{CRS}(1^\lambda)$;	$(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
$b' \leftarrow [\![P(crs,x,w); \mathcal{A}(crs)]\!]_{\mathrm{R}}$;	$b' \leftarrow [\![\widetilde{P}(crs,\tau,x); \mathcal{A}(crs)]\!]_{\mathrm{R}}$;
Return b'.	Return b'.

□

Definition 2.9.2 (Non-Erasure Zero-Knowledge ZKs). We say a secure ZK scheme $\Sigma(\mathsf{ZK}) = \mathsf{ZK}.\{\mathsf{CRS}, P, V\}$ is non-erasure zero-knowledge for relation R, if the following property holds:

- Non-Erasure Zero-Knowledge: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{P}, \widetilde{\mathsf{Eqiv}}$

such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)$
    $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs,x,w)$
    $(\alpha,\zeta) \leftarrow [\![P(crs,x,w);\mathcal{A}]\!]_{\mathrm{L}}$;
    Output $(\alpha,\zeta)$.

$\mathcal{O}_0(crs,\tau,x,w)$
    $(\alpha,\xi) \leftarrow [\![\widetilde{P}(crs,\tau,x);\mathcal{A}]\!]_{\mathrm{L}}$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(\xi,w)$;
    Output $(\alpha,\zeta)$.
```

□

Definition 2.9.3 (Knowledge-Extractable ZKs). We say a secure ZK scheme $\Sigma(\mathsf{ZK}) = \mathsf{ZK}.\{\mathsf{CRS}, P, V\}$ is knowledge-extractable for relation R, if the following property holds:

– KNOWLEDGE-EXTRACTION: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}$, $\widetilde{\mathsf{Ext}}$ such that

$$\Pr\left[\begin{array}{r} crs \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda); x \leftarrow \mathcal{A}(crs); (\beta,\eta) \leftarrow [\![\mathcal{A}; V(crs,x)]\!]_{\mathrm{R}}; \\ m \leftarrow \widetilde{\mathsf{Ext}}(crs,\tau,\eta) \,:\, (x,w) \notin R \wedge \beta = 1 \end{array}\right] \overset{c}{\approx} 0.$$

□

2.10 Sigma Protocols

In this subsection we introduce Sigma protocols, [CDS94]. Informally, a Sigma-protocol is a three move public randomness protocol (cf. Figure 2.7) with an honest verifier zero-knowledge property and a special soundness property. In our formulation of Sigma-protocols below we will formalize soundness only to satisfy membership consistency (rather than knowledge extraction).

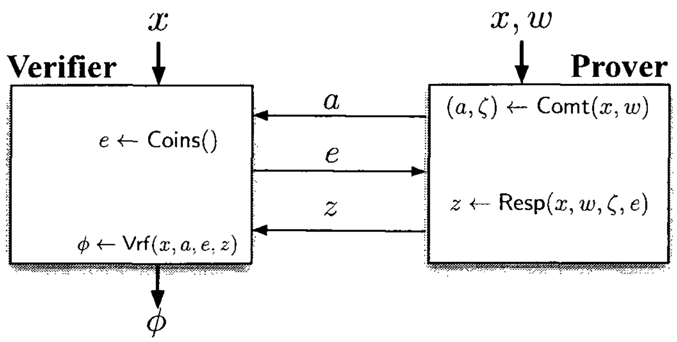

x
x, w
Verifier
Prover
$e \leftarrow \mathsf{Coins}()$
$\phi \leftarrow \mathsf{Vrf}(x, a, e, z)$
a
e
z
$(a, \zeta) \leftarrow \mathsf{Comt}(x, w)$
$z \leftarrow \mathsf{Resp}(x, w, \zeta, e)$
ϕ

Figure 2.7: Three-move public randomness protocol for relation R. Here $e \leftarrow \mathsf{Coins}()$ is implemented as $e \overset{\$}{\leftarrow} \{0,1\}^{\ell}$ where ℓ is the length of the challenge e.

Definition 2.10.1 (Secure Sigma Protocols). A Sigma protocol $\Sigma(\mathsf{3MZK}) = \mathsf{3MZK}.\{\mathsf{Comt}, \mathsf{Coins}, \mathsf{Resp}, \mathsf{Vrf}\}$ as demonstrated in Figure 2.7 is <u>secure</u> for relation R if the properties hold:

– COMPLETENESS: For any PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{r} (x,w) \leftarrow \mathcal{A}(1^\lambda); (a,\zeta) \leftarrow \mathsf{Comt}(x,w); e \leftarrow \mathsf{Coins}(); \qquad\qquad \\ z \leftarrow \mathsf{Resp}(x,w,\zeta,e); \phi \leftarrow \mathsf{Vrf}(x,a,e,z) \,:\, \phi = 0 \end{array}\right] \overset{c}{\approx} 0.$$

– SPECIAL MEMBERSHIP SOUNDNESS: For all PPT $\mathcal{A}$,

$$\Pr\left[\, (x,a) \leftarrow \mathcal{A}(1^\lambda); e \leftarrow \mathsf{Coins}(); z \leftarrow \mathcal{A}(e); \phi \leftarrow \mathsf{Vrf}(x,a,e,z) \,:\, \phi = 1 \wedge x \notin \mathcal{L}_R \,\right] \overset{c}{\approx} 0.$$

– HONEST-VERIFIER ZERO-KNOWLEDGE: For all PPT $\mathcal{A}$, there exist a PPT algorithm $\widetilde{\mathsf{Resp}}$

such that $\Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{SimZK}_{\mathcal{A}}(\lambda,b)$ is defined as

```
$\mathbf{SimZK}_{\mathcal{A}}(\lambda,1)$
    $(a,\zeta) \leftarrow \mathsf{Comt}(x,w)$;
    $e \leftarrow \mathsf{Coins}()$;
    $z \leftarrow \mathsf{Resp}(x,w,\zeta,e)$;
    $b' \leftarrow \mathcal{A}(x,w,a,e,z)$;
    Return $b'$.

$\mathbf{SimZK}_{\mathcal{A}}(\lambda,0)$
    $e \leftarrow \mathsf{Coins}()$;
    $(a,z,\xi) \leftarrow \widetilde{\mathsf{Resp}}(x,e)$;
    $b' \leftarrow \mathcal{A}(x,w,a,e,z)$;
    Return $b'$.
```

Definition 2.10.2 (Non-Erasure Zero-Knowledge Sigma Protocols). A secure Sigma protocol $\Sigma(\mathsf{3MZK}) = \mathsf{3MZK}.\{\mathsf{Comt}, \mathsf{Coins}, \mathsf{Resp}, \mathsf{Vrf}\}$ is non-erasure zero-knowledge for relation R if the following property also holds:

– NON-ERASURE HONEST-VERIFIER ZERO-KNOWLEDGE: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{Resp}}, \widetilde{\mathsf{Equiv}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)$
    $(a,\zeta) \leftarrow \mathsf{Comt}(x,w)$;
    $e \leftarrow \mathsf{Coins}()$;
    $z \leftarrow \mathsf{Resp}(x,w,\zeta,e)$;
    $b' \leftarrow \mathcal{A}(x,w,a,e,z,\zeta)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)$
    $e \leftarrow \mathsf{Coins}()$;
    $(a,z,\xi) \leftarrow \widetilde{\mathsf{Resp}}(x,e)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(\xi)$;
    $b' \leftarrow \mathcal{A}(x,w,a,e,z,\zeta)$;
    Return $b'$.
```

□

We note that the zero-knowledge formulated above is weak (honest verifier) and thus unsuitable for many settings. Nevertheless, there are generic methods of transforming an honest-verifier zero-knowledge protocol to one that satisfies zero-knowledge in more reasonable adversarial settings. For example, Damgård showed how one can use an equivocal commitment to extend a Sigma protocol to achieve concurrent zero-knowledge in the CRS model [Dam00]. Please refer to Section 2.9 of Nielsen's PhD thesis [Nie03] for a wonderful explanation of Sigma protocols.

2.11 Public Key Encryption

A public key encryption scheme is defined by a tuple (KG, Enc, Dec) consisting of the key-generation, encryption and decryption algorithms respectively.

Definition 2.11.1 (Secure Encryption Schemes). A public-key encryption scheme $\Sigma(\mathsf{PKE}) = \mathsf{PKE}.\{\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}\}$ is semantically secure if the following properties hold:

- COMPLETENESS: For all PPT $\mathcal{A}$,

$$\Pr\left[\ (pk, sk) \leftarrow \mathsf{KG}(1^\lambda); m \leftarrow \mathcal{A}(pk); (c, \zeta) \leftarrow \mathsf{Enc}(pk, m); m' \leftarrow \mathsf{Dec}(pk, sk, c) : m' = m\ \right] \overset{c}{\approx} 0.$$

- (INDISTINGUISHABLE) HIDING: *(Chosen-Plaintext Attack (CPA) Security)* For all PPT $\mathcal{A}$, $\Pr[\mathbf{IndHid}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{IndHid}_{\mathcal{A}}(\lambda, 1)]$, where $\mathbf{IndHid}_{\mathcal{A}}(\lambda, \cdot)$ is defined as below:

```
$\mathbf{IndHid}_{\mathcal{A}}(\lambda, b)$
    $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$;
    $(m_0, m_1) \leftarrow \mathcal{A}(crs)$;
    $(c_b, \zeta_b) \leftarrow \mathsf{Enc}(pk, m_b)$;
    Return $b' \leftarrow \mathcal{A}(c_b)$.
```

□

We can also define the hiding property in the simulation paradigm.

- (Simulatable) Hiding: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{KG}}$, $\widetilde{\mathsf{Enc}}$ such that $\Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{SimHid}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{SimHid}_{\mathcal{A}}(\lambda,1)$
    $(pk,sk) \leftarrow \mathsf{KG}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(pk,\cdot)}(pk)$;
    Return $b'$.

$\mathbf{SimHid}_{\mathcal{A}}(\lambda,0)$
    $(pk,sk) \leftarrow \widetilde{\mathsf{KG}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(pk,\cdot)}(pk)$;
    Return $b'$.

$\mathcal{O}_1(pk,m)$
    $(c,\zeta) \leftarrow \mathsf{Enc}(pk,m)$;
    Output $c$.

$\mathcal{O}_0(pk,m)$
    $(c,\xi) \leftarrow \widetilde{\mathsf{Enc}}(pk)$;
    Output $c$.
```

In [DN00], simulatable PKE is introduced to design non-committing encryption. For the definition of security, we also require the existence of an *oblivious public key generator* KG^* and a corresponding *key-faking algorithm* $\widetilde{\mathsf{KG}^*}$. Similarly, we require an *oblivious ciphertext generator* Enc^* and a corresponding *ciphertext-faking algorithm* $\widetilde{\mathsf{Enc}^*}$. Intuitively, the key-faking algorithm is used to explain a legitimately generated public key as an obliviously generated public key. Similarly, the ciphertext-faking algorithm is used to explain a legitimately generated ciphertext as an obliviously generated ciphertext.

Definition 2.11.2 (Simulatable PKE). We say a secure PKE scheme $\Sigma(\mathsf{PKE}) = \mathsf{PKE}.\{\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}\}$ is simulable if there exist PPT algorithms KG^*, $\widetilde{\mathsf{KG}^*}$, Enc^*, $\widetilde{\mathsf{Enc}^*}$ such that

– Key Obliviousness: For all PPT $\mathcal{A}$, $\Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,1)]$, where

$\mathbf{SimHid}_{\mathcal{A}}(\lambda, b)$ is defined as below

```
$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)$
    $(pk, \eta) \leftarrow \mathsf{KG}^*(1^\lambda)$;
    $b' \leftarrow \mathcal{A}(pk, \eta)$;
    Return $b'$.

$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)$
    $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$;
    $\eta \leftarrow \widetilde{\mathsf{KG}^*}(pk)$;
    $b' \leftarrow \mathcal{A}(pk, \eta)$;
    Return $b'$.
```

– Ciphertext Obliviousness: For all PPT $\mathcal{A}$, $\Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)]$, where $\mathbf{SimHid}_{\mathcal{A}}(\lambda, b)$ is defined as below

```
$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)$
    $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$;
    $(c, \zeta) \leftarrow \mathsf{Enc}^*(pk)$;
    $b' \leftarrow \mathcal{A}(pk, c, \zeta)$;
    Return $b'$.

$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)$
    $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$;
    $(c, \xi) \leftarrow \mathsf{Enc}(pk, m)$;
    $\zeta \leftarrow \widetilde{\mathsf{Enc}^*}(pk, c)$;
    $b' \leftarrow \mathcal{A}(pk, c, \zeta)$;
    Return $b'$.
```

□

Chapter 3
OBLIVIOUS TRANSFER

3.1 Introduction

When defining the security of cryptographic protocols, we generally strive to capture as wide a variety of adversarial attacks as possible. The most popular method of doing so is the *simulation paradigm* [GMW87] where the security of a real-world protocol is compared to that of an ideal-world (perfectly secure) implementation of the same task. Within the simulation paradigm there are several flavors. Firstly, basic simulation only guarantees security for single copy of a protocol executing in isolation. The *Universal Composability* (UC) framework [Can01, Can05] extends the simulation paradigm and defines security for protocols executed in arbitrary environments, where executions may be concurrent and even maliciously interleaved. Secondly, we generally distinguish between *static* and *adaptive* security. Static security protects against an adversary who controls some fixed set of corrupted parties throughout the computation. Adaptive security, on the other hand, defends against an adversary who can corrupt parties adaptively at any point during the course of the protocol execution (for example by bribing them or hacking into their machines). For adaptive security, we also make a distinction between the *erasure model,* where honest parties are trusted to securely erase data as mandated by the protocol, and the *non-erasure model,* where no such assumptions are made. Given the difficulty of erasing data securely it is valuable to construct protocols in the latter model, which is the subject of this work.

The seminal result of [CLOS02] shows that it is theoretically possible to design an adaptively secure and universally composable protocol for almost any task assuming the presence of some trusted setup such as a randomly selected common reference string (CRS). Unfortunately, the final protocol of [CLOS02] should be viewed as a *purely theoretical* construction. Its reliance on expensive Cook-Levin reductions precludes a practical implementation. Alternative efficient approaches to two-party and multi-party computation received a lot of attention in the recent works of [DN03, KO04, GMY04, JS07, LP07, IPS08, Lin09]. However, all of these results sacrifice some aspect of security to get efficiency. Concretely, the work of [LP07] only provides stand-alone static security, [JS07] provides UC static security, [GMY04, Lin09] provide UC/concurrent adaptive security but only in the erasure model, and [DN03] provides UC adaptive security but only for an honest majority, and [KO04] do not allow for an adversary that eventually corrupts *all* parties. The recent work of [IPS08] *can* provide UC adaptive security but only given an efficient adaptively secure Oblivious Transfer (OT) protocol. However, as we will discuss, no such protocols were known. Lastly, we mention the work of [CDD$^+$04], which gives a generic compiler from static to adaptive security using secure channels. Unfortunately, this compiler does not provide full adaptive security (does not allow for post-execution corruptions) and, as was noted in [Lin09], crucially relies on rewinding and hence cannot be used in the UC framework.

Indeed, thus far *no* efficient protocols for general multi-party computation, or even for many specific two-party function evaluation tasks, achieve adaptive security. This is not surprising given the difficulty of realizing adaptive security for even the most fundamental task in cryptography: *secure communication.* As was observed in [CFGN96], standard security notions for encryption do not suffice. Adaptively secure communication schemes, also called *non-committing encryption* schemes, were introduced and constructed in [CFGN96] and studied further in [Bea97, DN00], but these protocols are fairly complicated and inefficient for large messages.

It turns out that many useful two-party tasks (e.g., Oblivious Transfer, OR, XOR, AND, Millionaires' problem, etc.) are *strictly harder* to achieve than secure communication, the reason being that these tasks allow two honest parties to communicate by using the corresponding ideal functionality. For example, using Oblivious Transfer (OT), an honest sender can transfer a message to a receiver by setting it as *both* of his input values. Therefore, an adaptively secure OT protocol for the transfer of k bit messages can be used as a non-committing encryption of a k bit message and so all of the difficulty and inefficiency of non-committing encryption must **also** appear in protocols for tasks such as OT. Further, unlike secure communication, many tasks *also* require security against the active and malicious behavior of the *participants*. This might lead us to believe that the two difficulties will be compounded making efficient adaptively secure implementations of such tasks infeasible or too complicated to contemplate.

Taking Oblivious Transfer as an example, this indeed seems to be the case. The recent work of [LZ09], proves a (black-box) separation between enhanced trapdoor permutations (which allow for static OT) and adaptively secure OT, showing that the latter is indeed "more complex" in a theoretical sense. This complexity is reflected in practice as well. We are aware of only two examples (albeit inefficient) of adaptively secure OT protocols, from [Bea98] and [CLOS02]. Both of these works first construct an OT protocol for the honest-but-curious setting and then compile it into a fully-secure protocol using generic and inefficient zero knowledge proofs. In both constructions, the underlying honest-but-curious OT protocols rely on ideas from non-committing encryption[1] and hence inherit its complexity. Since the full constructions require us to run zero knowledge proofs *on top of* the complex underlying honest-but-curious protocol, there is little hope of making them efficient by only using proofs for simple relations. This is in contrast to static security (and adaptive security in the erasure model) for which we *have* recently seen efficient constructions of OT protocols. For example, [GMY04, JS07, DNO08] construct OT protocols by only using simple and efficient zero-knowledge proofs, while two very recent and efficient protocols [PVW08, Lin08] do not use zero-knowledge proofs at all! The protocol of [PVW08] is particularly exciting since it is a UC-secure protocol in the CRS model which runs in two rounds and uses a constant number of public key operations. Achieving adaptive security based on these protocols has, however, remained as an open problem.

In summary, we can use ideas from non-committing encryption to get honest-but-curious adaptively secure OT protocols, *or* we can use various clever ideas to achieve static security in the malicious setting, but there has been no known way to *combine* these techniques.

[1]The protocol of [Bea98] implicitly uses the plug-and-play approach from [Bea97], while the protocol of [CLOS02] uses non-committing encryption in a generic way.

3.1.1 Our Contributions

In this work we construct the first efficient (constant round, constant number of public-key operations) adaptively secure Oblivious Transfer protocol in the non-erasure model. Along the way we develop several techniques of independent interest which are applicable to adaptive security in general.

First, we introduce a new notion called *semi-adaptive* security which is slightly stronger than static security but significantly weaker than fully adaptive security. In particular, a semi-adaptively secure protocol for a task like OT, *does not* yield a non-committing encryption scheme and hence does not (necessarily) inherit its difficulty. We then give a generic compiler which transforms any semi-adaptively secure protocol into a (fully) adaptively secure protocol. The compiler is fairly simple: we take the original protocol and execute it over a secure communication channel (i.e., all communication from one party to another is sent over a secure channel). The compilation effectively decomposes the problem of adaptive security into two (simpler) problems which can be tackled separately: the problem of semi-adaptive security and the problem of realizing secure channels. We note that a similar compiler was studied in [CDD$^+$04]. As we mentioned, that compiler only works in the stand-alone setting and transforms a statically secure protocol into one which is adaptively secure *without* post-execution corruptions. In contrast, our protocol works in the UC setting, and results in fully adaptive security, but requires the starting protocol to be semi-adaptively secure (a new notion which we formally define later).

Unfortunately, we saw that the construction of secure-channels is a difficult problem and existing solutions are not very efficient. Also, as we already mentioned, we cannot completely bypass this problem since adaptive security for many tasks *implies* secure channels. However, for the sake of efficiency, we would like to limit the use of secure channels (and hence the use of non-committing encryption) to a minimum. For example, we know that an OT protocol for one-bit messages implies a non-committing encryption of a one-bit message. However, to get adaptive security for a bit-OT protocol, our compiler, as described above, would use non-committing encryption to encrypt the *entire* protocol transcript, and hence much more than one bit!

We fix this discrepancy by introducing a new notion called *somewhat non-committing encryption*. Somewhat non-committing encryption has two parameters: the equivocality ℓ (measuring just how *non-committing* the scheme is) and the message size k. We first observe that somewhat non-committing encryption is efficient for small values of the equivocality parameter ℓ, *even* when k is large (i.e., when we encrypt long messages). Secondly, we observe that our compiler can use somewhat non-committing encryption where the equivocality ℓ is proportional to the size of the input and output domains of the functionality. As a result, we obtain a very efficient compiler transforming any semi-adaptively secure protocol for a task with small input/output domains (such as bit-OT) into a fully adaptively secure protocol. We also show that this methodology can, in special cases, be applied to tasks with larger domain sizes such as string-OT with long strings.

We apply our methodology to the OT protocol of Peikert *et al.* [PVW08], resulting in the first efficient and adaptively secure OT protocols. Peikert *et al.* actually present a general framework for constructing static OT, and instantiate this framework using the Quadratic Residuocity (QR), Decisional Diffie-Hellman (DDH), and Lattice-based assumptions. In this work, we concentrate on the QR and DDH based schemes. We show that relatively small modifications suffice to make these schemes semi-adaptively secure. We then employ our compiler, using somewhat non-committing encryption, to convert them into (fully) adaptively UC-secure OT protocols.

3.1.2 Concurrent and Independent Work

The recent result of [CDMW09] gives a generic black-box compiler from semi-honest adaptively secure OT to fully malicious adaptively secure OT, using cut-and-choose techniques. Although the end result of our work is the same (adaptively secure OT), the two works take very different approaches which complement each other well: the compiler of [CDMW09] transforms semi-honest + adaptive security into malicious + adaptive security in the special case of OT, while our compiler is a general transformation from malicious + semi-adaptive security to malicious + adaptive security. The two starting notions of security (semi-honest + adaptive vs. malicious + semi-adaptive) are incomparable and thus both compilers are useful in different scenarios. In particular, our compiler can be used in conjunction with the OT protocol of [PVW08] and results in an extremely efficient adaptively-secure OT protocol using a constant number of rounds and $O(n)$ public-key operations to transfer an n-bit string. In contrast, the compiler of [CDMW09] shows how to base adaptively-secure OT on a simulatable cryptosystem in a black-box way, but at the expense of running $\Omega(\lambda^2)$ copies of the underlying semi-honest OT protocol, where λ is the security parameter, and thus requiring $\Omega(\lambda^2 n)$ operations for n-bit OT. Therefore our protocol can be significantly more efficient.

3.2 Building Block: Somewhat Non-Committing Encryption

What are some of the challenges in achieving adaptive security for a two-party protocol? Let's assume that a protocol π between two parties P_0, P_1 realizes a task $\mathcal{F}$ with respect to static adversaries. That means that there is a static simulator which can simulate the three basic cases: both parties are honest throughout the protocol, exactly one party is corrupted throughout the protocol or both parties are corrupted throughout the protocol. To handle adaptive adversaries, we require two more capabilities from our simulator: the ability to simulate a *first corruption* (i.e., the case that both parties start out honest and then one of them becomes corrupted) and simulating the *second corruption* (i.e., one party is already corrupted and the other party becomes corrupted as well).

Simulating the first corruption is often the harder of the two cases. The simulator must produce the internal state for the corrupted party in a manner that is consistent with the protocol transcript so far and with the actual inputs of that party (of which the simulator had no prior knowledge). Moreover, the simulator needs to have all the necessary trapdoors to continue the simulation *while* only one party is corrupted. Achieving both of these requirements at once is highly non-trivial and this is one of the reasons why efficient protocols for adaptively secure two-party computation have remained elusive.

Interestingly, simulating the first corruption becomes much easier if the protocol π employs secure channels for all communication between parties. At a high level, the simulator does not have to do any work while both parties are honest, since the real-world adversary does not see any relevant information during this time! When the first party *becomes* corrupted, we can just run a static simulation for the scenario in which this party *was* corrupted from the beginning but acting honestly and using its input. Then, we can "lie" and pretend that this communication (generated *ex post facto*) actually *took place* over the secure channel when both parties were honest. The lying is performed by setting the internal state of the corrupted party accordingly. Since our lie corresponds to the simulation of a statically corrupted party (which happens to act honestly), all of the trapdoors are in place to handle future mischievous behavior by that (freshly corrupted) party. The only problem left is in handling the second corruption – but this is significantly easier! To formalize this, we will define a notion of *semi-adaptive security* where the simulator needs to be able to simulate static corruptions as well as the case where one party starts out corrupted and the other party becomes corrupted later on (but *not* the case where *both* parties start out honest and may become corrupted later). The formal notion (with some additional restrictions imposed on the simulator) appears in Section 3.4.

Informally, we have argued that if two-party protocol is semi-adaptively secure, then the protocol is also fully adaptively secure if all communication between the parties is sent over an idealized secure channel. Unfortunately, idealized secure channels are hard to achieve physically and implementing such channels cryptographically in the real world requires the inefficient use of *non-committing encryption* [CFGN96] to encrypt the entire protocol transcript. Luckily, it turns out that we often do not need to employ fully non-committing encryption to make the above transformation hold. Indeed, we define a weaker primitive called *somewhat non-committing encryption* and show that this primitive can be implemented with significantly greater efficiency than (fully) non-committing encryption, and that it is often *good enough* to transform a semi-adaptively secure protocol into a fully adaptively secure protocol when the sizes of the input/output domains are small.

3.2.1 *Defining* Somewhat *Non-Committing Encryption*

First recall the notion of non-committing encryption from [CFGN96], which is a protocol used to realize secure channels in the presence of an *adaptive* adversary. In particular, this means that a simulator can produce "fake" ciphertexts and later explain them as encryptions of *any possible* given message. Several non-committing encryption schemes have appeared in literature [CFGN96, Bea97, DN00], but the main disadvantage of such schemes is the computational cost. All of the schemes are interactive (which was shown to be necessary in [Nie02]) and the most efficient schemes require $\Omega(1)$ public-key operations *per bit* of plaintext.

We notice that it is often unnecessary to require that the simulator can explain a ciphertext as the encryption of *any* later-specified plaintext. Instead, we define a new primitive, which we call *somewhat non-committing* encryption, where the simulator is given a set of ℓ messages during the generation of the fake ciphertext and must later be able to plausibly explain the ciphertext as the encryption of *any one of those ℓ messages.* In a sense, we distinguish between two parameters: the plaintext size (in bits) k and the equivocality ℓ (the number of messages that the simulator can plausibly explain). For fully non-committing encryption, the equivocality and the message size are related by $\ell = 2^k$. Somewhat non-committing encryption, on the other hand, is useful in accommodating the case where the equivocality ℓ is very small, but the message size k is large.

It is challenging to define an ideal-functionality for *somewhat* non-committing encryp-

Functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$

The ideal functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ interacts with adversary $\mathcal{S}$, and two parties including an initiator I and a receiver R. It consists of a channel-setup phase, after which the two parties can send arbitrarily many messages from one to another. The functionality is parameterized by a non-information oracle $\mathcal{N}$. For *all received messages, the functionality verifies that* $sid = (I, R, sid')$ for some sid', and ignores them otherwise.

Channel setup:

- Upon receiving ($\text{ChSetup}, I, sid$) from party I, initialize the machine $\mathcal{N}$ and pass this input to $\mathcal{N}$. Forward messages ($\text{LeakChSetup}, I, sid, \ldots$) from $\mathcal{N}$ to the adversary $\mathcal{S}$, and ($\text{InflChSetup}, R, sid, \ldots$) from the adversary $\mathcal{S}$ to $\mathcal{N}$. Once $\mathcal{N}$ outputs ($\text{ChSetupReturn}, R, sid$), output this output to party R, record the tuple $\langle sid, \mathcal{N}\rangle$. Ignore future ($\text{ChSetup}, \ldots$), and ($\text{InflChSetup}, \ldots$).

Message transfer:

- Upon receiving (Send, P, sid, m) from party P where $P \in \{I, R\}$, find a tuple $\langle sid, \mathcal{N}\rangle$ and, if none exists, ignore the input. Otherwise, invoke $\mathcal{N}$ with (Send, P, sid, m), and forward ($\text{LeakSend}, P, sid, \ldots$) and ($\text{InflSend}, \overline{P}, sid, \ldots$) messages between the oracle $\mathcal{N}$ and the adversary $\mathcal{S}$. Once $\mathcal{N}$ returns ($\text{SendReturn}, \overline{P}, sid, m$), output it to the *other* party $\overline{P} = \{I, R\} - \{P\}$.

Corruption:

- Upon receiving ($\text{Corrupt}, P, sid$) from the adversary $\mathcal{S}$, send ($\text{Corrupt}, P, sid$) to $\mathcal{N}$ and forward its returned information to $\mathcal{S}$. After the first corruption, stop the execution of $\mathcal{N}$ and give $\mathcal{S}$ complete control over the functionality (i.e., $\mathcal{S}$ learns all inputs and can specify any outputs).

Figure 3.1: The parameterized secure-channel ideal functionality, $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$.

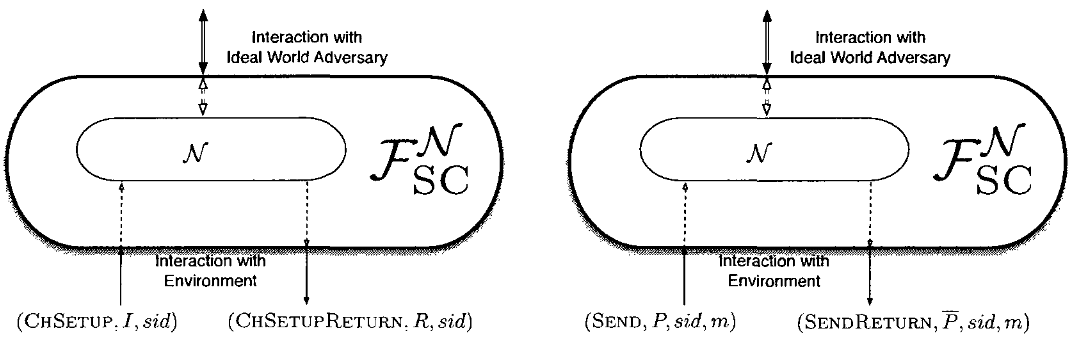

Interaction with Ideal World Adversary
$\mathcal{N}$
$\mathcal{F}_{\text{SC}}^{\mathcal{N}}$
Interaction with Environment
(ChSetup, I, sid)
(ChSetupReturn, R, sid)
Interaction with Ideal World Adversary
$\mathcal{N}$
$\mathcal{F}_{\text{SC}}^{\mathcal{N}}$
Interaction with Environment
(Send, P, sid, m)
(SendReturn, $\overline{P}$, sid, m)

Figure 3.2: Pictorial version of the ideal functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$. The left subfigure is for action ChSetup, and the right one for action Send where $P \in \{I, R\}$.

tion, since the ideal world captures a notion of security which is too strong. Here, we take the approach of [CK02] where ideal-world functionalities are weakened by the inclusion of a *non-information oracle* which is a PPT TM that captures the information leaked to the adversary in the ideal world. The ideal functionality for secure channels, given in Figure 3.1 (also refer to Figure 3.2), is parameterized using a non-information oracle $\mathcal{N}$ which gets the values of the exchanged messages m and outputs some side information to the adversary $\mathcal{S}$. The security of the secure channel functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ depends on the security properties required for the machine $\mathcal{N}$ and thus we can capture several meaningful notions. Let us first start with the most secure option which captures (fully) non-committing encryption.

Definition 3.2.1. Let $\mathcal{N}^{\text{full}}$ be the oracle, which, on input (Send, P, sid, m), produces the output (LeakSend, $P, sid, |m|$) and, on any inputs corresponding to the ChSetup, Corrupt commands, produces no output. We call the functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}^{\text{full}}}$, or just $\mathcal{F}_{\text{SC}}$ for brevity, a *(fully) non-committing secure channel.* A real-world protocol which realizes $\mathcal{F}_{\text{SC}}$ is called a *non-committing encryption scheme (NCE).*

Above, the oracle $\mathcal{N}$ never reveals anything about messages m exchanged by two honest parties, even if (both of the) parties later get corrupted. Hence the functionality is fully *non-committing.* To define *somewhat* non-committing encryption we first give the following definitions of non-information oracles.

Definition 3.2.2. A machine $\mathcal{R}$ is called a *message-ignoring* oracle if, on any input $(\textsc{Send}, P, sid, m)$, it ignores the value m and processes only the input $(\textsc{Send}, P, sid, |m|)$. A machine $\mathcal{M}$ called a *message-processing* oracle if it has no such restrictions. We call a pair of machines $(\mathcal{M}, \mathcal{R})$ *well-matched* if no PPT distinguisher $\mathcal{D}$ (with oracle access to either $\mathcal{M}$ or $\mathcal{R}$) can distinguish the message-processing oracle $\mathcal{M}$ from the message-ignoring oracle $\mathcal{R}$.

We are now ready to define the non-information oracle used by a somewhat non-committing secure channel ideal functionality. Please also refer to Figure 3.3.

Definition 3.2.3. Let $(\mathcal{M}, \mathcal{R})$ be a well-matched pair which consists of a message-processing and a message-ignoring oracle respectively. Let $\mathcal{N} = \mathcal{N}^{\ell}$ be a (stateful) oracle with the following structure.

- Upon initialization, $\mathcal{N}^{\ell}$ chooses a uniformly random index $i \overset{\$}{\leftarrow} \{1, \ldots, \ell\}$. In addition it initializes a tuple of ℓ *independent* TMs: $\langle \mathcal{N}_1, \ldots, \mathcal{N}_{\ell} \rangle$ where $\mathcal{N}_i = \mathcal{M}$ and, for $j \neq i$, the machines $\mathcal{N}_j$ are independent copies of the message-ignoring oracle $\mathcal{R}$.
- Whenever $\mathcal{N}^{\ell}$ receives inputs of the form $(\textsc{ChSetup}, P, sid)$ or $(\textsc{Send}, P, sid, m)$, it passes the input to each machine $\mathcal{N}_i$ receiving an output y_i. It then outputs the vector $(y_1, \ldots, y_{\ell})$.

– Upon receiving an input (Corrupt, P, sid), the oracle reveals the internal state of the message-processing oracle $\mathcal{N}_i$ only.

For any such oracle $\mathcal{N}^\ell$, we call the functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}^\ell}$ an *ℓ-equivocal non-committing secure channel*. For brevity, we will also use the notation $\mathcal{F}_{\text{SC}}^\ell$ to denote $\mathcal{F}_{\text{SC}}^{\mathcal{N}^\ell}$ for some such oracle $\mathcal{N}^\ell$. Lastly, a real world protocol which realizes $\mathcal{F}_{\text{SC}}^\ell$ is called an *ℓ-equivocal non-committing encryption scheme (ℓ-NCE).* □

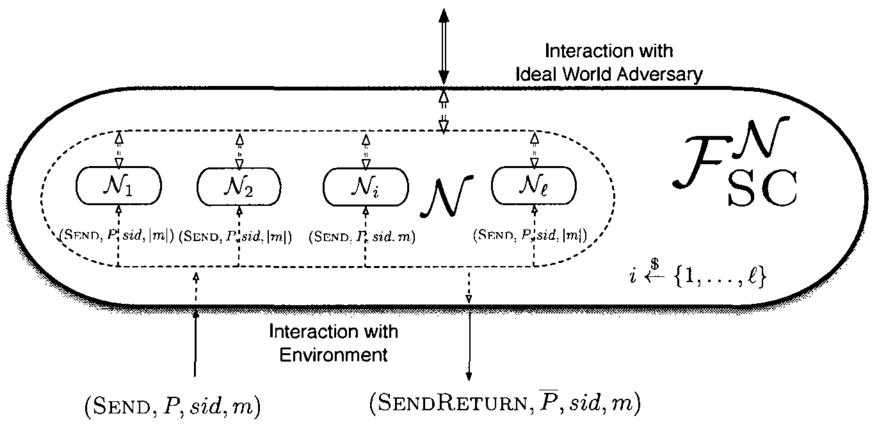

Interaction with
Ideal World Adversary
$\mathcal{F}_{\text{SC}}^{\mathcal{N}}$
$\mathcal{N}_1$
$\mathcal{N}_2$
$\mathcal{N}_i$
$\mathcal{N}$
$\mathcal{N}_\ell$
(Send, P, sid, $|m|$)
(Send, P, sid, $|m|$)
(Send, P, sid, m)
(Send, P, sid, $|m|$)
$i \xleftarrow{\$} \{1, \ldots, \ell\}$
Interaction with
Environment
(Send, P, sid, m)
(SendReturn, $\overline{P}$, sid, m)

Figure 3.3: Pictorial version of the ideal functionality $\mathcal{F}_{\text{SC}}^\ell$ for action Send where $P \in \{I, R\}$. Note that $\mathcal{N} = \langle \mathcal{N}_1, \ldots, \mathcal{N}_\ell \rangle$, where $\mathcal{N}_i \leftarrow \mathcal{M}$ for $i \xleftarrow{\$} \{1, \ldots, \ell\}$ while $\mathcal{N}_j \leftarrow \mathcal{R}$ for $j \in \{1, \ldots, \ell\} \setminus \{i\}$.

As before, no information about messages m is revealed during the "send" stage. However, the internal state of the message-processing oracle $\mathcal{N}_i$, which is revealed upon corruption, might be "committing." Nevertheless, a simulator can simulate the communication between two honest parties over a secure channel, as modeled by $\mathcal{F}_{\text{SC}}^\ell$, in a way that allows him to later explain this communication as any one of ℓ possibilities. In particular, the simulator creates ℓ message-processing oracles and, for every Send command, the simulator chooses ℓ distinct messages $m_1, \ldots, m_\ell$ that he passes to the oracles $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ respectively. Since message-processing and message-ignoring oracles are indistinguishable, this looks indistinguishable from the side information produced by $\mathcal{F}_{\mathrm{SC}}^{\ell}$. Later, when a corruption occurs, the simulator can convincingly explain the entire transcript of communication to any one of the ℓ possible options, by providing the internal state of the appropriate message-processing oracle $\mathcal{M}_i$.

3.2.2 *Construction for ℓ-NCE*

The construction of ℓ-NCE is based on a *simulatable public-key system* [DN00], wherein it is possible to generate public keys obliviously, without knowing the corresponding secret key, and to explain an honestly (non-obliviously) generated public key as one which was obliviously generated. In a similar way, there should be a method for obliviously generating ciphertexts (without knowing any plaintext) and to explain honestly generated (non-oblivious) ciphertexts as obliviously generated ones. Refer to the full version for review of the syntax and security properties of such a scheme. Our ℓ-NCE protocol construction, shown in Figure 3.4, uses a fully non-committing secure channel, but only to send a *very short* message during the setup phase. In addition, it uses a simulatable public-key system and a symmetric key encryption scheme where ciphertexts are indistinguishable from uniformly random values (the latter can be constructed from any one way function). For very long communications and small ℓ, our ℓ-NCE scheme is significantly more efficient than (full) NCE.

Protocol π^{ℓ}_{SC}

Let (KG, Enc, Dec) be a *simulatable public-key system* and KG*, Enc* be the corresponding *oblivious key generator* and *oblivious ciphertext generator* algorithms. Further, let ($\mathsf{KG}^{\mathsf{sym}}, \mathsf{Enc}^{\mathsf{sym}}, \mathsf{Dec}^{\mathsf{sym}}$) be a symmetric-key encryption scheme in which ciphertexts are indistinguishable from uniformly random values of the same length.

Channel Setup: An initiator I sets up a channel with a receiver R as follows:

- The initiator I sends a random index $i \in \{1,\ldots,\ell\}$ to the receiver R over a fully non-committing secure channel.
- The initiator I generates ℓ public keys. For $j \in \{1,\ldots,\ell\} \setminus \{i\}$, the keys $(pk_j, \eta_j) \leftarrow \mathsf{KG}^*(1^\lambda)$ are sampled obliviously, while $(pk_i, sk_i) \leftarrow \mathsf{KG}(1^\lambda)$ is sampled correctly. The keys $pk_1,\ldots,pk_\ell$ are sent to the receiver R while the initiator I stores sk_i.
- The receiver R chooses a random key $K \leftarrow \mathsf{KG}^{\mathsf{sym}}$ and computes $(C_i, \xi_i) \leftarrow \mathsf{Enc}(pk_i, K)$ correctly. In addition, R samples $(C_j, \zeta_j) \leftarrow \mathsf{Enc}^*(pk_j)$ obliviously for $j \in \{1,\ldots,\ell\} \setminus \{i\}$ and sends the ciphertexts $C_1,\ldots,C_\ell$ to the initiator I.
- The initiator I decrypts the key $K \leftarrow \mathsf{Dec}(pk_i, sk_i, C_i)$. Both parties store $\langle K, i \rangle$.

Encryption: An initiator I encrypts a message m to a receiver R as follows:

- The initiator I computes $E_i \leftarrow \mathsf{Enc}^{\mathsf{sym}}_K(m)$ and chooses E_j for $j \in \{1,\ldots,\ell\} \setminus \{i\}$ as uniformly random and independent values of length $|E_i|$. The tuple $(E_1,\ldots,E_\ell)$ is sent to the receiver R.
- The receiver R ignores all values other than E_i. It computes $m \leftarrow \mathsf{Dec}^{\mathsf{sym}}_K(E_i)$.

Figure 3.4: The ℓ-NCE protocol π^{ℓ}_{SC}.

Theorem 3.2.4. *The protocol π^{ℓ}_{SC} in Figure 3.4 is an ℓ-NCE scheme. Specifically, it UC-realizes functionality $\mathcal{F}^{\ell}_{\mathrm{SC}}$ in the presence of an active and adaptive adversary.*

Let us look at the efficiency of the construction. For concreteness we will assume that the secure channel used to encrypt the index i are implemented using the NCE protocol of [DN00]. The simulatable public-key system (which we use during channel setup and also which is used in the protocol of [DN00]) can be instantiated with e.g. ElGamal Encryption. Lastly, the symmetric key system can be implemented very efficiently using some variable length encryption algorithm e.g. AES in CBC mode.

The protocol of [DN00] is an *expected* 6 round protocol which exchanges a t bit message using an *expected* $\mathcal{O}(t)$ number of operations.[2] Since we use the NCE protocol to send an index $i \in \{1,\ldots,\ell\}$, this requires $\mathcal{O}(\log \ell)$ number of public key operations. In addition to NCE, we use the simulatable public key system. However, we only use it to perform one encryption/decryption operation and $\mathcal{O}(\ell)$ oblivious key-sampling, ciphertext-sampling operations. For ElGamal, this just means choosing random group elements which is efficient and hence we do not count it as a public key operation. Lastly, for the encryption phase we only use symmetric key operations. This gives us a total of (expected) $\mathcal{O}(\log \ell)$ public key operations, $\mathcal{O}(\ell)$ communication and fewer than (expected) 8 rounds of interaction for the channel setup phase. After channel-setup, encryption is non-interactive and requires only symmetric key operations. However, the encryption of a k bit message requires $\mathcal{O}(\ell k)$ communication.

[2]For large messages (larger than the security parameter) the protocol of [DN00] can be made *guaranteed* 3 round. However, the tradeoff is that the number of public key operations is at least security parameter. Since we consider very small messages (3 bits) we settle for the *expected* 6 round protocol.

3.3 Building Block: Semi-Adaptively Secure Protocols

As an application of ℓ-NCE, we now give a general theorem showing that a protocol with semi-adaptive security can be compiled into a protocol with (full) adaptive security when all of the communication is encrypted using ℓ-NCE for some appropriate ℓ. However, we must first give a formal definition of semi-adaptive security.

Definition 3.3.1. An adversarial strategy is second-corruption adaptive if *either* at least one of the parties is corrupted prior to protocol execution *or* no party is ever corrupted. In the former case, the other party can be adaptively corrupted at any point during or after protocol execution.

Intuitively, we would like to say that a protocol is semi-adaptively secure if it is secure with respect to second-corruption adaptive strategies. Unfortunately, there are two subtleties that we must consider. Firstly, we know that most tasks cannot be realized in the Universal Composability framework without the use of *trusted setup*. However, the use of trusted setup complicates our transformation. The point of using (somewhat) non-committing encryption is that the simulator can lie about *anything that occurs while both parties are honest*. However, we often rely on trusted setup in which some information is given to the adversary even when both parties are honest. For example, the usual modeling of a common reference string specifies that this string is made public and given to the adversary even when *none* of the participants in the protocol are corrupted. In this case the simulator is committed to such setup even if the parties communicate over secure channels. Therefore we require that, when trusted setup is used, the semi-adaptive simulator simulates this setup independently of which party is corrupted. We call this property *setup-adaptive simulation.*

The second subtlety comes from the following type of problem. As we outlined in our informal discussion, we wish to run the semi-adaptive simulator once the first party gets corrupted and then "lie" that the simulated conversation took place over the secure channel. However, when the first party gets corrupted after the protocol execution, then the ideal functionality has already computed the outputs using the honest inputs and will therefore not accept anymore inputs from the semi-adaptive simulator. Recall that we run the semi-adaptive simulator with respect to an adversary $\mathcal{A}$ which follows the protocol execution using the corrupted party's honest input x. If the semi-adaptive simulator extracts the same input x as the one used by $\mathcal{A}$, then we also know the corresponding output and can give it to the semi-adaptive simulator on behalf of the ideal functionality. Therefore it is crucial that the semi-adaptive simulator can only submit the actual input x. We call this property *input-preserving*. Putting Definition 3.3.1 and the above notions together, we are finally ready to define semi-adaptive security.

Definition 3.3.2. We say that a protocol π semi-adaptively realizes the ideal functionality $\mathcal{F}$ if there exists a setup-adaptive and input-preserving PPT simulator $\mathcal{S}$ such that, for any PPT adversary $\mathcal{A}$ and environment $\mathcal{Z}$ which follow a second-corruption adaptive adversarial strategy, we have $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Lastly, we define the notion of a *well-structured* protocol. Since even non-committing encryption *commits* the simulator to the lengths of the exchanged messages, the number of such messages, and the identities of the sender and receiver of each message, we require that this information is fixed and always the same any given execution of a protocol. Almost all known constructed protocols for cryptographic tasks are well-structured and any protocol can be easily converted into a well-structured protocol.

3.4 *Compiler: Adaptively Secure Two-Party SFE*

Functionality of two party secure function evaluation for a function $f : X_I \times X_R \longrightarrow Y_I \times Y_R$ can be defined as in Figure 3.5.

Functionality $\mathcal{F}_{\text{SFE}}^{f}$

$\mathcal{F}_{\text{SFE}}$, parameterized with a function $f : X_I \times X_R \rightarrow Y_I \times Y_R$, interacts with adversary $\mathcal{S}$, and two parties an initiator I and a responder R. For each input or message, the functionality verifies that $sid = (I, R, sid')$ for some sid', and ignores it otherwise.

Evaluation:

- Upon receiving the input value $(\textsc{Evaluate}, P, sid, x_P)$ from party P, where $P \in \{I, R\}$, record the value $\langle P, x_P\rangle$ and send the message $(\textsc{LeakEvaluate}, P, sid)$ to $\mathcal{S}$. Ignore future $(\textsc{Evaluate}, P, \ldots)$ inputs.
- Upon receiving the message $(\textsc{InflEvaluate}, P, sid)$ from $\mathcal{S}$ where $P \in \{I, R\}$, if either $\langle I, x_I\rangle$ or $\langle R, x_R\rangle$ is not recorded, ignore the message. Else if $\langle y_I, y_R\rangle$ is not recorded, then compute $(y_I, y_R) \leftarrow f(x_I, x_R)$ and record $\langle y_I, y_R\rangle$; send the output value $(\textsc{EvaluateReturn}, P, sid, y_P)$ to party P. Ignore future $(\textsc{Evaluate}, P, \ldots)$ messages from the adversary.

Figure 3.5: Two-party secure evaluation functionality $\mathcal{F}_{\text{SFE}}^{f}$.

First we look at the simple compiler using idealized secure channels.

Theorem 3.4.1. *Let $\mathcal{F}_{\text{SFE}}^{f}$ be the two-party ideal functionality which computes some function f as defined in Figure 3.5. Assume that a well-structured two-party protocol π for $\mathcal{F}_{\text{SFE}}^{f}$ is* *semi-adaptively secure. Let π' be the protocol in which the parties run π but only communicate with each other using non-committing secure channels as modeled by $\mathcal{F}_{\mathrm{SC}}$. Then π' is (fully) adaptively secure.*

As we already mentioned, this compiler is usually not very efficient because of its excessive use of secure channels and hence NCE. Recall that secure channels are employed so that, when both parties are honest, the adversary does not see any useful information and so this case is easy to simulate. Then, when the first party gets corrupted, our simulator simply makes up the transcript of the communication that should have taken place *ex post facto*. This transcript is generated based on which party got corrupted, what its inputs were and what its outputs were. However, we notice that for many simple protocols there are not too many choices for this information. The simulator must simply be able to credibly lie that the communication which took place over the secure channel corresponds to any one of these possible choices. Using this intuition, we show that a more efficient compiler using ℓ-NCE (for some small ℓ) suffices.

Theorem 3.4.2. *Let $\mathcal{F}^{f}_{\mathrm{SFE}}$ be the two-party ideal functionality computing some function $f : X_I \times X_R \to Y_I \times Y_R$, as defined in Figure 3.5. Assume that a well-structured two-party protocol π for $\mathcal{F}^{f}_{\mathrm{SFE}}$ is semi-adaptively secure. Let π' be the protocol in which the parties run π but only communicate with each other using ℓ-equivocal secure channels as modeled by $\mathcal{F}^{\ell}_{\mathrm{SC}}$ where $\ell = |X_I||Y_I| + |X_R||Y_R|$. Then π' is (fully) adaptively secure.*

3.5 PVW Compiler for Statically Secure OT

We start by giving an ideal functionality for OT, following the modeling of [Can05].

Functionality $\mathcal{F}_{\mathrm{OT}}$

$\mathcal{F}_{\mathrm{OT}}$ interacts with adversary $\mathcal{S}$, and two parties a sender S and a receiver R. For each input or message, the functionality verifies that $sid = (R, S, sid')$ for some sid', and ignores it otherwise.

Transfer:

- Upon receiving an input $(\textsc{Transfer}, S, sid, \langle x_0, x_1 \rangle)$ from party S, where each $x_i \in \{0,1\}$, record $\langle x_0, x_1 \rangle$ and send $(\textsc{LeakTransfer}, S, sid)$ to $\mathcal{S}$. Ignore further $(\textsc{Transfer}, S, \ldots)$.
- Upon receiving an input $(\textsc{Transfer}, R, sid, \sigma)$ from party R, where $\sigma \in \{0,1\}$, record σ and send $(\textsc{LeakTransfer}, R, sid)$ to $\mathcal{S}$. Ignore further $(\textsc{Transfer}, R, \ldots)$.
- Upon receiving a message $(\textsc{InflTransfer}, R, sid)$ from $\mathcal{S}$, if either x_0, x_1 or σ is not recorded, ignore the message. Else send output $(\textsc{TransferReturn}, R, sid, x_\sigma)$ to party R. Ignore further $(\textsc{InflTransfer}, R, \ldots)$ messages from $\mathcal{S}$.

Figure 3.6: Oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$.

In [PVW08], Peikert *et al.* construct an efficient OT protocol in the CRS model with UC security against a malicious but static adversary. They do so by introducing a new primitive called a *dual-mode cryptosystem*, which almost immediately yields an OT protocol in the CRS model, and give constructions of this primitive under the DDH, QR and lattice hardness assumptions. We therefore first present a brief (informal and high-level) review of dual-mode encryption as in [PVW08], and then will formally define a modified version of this primitive which will allow us to get adaptive security.

A dual-mode cryptosystem is initialized with *system parameters* which are generated by a trusted third party. For any choice of system parameters, the cryptosystem has two types of public/private key pairs: *left* key pairs and *right* key pairs. The key-generation algorithm can sample either type of key pair and the user specifies which type is desired. Similarly, the encryption algorithm can generate a *left* encryption or a *right* encryption of a message. When the key pair type matches the encryption type (i.e. a left encryption of a message under a left public key) then the decryption algorithm (which uses the matching secret key) correctly recovers the message.

Definition 3.5.1 (Dual-Mode Encryption). A dual-mode cryptosystem for message space $\{0,1\}^n$ is defined by the following polynomial-time algorithms:

- $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$. The parameter generation algorithm PG is a randomized algorithm which takes security parameter λ and mode $\mu \in \{0,1\}$ as input, and outputs (crs, τ), where *crs* is a common reference string and τ is the corresponding trapdoor information. For notational convenience, the random coins used for parameter generation are also included in τ.
- $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$. The key generation algorithm KG is a randomized algorithm which takes *crs* and a key type $\sigma \in \{0,1\}$, and outputs a key pair (pk, sk), where *pk* is an encryption key and *sk* the corresponding decryption key for message encrypted on key type σ. The random coins used for key generation are also included in *sk*.
- $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, b, m)$. The encryption algorithm Enc is a randomized algorithm which outputs a ciphertext c for a message m on encryption type b. Here ζ is the random coins used for encryption.
- $m \leftarrow \mathsf{Dec}(crs, pk, sk, c)$. The decryption algorithm Dec is a deterministic algorithm which decrypts ciphertext c into plaintext m.

□

Definition 3.5.2 (Secure Dual-Mode Encryption). A dual-mode cryptosystem is <u>secure</u> if the following properties hold:

- Completeness: For all PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} \mu,\sigma \leftarrow \mathcal{A}(1^\lambda); (crs,\tau) \leftarrow \mathsf{PG}(1^\lambda,\mu); m \leftarrow \mathcal{A}(crs); (pk,sk) \leftarrow \mathsf{KG}(crs,\sigma); \\ (c,\zeta) \leftarrow \mathsf{Enc}(crs,pk,\sigma,m); m' \leftarrow \mathsf{Dec}(crs,pk,sk,c) \;:\; m' \neq m \end{array}\right] \overset{c}{\approx} 0$$

- Mode Indistinguishability: For all PPT $\mathcal{A}$, $\Pr[\mathbf{IndMode}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{IndMode}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{IndMode}_{\mathcal{A}}(\lambda,\cdot)$ is defined as below:

```
$\mathbf{IndMode}_{\mathcal{A}}(\lambda,b)$
$(crs_b,\tau_b) \leftarrow \mathsf{PG}(1^\lambda,b);$
Return $b' \leftarrow \mathcal{A}(crs_b)$.
```

- Messy Branch Identification: For all PPT $\mathcal{A}$, there exists a PPT algorithm $\widetilde{\mathsf{Idf}}$ computing the "messy" branch σ' such that

$$\Pr\left[\begin{array}{r} (crs,\tau) \leftarrow \mathsf{PG}(1^\lambda,0); (pk,sk,\sigma) \leftarrow \mathcal{A}(crs); \phi \leftarrow \mathsf{KG}(crs,\sigma,pk,sk); \\ \sigma' \leftarrow \widetilde{\mathsf{Idf}}(crs,\tau,pk) \;:\; \phi = 1 \wedge \sigma = \sigma' \end{array}\right] \overset{c}{\approx} 0.$$

- (Messy Branch) Ciphertext Statistical Hiding: For all PPT $\mathcal{A}$, there exist a PPT algorithm $\widetilde{\mathsf{Enc}}$ such that $\Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,0)] \overset{s}{\approx} \Pr[\mathbf{SimHid}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{SimHid}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 1)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, 0)$;
    $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,pk,\cdot)}(crs, pk)$;
    Return $b'$.

$\mathbf{SimHid}_{\mathcal{A}}(\lambda, 0)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, 0)$;
    $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,pk,\cdot)}(crs, pk)$;
    Return $b'$.

$\mathcal{O}_1(crs, pk, m)$
    $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, 1 - \sigma, m)$;
    Output $c$.

$\mathcal{O}_0(crs, \tau, pk, m)$
    $(c, \xi) \leftarrow \widetilde{\mathsf{Enc}}(crs, \tau, pk)$;
    Output $c$.
```

– ENCRYPTION KEY DUALITY: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{KG}}, \widetilde{\mathsf{EquivKey}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda, 0)] \overset{s}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda, 1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda, b)$ is defined as below

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda, 1)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, 1)$;
    $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$;
    $b' \leftarrow \mathcal{A}(crs, pk, sk)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda, 0)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, 0)$;
    $(pk, \xi) \leftarrow \widetilde{\mathsf{KG}}(crs)$;
    $sk \leftarrow \widetilde{\mathsf{EquivKey}}(pk, \xi, \sigma)$;
    $b' \leftarrow \mathcal{A}(crs, pk, sk)$;
    Return $b'$.
```

□

As shown in [PVW08], a dual-mode cryptosystem can be used to get an OT protocol as shown in Figure 3.7. The receiver chooses to generate a left or right key depending on his input bit σ, and the sender uses left-encryption ($b = 0$) for the left message x_0 and right-

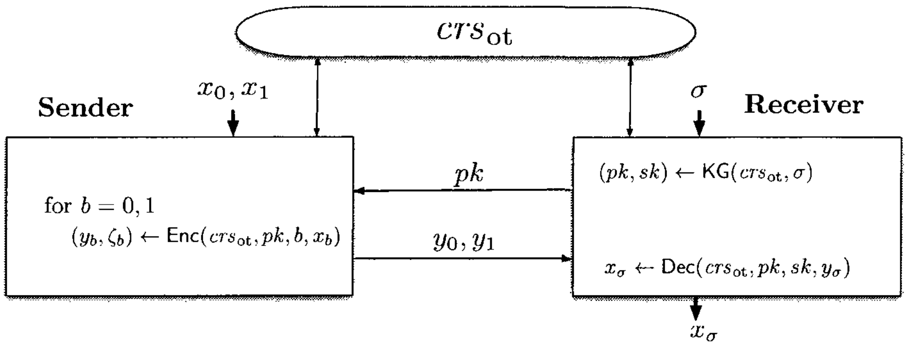

crs_{ot}
Sender
x_0, x_1
σ
Receiver
for $b = 0, 1$
$(y_b, \zeta_b) \leftarrow \mathsf{Enc}(crs_{\text{ot}}, pk, b, x_b)$
pk
y_0, y_1
$(pk, sk) \leftarrow \mathsf{KG}(crs_{\text{ot}}, \sigma)$
$x_\sigma \leftarrow \mathsf{Dec}(crs_{\text{ot}}, pk, sk, y_\sigma)$
x_σ

Figure 3.7: The generic OT protocol in [PVW08].

encryption for the right message. The receiver then uses the secret key to correctly decrypt the chosen message.

Security against malicious (static) adversaries in the UC model relies on the two different modes for generating the system parameters: *messy* mode and *decryption* mode. In *messy mode*, the system parameters are generated together with a *messy trapdoor*. Using this trapdoor, any public key (even one which is maliciously generated) can be easily labeled a left key or a right key. Moreover, in messy mode, when the encryption type does not match the key type (e.g. a left encryption using a right public key) then the ciphertext is statistically independent of the message. Messy mode is useful to guarantee security against a *corrupt receiver:* the messy trapdoor makes it easy to extract the receiver bit and to create a fake ciphertext for the message which should not be transferred. On the other hand, in *decryption mode*, the system parameters are generated together with a *decryption trapdoor* which can be used to decrypt both left and right ciphertexts. Moreover, in decryption mode, left public keys are statistically indistinguishable from right public keys.

Decryption mode is useful to guarantee security against a *corrupt sender:* the decryption trapdoor is used to create a public key which completely hides the receiver's selection bit, and to compute a decryption trapdoor and extracting both of the sender's messages. In each mode, the security of one party (i.e., the sender in messy mode, and the receiver in decryption mode) is guaranteed information theoretically. To achieve security for both parties simultaneously all that is needed is one simple computational requirement: the system parameters generated in messy mode need to be computationally indistinguishable from those generated in decryption mode.

3.6 Semi-Adaptively Secure OT

In order to make the PVW OT protocol adaptively secure using our methodology, we need to make it semi-adaptively secure (section 3.4). We do so by a series of simple transformations.

First, we observe that in the PVW protocol, the simulator must choose the CRS crs_{ot} based on which party is corrupt – i.e. the CRS should be in messy mode to handle a corrupt receiver or in decryption mode to handle a corrupt sender. This is a problem for us since the definition of semi-adaptive security requires that the simulator is setup-adaptive which means that it must simulate the CRS independently of any information on which parties are corrupted. We solve this issue by using a coin-tossing protocol to choose the CRS of the PVW OT protocol. Of course, coin-tossing requires the use of a UC secure commitment scheme which also needs their own CRS (crs_{com})! However, if we use an (efficient) adaptively secure commitment scheme (e.g. [DN02, DG03]) then the simulator's choice of

crs_{com} can be independent on which party is corrupted. Unfortunately, this approach only works if the CRS for the OT protocol comes from a uniform distribution (over some group) and this too is not the case in all instantiations of the PVW protocol. However, we observe that the CRS of the OT protocol (crs_{ot}) can be divided into two parts $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$, where a *system CRS* crs_{sys} can be independent of which party is corrupted (i.e. can be the same for both messy and decryption mode) but may not be uniform, while crs_{tmp} determines the mode (messy or decryption) and thus needs to depend on which party is corrupted, but this part is required to be uniform. Therefore we can use an ideal CRS functionality to choose the setup for our protocol which consists of $(crs_{\mathrm{com}}, crs_{\mathrm{sys}})$ and then run a coin-flipping protocol to choose the uniform value crs_{tmp}.

Secondly, we must now consider the cases where one party is corrupted from the beginning, but the second party becomes corrupted adaptively during the protocol execution. Let us first consider the case where the sender starts out corrupted. In this case, to handle the corrupt sender, the simulator needs to simulate the execution in decryption mode. Moreover, to extract the sender's value, the simulator uses the decryption trapdoor to create a *dual public key* (on behalf of the receiver) which comes with both a left and a right secret key. Later, if the receiver becomes corrupted, the simulator needs to explain the randomness used by the receiver during key generation to create such a public key. Luckily, current dual-mode schemes already make this possible and we just update the definition with a property called *encryption key duality* to capture this. Now, consider the case where the receiver is corrupted at the beginning but the sender might *also* become corrupted later on. In this case the simulator simulates the execution in messy mode. In particular, the simulator uses the messy trapdoor to identify the receiver key type (right or left) and thus extract the receiver bit. Then the simulator learns the appropriate sender message for that bit and (honestly) produces the ciphertext for that message. In addition, the simulator must produce a "fake" ciphertext for the other message. Since, in messy mode, this other ciphertext is statistically independent of the message, it is easy to do so. However, if the sender gets corrupted later, the simulator must *explain* the fake ciphertext as an encryption of some particular message. To capture this ability, we require the existence of *internal state reconstruction* algorithm which can explain the fake ciphertext as an encryption of any message. Again, we notice that the QR instantiation of the PVW scheme already satisfies this new notion as well.

As explained above, in the enhanced version of DME, our parameter generation PG includes two stages $\mathsf{PG}_{\mathsf{sys}}$ and $\mathsf{PG}_{\mathsf{tmp}}$, i.e., compute $(G, crs_{\mathsf{sys}}, \tau_{\mathsf{sys}}) \leftarrow \mathsf{PG}_{\mathsf{sys}}(1^\lambda)$ and $(crs_{\mathsf{tmp}}, \tau_{\mathsf{tmp}}) \leftarrow \mathsf{PG}_{\mathsf{tmp}}(\mu, G, crs_{\mathsf{sys}}, \tau_{\mathsf{sys}})$ where G is a group with operator "+", and set $crs \leftarrow (crs_{\mathsf{sys}}, crs_{\mathsf{tmp}})$ and $\tau \leftarrow (\tau_{\mathsf{sys}}, \tau_{\mathsf{tmp}})$. Note that the system CRS is independent of mode μ.

Definition 3.6.1 (Enhanced DME). We say a secure DME is an enhanced scheme if the following properties hold

– Enhanced Mode Indistinguishability: For all PPT $\mathcal{A}$,

$$\Pr[\mathbf{IndMode}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{IndMode}_{\mathcal{A}}(\lambda, 2)] \overset{c}{\approx} \Pr[\mathbf{IndMode}_{\mathcal{A}}(\lambda, 1)]$$

where $\mathbf{IndMode}_{\mathcal{A}}(\lambda,\cdot)$ is defined as below:

```
$\mathbf{IndMode}_{\mathcal{A}}(\lambda, b)$
    $(G, crs, \tau) \leftarrow \mathsf{PG}_{\mathsf{sys}}(1^{\lambda})$;
    If $b \in \{0,1\}$ then $(crs_b, \tau_b) \leftarrow \mathsf{PG}_{\mathsf{tmp}}(b, G, crs, \tau)$
    else $b = 2$ then $crs_b \overset{\$}{\leftarrow} G$;
    Return $b' \leftarrow \mathcal{A}(G, crs, crs_b)$.
```

– (Messy Branch) Ciphertext Statistical Equivocality: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Eqiv}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)] \overset{s}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{Equiv}_{\mathcal{A}}(\lambda,b)$ is defined as below

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda, 1)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^{\lambda}, 0)$;
    $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs, m)$
    $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, 1-\sigma, m)$;
    Output $(c, \zeta)$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda, 0)$
    $(crs, \tau) \leftarrow \mathsf{PG}(1^{\lambda}, 0)$;
    $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_0(crs, \tau, m)$
    $(c, \xi) \leftarrow \widetilde{\mathsf{Enc}}(crs, \tau)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(c, \xi, m)$;
    Output $(c, \zeta)$.
```

□

3.6.1 Construction

Based on the above transformations, a generic construction for a semi-adaptively secure OT protocol is given in Figure 3.8. It consists of two phases, the coin tossing phase and the transferring phase (which is separated by a dot line in the figure). The CRS consists of two pieces: the first piece is a system CRS denoted as crs_{sys}, while the second piece is for an adaptively secure UC commitment protocol which will be used for constructing a coin tossing protocol. The UC commitment includes two stages, the commit and the open stages which could be interactive; a randomly selected value r is committed by the receiver for the sender in the commit stage, and after receiving a randomly selected value s from the sender, the receiver open the committed r to the sender, and both sender and the receiver can compute a temporal CRS crs_{tmp} based on s and r. The temporal CRS crs_{tmp} together with the system CRS crs_{sys} will be used as the CRS for the transferring phase and we denote it as crs_{ot}. With crs_{ot} in hand, we "plug in" the PVW protocol (Figure 3.7) but based on the enhanced dual-mode cryptosystem to achieve message transferring.

Theorem 3.6.2. *Given an adaptively UC-secure commitment scheme and an enhanced dual-mode cryptosystem as in Definition 3.5.2, the protocol in Figure 3.8 semi-adaptively realizes $\mathcal{F}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model.*

3.7 QR-based Concrete Construction for Bit OT

3.7.1 QR-based Enhanced Dual Mode Encryption

We first review the dual mode encryption based on Quadratic Residuosity (QR) assumption in [PVW08], then we show this scheme is actually an *enhanced* dual mode encryption

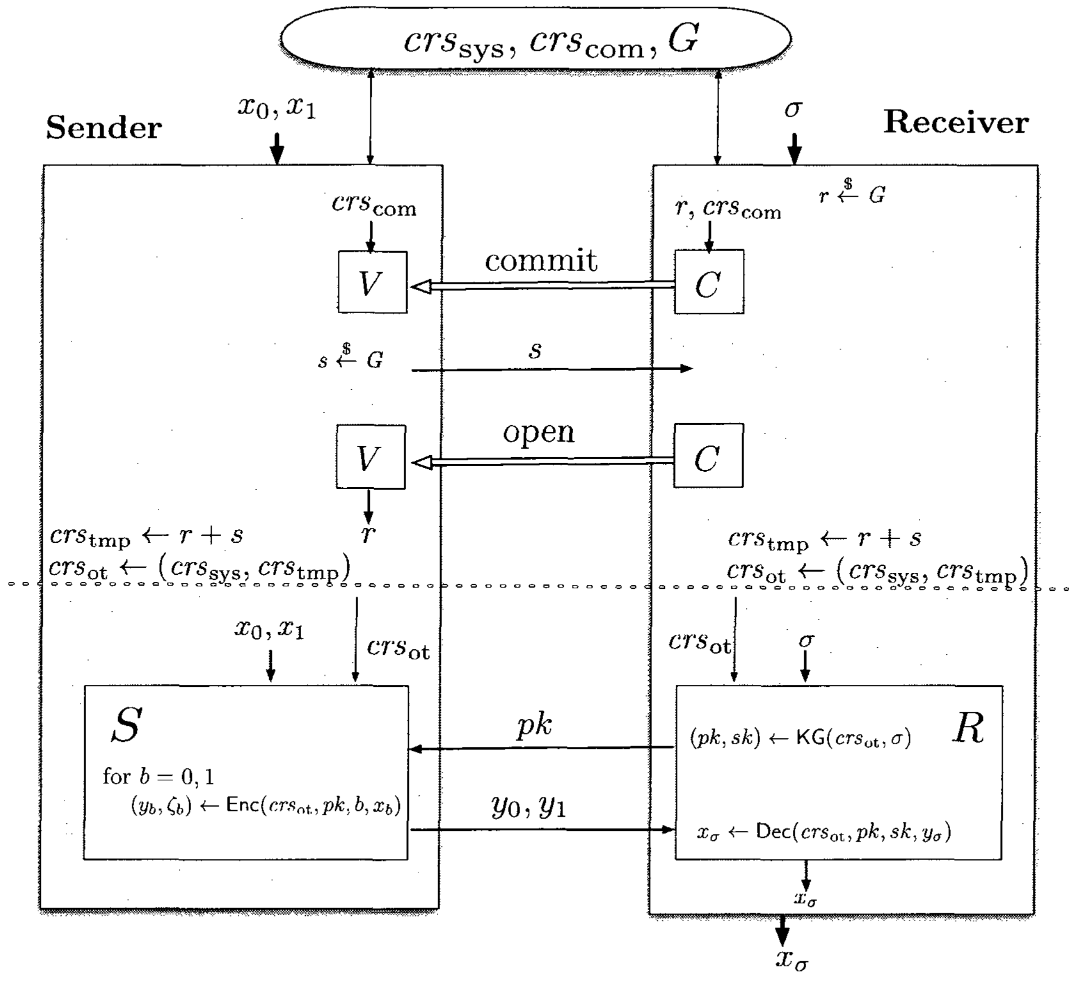

$crs_{\text{sys}}, crs_{\text{com}}, G$
Sender
x_0, x_1
σ
Receiver
crs_{com}
r, crs_{com}
$r \xleftarrow{\$} G$
V
commit
C
$s \xleftarrow{\$} G$
s
V
open
C
r
$crs_{\text{tmp}} \leftarrow r + s$
$crs_{\text{ot}} \leftarrow (crs_{\text{sys}}, crs_{\text{tmp}})$
$crs_{\text{tmp}} \leftarrow r + s$
$crs_{\text{ot}} \leftarrow (crs_{\text{sys}}, crs_{\text{tmp}})$
x_0, x_1
crs_{ot}
crs_{ot}
σ
S
for $b = 0, 1$
$(y_b, \zeta_b) \leftarrow \text{Enc}(crs_{\text{ot}}, pk, b, x_b)$
pk
y_0, y_1
$(pk, sk) \leftarrow \text{KG}(crs_{\text{ot}}, \sigma)$
R
$x_\sigma \leftarrow \text{Dec}(crs_{\text{ot}}, pk, sk, y_\sigma)$
x_σ
x_σ

Figure 3.8: Generic semi-adaptively secure OT protocol. Here $\boxed{C} \xrightarrow{\text{commit}} \boxed{V}$ and $\boxed{C} \xrightarrow{\text{open}} \boxed{V}$ denote the commit and the open stages of an adaptive UC-secure commitment protocol based on CRS crs_{com}. crs_{sys} is the system CRS, and $\boxed{S} \leftrightarrows \boxed{R}$ is the PVW protocol (Figure 3.7), but based on our enhanced dual-mode encryption scheme.

as defined in Definition 3.5.2 under the same assumption.

The QR-based dual mode encryption in [PVW08] is based on a variant of Cocks' encryption scheme [Coc01] as follows. For $N \in \mathbb{N}$, let $\mathbb{J}_N$ denote the set of all $x \in \mathbb{Z}_N$ with Jacobi symbol +1, and $\mathbb{QR}_N \subset \mathbb{J}_N$ denote the set of all quadratic residues in $\mathbb{Z}_N^*$, and $(\frac{t}{N})$ denote the Jacobi symbol of t in $\mathbb{Z}_N^*$. The message space is $\{\pm 1\}$.

- The key generation algorithm $(pk, sk) \leftarrow \mathsf{CocKG}(1^\lambda)$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$ and set $y \leftarrow r^2$. Set $pk \leftarrow (N, y)$, and $sk \leftarrow r$. Output (pk, sk).
- The encryption algorithm $(c, s) \leftarrow \mathsf{CocEnc}(pk, m)$: Parse pk as (N, y). Randomly select $s \xleftarrow{\$} \mathbb{Z}_N^*$ such that $(\frac{s}{N}) = m$, and compute $c \leftarrow s + y/s$. Output (c, s).
- The decryption algorithm $m \leftarrow \mathsf{CocDec}(sk, c)$: Parse sk as r. Compute the Jacobi symbol of $c + 2r$, i.e., $m \leftarrow (\frac{c+2r}{N})$. Output m.

We next present the enhanced dual mode cryptosystem which is based on the above scheme.

- The parameter generation algorithm $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$ for the messy mode and the decryption mode are presented as follows where $\mu \in \{0, 1\}$, $crs = (\mathbb{J}_N, crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$, and $\tau = (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$.
  - ⋆ $(\mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}}) \leftarrow \mathsf{PG}_{\mathrm{sys}}(1^\lambda)$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Set $crs_{\mathrm{sys}} \leftarrow N$ and $\tau_{\mathrm{sys}} \leftarrow (p, q)$. Output $(\mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$.
  - ⋆ $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(0, \mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$: Randomly select $y \xleftarrow{\$} \mathbb{J}_N \backslash \mathbb{QR}_N$. Set $crs_{\mathrm{tmp}} \leftarrow y$ and $\tau \leftarrow \emptyset$. Output $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}})$.

$\star$ $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(1, \mathbb{J}_N, crs_{\text{sys}}, \tau_{\text{sys}})$: Randomly select $s \xleftarrow{\$} \mathbb{Z}_N^*$ and set $y \leftarrow s^2 \bmod N$. Set $crs_{\text{tmp}} \leftarrow y$ and $\tau_{\text{tmp}} \leftarrow s$. Output $(crs_{\text{tmp}}, \tau_{\text{tmp}})$.

– The key generation algorithm $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$:

Parse the crs as $(\mathbb{J}_N, N, y)$. Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$, set $sk \leftarrow r$ and $pk \leftarrow r^2/y^\sigma$. Output (pk, sk).

– The encryption algorithm $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, b, m)$:

Parse the crs as $(\mathbb{J}_N, N, y)$. Set $pk_b \leftarrow (N, pk \cdot y^b)$. Compute $(c, \zeta) \leftarrow \mathsf{CocEnc}(pk_b, m)$. Output (c, ζ).

– The decryption algorithm $m \leftarrow \mathsf{Dec}(crs, pk, sk, c)$:

Parse the crs as $(\mathbb{J}_N, N, y)$. Compute $m \leftarrow \mathsf{CocDec}(sk, c)$. Output m.

– The messy branch identification algorithm $\rho \leftarrow \widetilde{\mathsf{Idf}}(crs, \tau, pk)$:

Parse the crs as $(\mathbb{J}_N, N, y)$ where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. If $pk \in \mathbb{QR}_N$, then set $\rho \leftarrow 1$; otherwise set $\rho \leftarrow 0$. Output ρ.

– The fake encryption algorithm $(c, \omega) \leftarrow \widetilde{\mathsf{Enc}}(crs, \tau, pk, \rho)$:

Parse the crs as $(\mathbb{J}_N, N, y)$ where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. Set $y' \leftarrow pk \cdot y^\rho$ and set $pk_\rho \leftarrow (N, y')$; note that $y' \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Compute $(c, \zeta_0) \leftarrow \mathsf{CocEnc}(pk_\rho, m)$, i.e., and compute $c \leftarrow \zeta_0 + y'/\zeta_0$. Based on the messy characterization explored in Lemma 6.1 in [PVW08], compute $\zeta_1, \zeta_2, \zeta_3$ such that $\zeta_1 = \zeta_0 \bmod p$ and $\zeta_1 = y'/\zeta_0 \bmod q$, $\zeta_2 = y'/\zeta_0 \bmod p$ and $\zeta_2 = \zeta_0 \bmod q$, $\zeta_3 = y'/\zeta_0 \bmod p$ and $\zeta_3 = y'/\zeta_0 \bmod q$; note that two of $(\frac{\zeta_i}{N})$ are $+1$, and the other two are -1. Set $\omega \leftarrow (\zeta_0, \zeta_1, \zeta_2, \zeta_3)$. Output (c, ω).

– The internal state reconstruction algorithm $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(crs, \tau, pk, \rho, c, \omega, m)$:

Parse the *crs* as $(\mathbb{J}_N, N, y)$ where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p,q) where $N = pq$. Set $\zeta \leftarrow \zeta_i$ where ζ_i is from ω such that $(\frac{\zeta_i}{N}) = m$. Output ζ.

– $(pk, sk_0, sk_1) \leftarrow \widetilde{\mathsf{EquivKey}}(crs, \tau)$:

Parse *crs* as $(\mathbb{J}_N, N, y)$ and the trapdoor τ as s where $y = s^2 \bmod N$ and $N = pq$. Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$, set $sk_b \leftarrow r \cdot s^b$ for $b \in \{0,1\}$, and $pk \leftarrow r^2$. Output (pk, sk_0, sk_1).

Theorem 3.7.1. *Under the quadratic residuosity assumption, the above scheme is an enhanced dual mode encryption as defined in Definition 3.5.2.*

Proof sketch. The proof is very similar to that of Theorem 6.2 in [PVW08]. The first two properties can be argued directly based on their proof. The fourth property can also be argued based on their proof because in the key generation, the randomness used by the KeyGen is exactly the decryption information which is included in decryption key *sk*. For the third property, besides their argument, we need further to show for all $m \in \{\pm 1\}$, the distribution of (c, ζ) from the honest encryption algorithm is identical to that produced by the fake encryption and reconstruction algorithms.

For $m = +1$, the former distribution can be written as $\{(c,\zeta) | \zeta \xleftarrow{\$} \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c,\zeta_0) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c,\zeta_0) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0\}$; so the two distribution are identical.

For $m = -1$, the former distribution can be written as $\{(c,\zeta) | \zeta \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c,\zeta_1) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c,\zeta_1) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$; note that $(\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N}) = -1 \cdot (\frac{\zeta_0}{N})$ since $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$;

given $\zeta_0 \xleftarrow{\$} \mathbb{J}_N$, we can have $\zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N$; therefore the latter distribution can be rewritten as $\{(c, \zeta_1) | \zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$, and then $\{(c, \zeta_1) | \zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_1 + y/\zeta_1\}$ which is identical to the former distribution.

Together we show the two distributions are identical, which concludes that the third property is also satisfied. This completes the proof. □

3.7.2 *Bit OT*

We now apply our compiler from section 3.4 to the protocol in Figure 3.8, to immediately obtain an efficient adaptively secure OT protocol in the UC framework.

Corollary 3.7.2. *Assume that the DDH, QR, and DCR assumptions hold. Then there exists an adaptively secure protocol that UC-realizes the* bit-OT *functionality $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid world, running in (expected) constant number of rounds and using (expected) constant number of public-key operations.*

Justification for the assumptions is as follows: efficient adaptive UC commitments can be realized in the CRS model under the DCR assumption [DN02], non-committing and somewhat non-committing encryption can be constructed under DDH ([DN00] and Section 3.2, respectively), while enhanced dual-model encryption exists under the QR assumption ([PVW08] and Section 3.6).

3.8 DDH-based Concrete Construction for Bit and String OT

Here we give a high-level description of adaptively-secure OT constructions from the DDH-based protocol in [PVW08]. We first review the original DDH-based OT construction from PVW. The we show how to use it to construct adaptively secure bit OT and finally String OT.

3.8.1 *Dual Mode Encryption Based on DDH*

We assume that the parties agree to work in the DDH group $\mathbb{G}$ of order p where p is chosen based on the security parameter.

- The common random string consists of four group elements $crs = (g_0, h_0, g_1, h_1)$.
  - ⋆ $\mathsf{PG}(1^\lambda, 0)$. In *messy mode* the four group elements are uniformly and independently distributed. They are generated by choosing g_0, g_1 randomly in $\mathbb{G}$, (x_0, x_1) randomly in $\mathbb{Z}_p$ and setting $h_0 = g^{x_0}, h_1 = g^{x_1}$. The messy trapdoor is $t = (x_0, x_1)$. We output $\langle crs = (g_0, h_0, g_1, h_1), t\rangle$.
  - ⋆ $\mathsf{PG}(1^\lambda, 1)$ In *decryption mode* the four group elements form a DDH tuple: g_0 is chosen randomly in $\mathbb{G}$ and two values x, y are chosen randomly in $\mathbb{Z}_p$. We set $g_1 = g_0^y, h_0 = g_0^x, h_1 = g_1^x = g_0^{xy}$. The decryption trapdoor is $t = y$. We output $\langle crs = (g_0, h_0, g_1, h_1), t\rangle$.

  The main difference between messy mode and decryption mode is that in decryption mode, $\mathsf{Dlog}_{g_0}(h_0) = \mathsf{Dlog}_{g_1}(h_1)$ while in messy mode this happens with negligible probability.
- The key generation algorithm $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$:

  Given a bit $\sigma \in \{0,1\}$, choose $r \leftarrow \mathbb{Z}_p$. Let $g = g_\sigma^r, h = h_\sigma^r$. Set $pk = (g, h)$, $sk = r$. Output (pk, sk).
- The encryption algorithm $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, b, m)$:

Given a public key $pk = (g,h)$, a message m and a bit b, choose $s,t \leftarrow \mathbb{Z}_p$ and set $u := g_b^s h_b^t$ $v := g^s h^t$. Set the ciphertext $c \leftarrow (u, v \cdot m)$, and $\zeta \leftarrow (s,t)$. Output (c,b).

- The decryption algorithm $m \leftarrow \mathsf{Dec}(crs, pk, sk, c)$:

  Given a secret key $sk = r$ with label σ and a ciphertext $c = (c_0, c_1)$ with label $b = \sigma$, output $m' = c_0^r / c_1$.

- The messy branch identification algorithm $e \leftarrow \widetilde{\mathsf{Idf}}(crs, \tau, pk)$:

  Given the messy trapdoor $t = (x_0, x_1)$ and a public key $pk = (g,h)$ test for $h = g_0^x$. If the test passes, label the key as a left-key by outputting $e = 0$. Otherwise label it as a right key by outputting $e = 1$.

- $(pk, sk_0, sk_1) \leftarrow \widetilde{\mathsf{EquivKey}}(crs, \tau)$:

  Given the decryption trapdoor $\tau = y$, we can compute a key pair (pk, sk) which can be used to decrypt *both* left and right decryptions. To do so compute $(g,h) = (g_0^{r_0}, h_0^{r_0})$, where r_0 is chosen randomly, and then compute $r_1 = r/y$ so that $(g,h) = (g_1^{r_1}, h_1^{r_1})$. Output $(pk, sk = (r_0, r_1))$. Later, we can call the Dec procedure with secret keys r_0 or r_1 to decrypt both left and right ciphertexts.

3.8.2 Bit OT

There is one main problem in the above scheme when it comes to adaptive security. When the key and the encryption type do not match in messy mode (i.e. a left encryption under a right key) the encryption is statistically hiding. However, for adaptive security, the simulator also needs to be able to generate a valid looking ciphertext of this form and later be able to explain it as a valid encryption of any specified message. Unfortunately, in the current scheme, this might not be possible since the encryption can be binding even for a simulator who knows the messy trapdoor.

To see this in detail, recall that the receiver can choose an arbitrary public key g, h. If $h \neq g^{x_0}$, then left encryption is statistically hiding since $u = g_0^s h_0^t = g_0^{s+tx_0}, v = g^s h^t = g_0^{r(s+tx_1')}$ for some $r, x_1' \neq x_0$ and hence u, v are statistically independent. However, for any left ciphertext (c_0, c_1), if the simulator can explain the ciphertext as an encryption of either one of two different messages $m \neq m'$, then the simulator can compute s, t, s', t' such that $g^s h^t m = c_1 = g^{s'} h^{t'} m'$. This presents two problems. Firstly, the simulator must know $\mathsf{Dlog}_g(h)$ but g, h can be chosen arbitrarily by the receiver. Secondly, the simulator must know $\mathsf{Dlog}_g(m)$ but m is also chosen arbitrarily without input from the simulator.

The second problem is the easier of the two to solve - we simply shrink the message space of the encryption scheme to 1 bit: to encrypt a bit $e \in \{0, 1\}$ we simply use the original scheme on the message $m = g^e \in \{1, g\}$. This way, the simulator knows $\mathsf{Dlog}_g(m)$ for both eligible messages.

The first problem is a little trickier, but we notice that the honest receiver always chooses g, h such that $\mathsf{Dlog}_g(h) \in \{x_0, x_1\}$. Somehow we need to disallow the cheating receiver from choosing arbitrary g, h. To do so, we want the receiver to prove that either $\mathsf{Dlog}_g(h) = \mathsf{Dlog}_{g_0}(h_0)$ or $\mathsf{Dlog}_g(h) = \mathsf{Dlog}_{g_1}(h_1)$. Actually, we can make the receiver prove an equivalent statement that either $\mathsf{Dlog}_{g_0}(g) = \mathsf{Dlog}_{h_0}(h)$ or $\mathsf{Dlog}_{g_1}(g) = \mathsf{Dlog}_{h_1}(h)$. We take advantage of the fact that there are efficient Σ protocols for the equality of discrete logs. Actually we need the Sigma protocol to be *non-erasure* in the sense that if the receiver, after generating the transcript with the sender, is corrupted, the simulator can "explain" such Sigma protocol transcript. Further the Sigma protocol is required to defend against a dishonest verifier; we can use a trick by Damgaard [Dam00] to transform a Sigma protocol against honest verifier into one against dishonest verifier by using an equivocal commitment to "hide" the first move of the Sigma protocol until receiving the challenge from the verifier. Please refer to [Nie03, Section 2.9] and [Nie05] for more details. The protocol therefore becomes:

- Let $crs = (g_0, g_1, h_0, h_1)$.
- The receiver chooses a key $pk = (g, h), sk$ using the key generation algorithm on his bit σ. In addition the receive computes the first message a of a Σ-protocol.
- The sender sends a challenge c.
- The receiver sends a response z.
- The sender verifies that (a, c, z) is accepting and, if so, computes a left encryption of $m_0 = g^{e_0}$ for the left message $e_0 \in \{0, 1\}$ and a right encryption of $m_1 = g^{e_1}$ for the right message $e_0 \in \{0, 1\}$ under pk.

The above yields a semi-adaptively secure bit-OT protocol. We now apply our compiler from Section 3.4, and use somewhat non-committing encryption to protect the protocol transcripts to obtain a very efficient adaptively secure bit OT based on the DDH assumption. We further remark that short (logarithmic size) string OT can be similarly obtained.

3.8.3 String OT

The above approach for semi-adaptively secure bit OT based on DDH can be *efficiently* extended to obtain semi-adaptively secure string OT, for larger strings of n bits. There

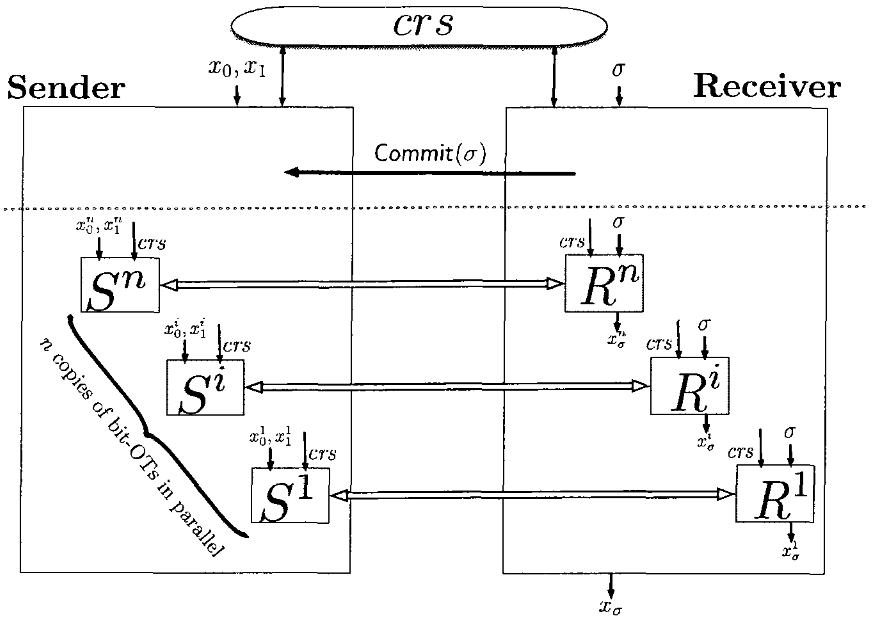


Figure 3.9: DDH-based fully adaptively-secure String-OT protocol. The protocol consists of two stages. In the first stage, the receiver sends an equivocal commitment on its input σ; for the purpose of giving a concrete example, we here instantiate the equivocal commitment with Peterson commitment $c = \text{Commit}(\sigma) = g_\Delta^\sigma h_\Delta^\gamma$. In the second stage, the sender and the receiver run n copies of fully adaptively-secure bit-OTs explored in Section G.1 but with the relation updated in the Sigma protocol, i.e., in i-th bit-OT, $\boxed{S^i} \Longleftrightarrow \boxed{R^i}$, the relation is updated into $\mathsf{R}^i = \{(c, pk^i, crs_{\text{ot}}^i), (r^i, \gamma) | (g^i = (g_0^i)^{(r^i)} \wedge h^i = (h_0^i)^{(r^i)} \wedge c = h_\Delta^\gamma) \vee (g^i = (g_1^i)^{(r^i)} \wedge h^i = (h_1^i)^{(r^i)} \wedge c = g_\Delta h_\Delta^\gamma)\}$ where $crs_{\text{ot}}^i = \langle g_0^i, h_0^i, g_1^i, h_1^i, \mathbb{G}\rangle$ is obtained by S^i and R^i after coin tossing phase, and pk^i is in the first move in transferring phase. Further $\mathbb{G} = (G, p, g_*)$ is a cyclic group with prime order p and generator g_*, and $g_0^i, h_0^i, g_1^i, h_1^i, g_\Delta, h_\Delta \in G$.

are to issues that we must face in order to convert the bit-OT scheme into a string-OT scheme. Firstly, we cannot simply increase the message space to all of $\mathbb{Z}_p$ since that would not allow for efficient equivocation. Secondly, we cannot use our standard somewhat non-committing compilation since such a compiler is only efficient for small domains/ranges. Here we take a detour; we simply "bundle" n copies of bit OT into a string OT for n bits. This can be done as follows. First the receiver computes an equivocal commitment of his bit σ. Then, the sender and the receiver run n copies of bit-OTs in parallel, and the Sigma protocol guarantees not only that each "public key" pk chosen by the receiver is well-formed, but also that all public keys are based on same bit σ, which matches the commitment. Using Pedersen commitments, we have efficient Σ-protocols for this language.

This already gets us an adaptive OT protocol assuming secure channels. However, we now show how to efficiently use somewhat non-committing encryption to improve computational efficiency. The idea is that each of the seperate bit-OT protocols is executed over its own "ℓ-equivocal channel" (essentially a fresh an independent channel for SNCE). It is easy to show that this still allows the simulator to simulate the "initial corruption". Moreover, we notice that it is very efficient to set-up n separate ℓ-SNCE channels using (strict) $O(n)$ public key operations and strict constant number of rounds.[3] This results in the parameters mentioned in Theorem 3.8.1 below.

Theorem 3.8.1. *Assume that the DDH and DCR assumptions hold. Then there exists an adaptively secure protocol that UC-realizes the* string-OT *functionality $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid*

[3]Essentially, we need to use fully non-committing encryption to send n indices in the range $[1,\ell]$. We can do so with the above efficiency assuming $n = \Omega(\lambda)$. Recall that previously, for NCE of a single bit, we did not wish to sacrifice $O(\lambda)$ operations and thus were forced to use *expected* constant number of rounds/operations.

world, and can transfer an n-bit string in (strict) constant number of rounds and using (strict) $O(n)$ public-key operations.

3.9 Delayed Proofs

3.9.1 Proof of Theorem 3.2.4

We need to show that our protocol in Figure 3.4 realizes the $\mathcal{F}_{\mathrm{SC}}^{\ell}$ functionality described in Definition 3.2.3. This functionality is parametrized by two oracles: the message-ignoring oracle $\mathcal{R}$ and the message-processing oracle $\mathcal{M}$. We define these oracles based on our actual protocol construction. In essence the oracle $\mathcal{M}$ corresponds to the actions of the parties for the index i chosen during channel setup, while the oracle $\mathcal{R}$ corresponds to all other indices in $\{1,\ldots,\ell\}$. In particular:

- On input $(\textsc{ChSetup}, I, sid)$, the oracle $\mathcal{M}$ samples $(pk,sk) \leftarrow \mathsf{KG}(1^\lambda)$, $K \leftarrow \mathsf{KG}^{\mathsf{sym}}()$, $(C,\xi) \leftarrow \mathsf{Enc}(pk,K)$. The oracle $\mathcal{R}$ samples $(pk,\zeta) \leftarrow \mathsf{KG}^*(1^\lambda)$, $(C,\eta) \leftarrow \mathsf{Enc}^*(pk)$.
- On input $(\textsc{Send}, I, sid, m)$, the oracle $\mathcal{M}$ samples $E \leftarrow \mathsf{Enc}_K^{\mathsf{sym}}(m)$. The oracle $\mathcal{R}$ samples E randomly.

We must first show that the oracles $\mathcal{M}$ and $\mathcal{R}$ are indistinguishable. We do so using a hybrid argument.

1. We start with the message-processing oracle $\mathcal{M}$.
2. We now modify the oracle so that, for all $\textsc{Send}$ commands, it chooses the symmetric-key ciphertext E randomly instead of computing $E \leftarrow \mathsf{Enc}_K^{\mathsf{sym}}(m)$. This modification

is indistinguishable from the initial oracle since ciphertexts produced by the symmetric key scheme are indistinguishable from random ciphertexts.

3. We now modify the oracle from step 2 so that, for all ChSetup commands, instead of computing $(C, \xi) \leftarrow \mathsf{Enc}(pk, K)$, it computes $(C, \zeta) \leftarrow \mathsf{Enc}^*(pk)$. This oracle is indistinguishable from that of step 2 by the oblivious ciphertext generation property.

4. Lastly we modify the oracle from step 3 so that, for ChSetup commands, instead of computing $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ it computes $(pk, \xi) \leftarrow \mathsf{KG}^*(1^\lambda)$. This oracle is indistinguishable from that of step 3 by the oblivious key generation property.

The last step yields the message-ignoring oracle $\mathcal{R}$. Hence $\mathcal{M}$ and $\mathcal{R}$ are computationally indistinguishable.

The two oracle $\mathcal{M}$, $\mathcal{R}$ now define the complete non-information oracle $\mathcal{N}^\ell$ and hence the ℓ-equivocal secure channel functionality $\mathcal{F}_{\mathsf{SC}}^{\mathcal{N}^\ell}$. We must now describe an ideal-world simulation of our ℓ-NCE protocol.

The simulation while both parties are honest is actually very simple since the non-information oracle $\mathcal{N}$ actually runs the protocol and hence there is little that our simulator $\mathcal{S}$ must do! Essentially, to simulate the secure transfer of the index i, the simulator simply sends the length of the message (which is known since ℓ is known) to the adversary $\mathcal{A}$. To simulate the rest of the channel setup phase and also any encryption commands, the simulator $\mathcal{S}$ just passes all information from $\mathcal{N}$ to the adversary $\mathcal{A}$.

The only difficult part is simulating the first corruption. In the ideal world, the simulator is only given the internal state of the single machine $\mathcal{N}_i$ which is the message-processing oracles. In the real-world the state of the corrupted party also includes the randomness for all of the obliviously-generated public keys and ciphertexts (i.e. the internal state of all of the message-ignoring oracles). But the simulator can simulate this easily by using the ciphertext-faking and key-faking algorithms. In other words, for each symmetric key ciphertext E produced by an oracle $\mathcal{N}_i$, the simulator simply sets the random coins of the sender as E (since it is just a uniformly random value). Moreover, for each public key ciphertext C, the simulator sets the internal state of its sender as $\widetilde{\mathsf{Enc}}^*(pk, C)$. Lastly for each public key pk, the simulator sets the internal state of its sender as $\widetilde{\mathsf{KG}}^*(pk)$.

We can view the real world protocol as a series of the same ℓ oracles $\mathcal{N}_1, \ldots, \mathcal{N}_\ell$ where $\mathcal{N}_i$ (for the transferred index i) is the above described message-processing oracle and the rest are message-ignoring oracles. Hence the only difference between the real world and the ideal world is how the random coins of the corrupted party for its message-ignoring oracles are generated. In the real-world the adversary gets the actual coins while in the ideal-world the adversary gets coins produced by the faking algorithms. Let us denote the worlds by tuples showing which oracles are running for all indices other than i (i.e. either message-processing or messsage-ignoring) and how the state is generated (i.e. actual state, faked state). So the real world can be represented as a tuple (message-ignoring, actual). But, by the properties of the simulatable public key system, this is indistinguishable from the world (message-processing, faked). Lastly, since the fake algorithms do not use any secrets, and message-ignoring oracles are indistinguishable from message processing oracles, the world (message-processing, faked) is indistinguishable from (message-ignoring, faked). But this is the ideal world, and hence we have shown that the real-world is indistinguishable from the ideal world.

3.9.2 *Proof of Theorem 3.4.1*

By assumption, the underlying protocol π is well-structured and there is a simulator $\mathcal{S}_{\mathsf{semi}}$ which is setup-adaptive and input-preserving, such that π realizes the two-party functionality $\mathcal{F}^{f}_{\mathsf{SFE}}$ against a second-corruption adaptive adversary. In particular for any second-corruption adaptive adversary $\mathcal{A}_{\mathsf{sec}}$ there is $\mathcal{S}_{\mathsf{semi}}$ such that for any environment $\mathcal{Z}_{\mathsf{sec}}$

$$\mathrm{REAL}_{\pi,\mathcal{A}_{\mathsf{sec}},\mathcal{Z}_{\mathsf{sec}}} \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{F}^{f}_{\mathsf{SFE}},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}}$$

We now construct a simulator $\mathcal{S}$ for the protocol π' that uses fully non-committing secure channels. The simulator $\mathcal{S}$ runs an internal copy of the (fully adaptive) adversary $\mathcal{A}$ attacking the protocol π'. We can assume, without loss of generality, that the adversary $\mathcal{A}$ is just the "dummy" adversary which only follows instructions from the environment $\mathcal{Z}$. We define the simulation in terms of the following four stages:

I. The setup phase while both parties are honest.

II. The communication phase while both parties are honest.

III. The first corruption.

IV. Execution after the first corruption.

I. The setup phase while both parties are honest. First our simulator $\mathcal{S}$ initializes a copy of the semi-adaptive simulator $\mathcal{S}_{\mathsf{semi}} = (\mathcal{S}^{\mathsf{setup}}_{\mathsf{semi}}, \mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}})$. For stage (I), the entire setup phase of the protocol π is simulated by $\mathcal{S}^{\mathsf{setup}}_{\mathsf{semi}}$ which is oblivious of which parties are corrupted

II. The communication phase while both parties are honest. The main idea behind this simulation is that the real-world adversary does not see much at this time – just the message lengths and identities of the sender and receiver. However, because the protocol π is well-structured, these are known ahead of time. Hence the simulator $\mathcal{S}$ can simulate this phase by simply forwarding this publicly known data. Recall that the communication phase proceeds in rounds $i = 0, \ldots, n$ where in each round i the party b_i sends a message of length k_i to party $1 - b_i$. The party b_0 is the initiator, and afterwards $b_i = 1 - b_{i-1}$. The simulator $\mathcal{S}$ proceeds with "rounds" $i = 1, \ldots, n$ and in each round i sends the message $(\textsc{Send}, sid, P_{b_i}, k_i)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{SC}}$. We call this the *dummy execution* since the simulator $\mathcal{S}$ does not run any protocol behind the scenes but only passes message lengths to $\mathcal{Z}$. The adversary $\mathcal{A}$ may corrupt a party at any point during this interaction.

III. The first corruption. Assume that the first corruption is that of the party P_{first}. Then the simulator $\mathcal{S}$ gets the entire content of the ideal-world view of P_{first} including its input x_{first} from the environment and (possible) outputs y_{first} from $\mathcal{F}^f_{\mathsf{SFE}}$ depending on the point at which the corruption occurs.

The simulator $\mathcal{S}$ then produces an *imagined execution* between P_0 and P_1 as follows. It constructs a machine $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ which is a (new) environment that corrupts P_{first} immediately after the setup phase but honestly uses the inputs specified by $\mathcal{Z}$ to run the protocol π on behalf of P_{first}. The simulator $\mathcal{S}$ then activates the machine $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ (passing it any output from $\mathcal{S}^{\mathsf{setup}}_{\mathsf{semi}}$) and runs the simulation by $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ for the environment $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ (the adversary $\mathcal{A}$ is always dummy w.l.o.g. and hence we do not specify it). It runs this simulation, for as many rounds of the communication phase as occurred *before* P_{first} was corrupted.

In addition, $\mathcal{S}$ acts as the ideal functionality $\mathcal{F}^f_{\mathsf{SFE}}$ for $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$. If $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ attempts to give input to the ideal functionality then, since $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ is input-preserving, this input *is* x_{first} and $\mathcal{S}$ just ignores this. If $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ asks for output (at a point after both inputs have been submitted) then $\mathcal{S}$ passes it the actual output y_{first} on behalf of $\mathcal{F}^f_{\mathsf{SFE}}$.

Once $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ reaches the point where the environment $\mathcal{Z}$ corrupted P_{first}, the simulator $\mathcal{S}$ takes the internal state of $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ (i.e. the randomness used to run the protocol on behalf of the party P_{first} and the view of the protocol execution) and sets it as the internal state of P_{first} to be given to the actual environment $\mathcal{Z}$. This corresponds to *patching* the dummy execution by claiming that the imagined execution took place over the secure channel.

IV. Execution after the first corruption. After the first corruption, the simulator $\mathcal{S}^{\mathsf{prot}}_{\mathsf{semi}}$ is left to simulate the execution without interference.

Indistinguishability of Simulation: Now we need to argue the indistinguishability of the simulation. Intuitively, this follows simply from the fact that $\mathcal{S}_{\mathsf{semi}}$ can simulate a second-corruption adaptive adversary. In the following arguments we assume that the environment $\mathcal{Z}$ corrupts some party at some point. If not then in the ideal world as well as the real world, the environment $\mathcal{Z}$ only sees the lengths of the messages defined by the protocol π so the real world and ideal world are indistinguishable. If a corruption does occur at some point, then we use the following hybrid argument for a series of indistinguishable games.

Game 1 – The Ideal World: We define Game 1 to be $\mathrm{IDEAL}_{\mathcal{F}^f_{\mathsf{SFE}},\mathcal{S},\mathcal{Z}}$ - that is the ideal-world simulation with the environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ (as described above) interacting with the ideal functionality $\mathcal{F}$.

Game 2 – Second Corruption Adaptive Ideal World: In the simulation, the imagined execution is generated after the first party is corrupted *ex post facto*. We now define a new execution in which the imagined execution is the one that executes all along.

More precisely, we define an environment $\mathcal{Z}_{\mathsf{sec}}$, which runs an internal copy of $\mathcal{Z}$. However $\mathcal{Z}_{\mathsf{sec}}$ chooses a bit $b \leftarrow \{0,1\}$ randomly and corrupts the party P_b prior to protocol execution. The environment $\mathcal{Z}_{\mathsf{sec}}$ gets the input x_b for P_b from $\mathcal{Z}$, and runs a the protocol for P_b honestly using the input x_b and randomness r_b. In addition $\mathcal{Z}^b_{\mathsf{sec}}$ passes the message lengths, and identities of sender and receiver to $\mathcal{Z}$.

If $\mathcal{Z}$ requests the corruption of a party P_{first} then, if $\mathsf{first} \neq b$ the environment $\mathcal{Z}_{\mathsf{sec}}$ gives the output 0 and quits. Else, if $\mathsf{first} = b$, give the randomness r_b and the actual messages produced and received on behalf of the party P_b to the environment $\mathcal{Z}$. From this point on, $\mathcal{Z}_{\mathsf{sec}}$ just executes $\mathcal{Z}$ (allowing $\mathcal{Z}$ to produce the final output).

Now we consider the ideal world game with the environment $\mathcal{Z}_{\mathsf{sec}}$ and the simulator $\mathcal{S}_{\mathsf{semi}}$. We note that, if $b = \mathsf{first}$, then this is a syntactical re-writing of $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ where the "imagined execution" produced by $\mathcal{S}$ is actually being run by the environment $\mathcal{Z}^b_{\mathsf{sec}}$. Let G be an event that this is the case (i.e. $\mathsf{first} = b$). Then conditioned on G, Game 2 matches Game 1.

$$\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\} \overset{c}{\approx} \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

Game 3 – Second Corruption adaptive Real World: We now use the semi-adaptive security of the protocol π (as simulated by $\mathcal{S}_{\mathsf{semi}}$) to switch from the ideal-world (with environment $\mathcal{Z}_{\text{sec}}$ and simulator $\mathcal{S}_{\text{semi}}$) to the real-world (with environment $\mathcal{Z}_{\text{sec}}$ and a dummy-adversary $\mathcal{A}$). In particular we claim:

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} \mid G\} \overset{c}{\approx} \{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} \mid G\}$$

To see this, we note that the event G occurs with probability 1/2 (since the view of $\mathcal{Z}$ is independent of the choice of b by $\mathcal{Z}_{\text{sec}}$). Also, if G does not occur, then the output of both of games is 0. Assume that our claim does not hold, and so

$$\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} = 1 \mid G] \geq \epsilon$$

for some ϵ which is not negligible. Then

$$\begin{aligned}
&\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} = 1] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} = 1] \\
&= \Pr[G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} = 1 \mid G]) \\
&\quad + \Pr[\neg G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} = 1 \mid \neg G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} = 1 \mid \neg G]) \\
&= \Pr[G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{semi}},\mathcal{Z}_{\text{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} = 1 \mid G]) \\
&= \epsilon/2
\end{aligned}$$

where the summand in the 3rd line is 0 since, conditioned on $\neg G$, $\mathcal{Z}_{\text{sec}}$ always outputs 0. This contradicts the semi-adaptive security of π and hence out claim follows.

Game 4 – Real World: Now we notice that,

$$\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{sec}}} | G\} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

This follows simply from the fact that, conditioned on G the left-hand side is simply a syntactic rewriting of the right-hand side, where the environment $\mathcal{Z}$ runs the protocol on behalf of P_b and passes the state of P_b to $\mathcal{Z}$.

By the hybrid argument, we therefore get

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

which completes the proof.

3.9.3 *Proof Sketch of Theorem 3.4.2*

The simulation and the proof are similar to that of Theorem 3.4.1 as presented above. The main difference is that $\mathcal{S}$ needs to set-up ℓ possibilities for how to explain the communication over the ℓ-non-committing channel. Formally, the simulator $\mathcal{S}$ simulates stage (I) as in the proof of Theorem 3.4.1.

Then for stage (II), the simulator $\mathcal{S}$ creates ℓ copies of the simulators $\mathcal{S}_{\mathsf{semi}}^{\mathsf{prot}}$ and creates ℓ "honest-environments" ITMs $\mathcal{Z}_{\mathsf{sec}}^{b}(x,y)$: one (environment, simulator) pair for each choice of party b, its input $x \in X_b$ and its output $y \in Y_b$. We use the tuples (b,x,y) as indices and denote the (environment, simulator) pairs by

$$\left\{\left(\mathcal{S}_{\mathsf{semi}}, \mathcal{Z}_{\mathsf{sec}}^{b}(x,y)\right)\right\}_{b\in\{0,1\},x\in X_b,y\in Y_b}$$

Each "honest-environment" $\mathcal{Z}_{\mathsf{sec}}^{b}(x,y)$ runs the code of the party P_b as described by the protocol π using input x. In addition, if $\mathcal{S}^{\mathsf{sec}}$ needs to be given output on behalf of $\mathcal{F}_{\mathrm{SFE}}^{f}$, it is given y. To simulate the communication, $\mathcal{S}$ runs all ℓ simulations. It then initiates ℓ message-processing oracles $\mathcal{M}_i$ and randomly associates each i with an environment $\mathcal{Z}_{\mathsf{sec}}^{b}(x,y)$. On each tuple of messages produced by the ℓ (environment,simulator) pairs, these messages are given to the appropriate oracles and the output of the oracles is then used as side-channel information given by $\mathcal{S}$ to $\mathcal{Z}$. This is essentially a slightly more complicated version of the dummy communication phase used in the simulation for Theorem 3.4.1.

For stage (III), when a party P_{first} is corrupted, the simulator $\mathcal{S}$ learns the input x and (possibly) output y of the party P_{first}. It then explain the communication in stage II, by providing the internal state of the oracle corresponding to the (environment, simulator) pair for environment $\mathcal{Z}_{\mathsf{sec}}^{\mathsf{first}}(x,y)$ (if no input or output has been received then $\mathcal{S}$ can choose one of few consistent options for which environment to open). It passes the internal state of $\mathcal{Z}_{\mathsf{sec}}^{\mathsf{first}}(x,y)$ (i.e. its random coins and view of the protocol execution) to $\mathcal{Z}$ on behalf of P_{first}.

For stage (IV), the simulator $\mathcal{S}$ continues the simulation by using the copy of $\mathcal{S}_{\mathsf{semi}}$ chosen in the above step.

Indistinguishability of Simulation: This is similar to the proof of Theorem 3.4.1 and proceeds as a series of games argument.

Game 1 – The Ideal World: We define Game 1 to be $\text{IDEAL}_{\mathcal{F}^f_{\text{SFE}},\mathcal{S},\mathcal{Z}}$ - that is the ideal-world simulation with the environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ (as described above) interacting with the ideal functionality $\mathcal{F}_f$.

Game 2 – Second Corruption Adaptive Ideal World: This is similar to Game 2 of the proof of Theorem 3.4.1. The one difference is that, for stage (II), the environment $\mathcal{Z}_{\mathsf{sec}}$ now produces the side-information for $\mathcal{Z}$ by using a message-processing oracle to encrypt the actual protocol execution by P_b, and $\ell - 1$ message-ignoring oracles for the rest. Again, we define the event G as in Game 2 of the proof for Theorem 3.4.1. It is easy to see that:

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\} \overset{c}{\approx} \{\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

Indeed, the two worlds are syntactical re-writings of each other, in the sense that the left hand side has $\mathcal{Z}$ being simulated by $\mathcal{S}$ which runs a conversation $\mathcal{Z}_{\mathsf{sec}}$ and $\mathcal{S}_{\mathsf{semi}}$ while, on the right hand side, $\mathcal{Z}_{\mathsf{sec}}$ is running the whole time and is simulated by $\mathcal{S}_{\mathsf{semi}}$. The one difference is that, in Game 1, there are ℓ message-processing oracles, while in Game 2, there is only one and the rest are message-ignoring. However, by assumption, these are indistinguishable.

Game 3 – Second Corruption adaptive Real World: We now use the second-corruption adaptive security of the protocol π (as simulated by $\mathcal{S}_{\mathsf{semi}}$) to switch from the ideal-world to the real-world. In particular we get:

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\} \overset{c}{\approx} \{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

The argument is the same as in the proof of Theorem 3.4.1.

Game 4 – Real World: Now we claim that,

$$\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} | G\} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

This follows simply from the fact that, conditioned on G the left-hand side is simply a syntactic rewriting of the right-hand side, where the environment $\mathcal{Z}$ runs the protocol on behalf of P_b and passes the state of P_b to $\mathcal{Z}$.

By the hybrid argument, we therefore get

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

which completes the proof.

3.9.4 Proof of Theorem 3.6.2

To finish the proof, we first need to construct a simulator such that there is no PPT environment $\mathcal{Z}$ which follows a second-corruption adaptive adversarial strategy can distinguish the execution with the adversary $\mathcal{A}$ and OT protocol in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world (see Figure 3.8), and the execution with the setup-adaptive simulator $\mathcal{S}$ and ideal functionality $\mathcal{F}_{\mathrm{OT}}$. We further need to show the constructed simulator is input preserving.

We give the construction of the simulator. Note that the underlying commitment is an adaptively secure commitment; so the simulator can take advantage of the extractability and equivocality of the UC commitment to set up crs_{ot} in the coins tossing stage based on the mode information. The simulation is very similar to that in [PVW08]. But now the underlying dual mode encryption satisfies an enhanced definition, and the simulator has more power to handle the second-corruption adaptive attacks from the adversaries; the simulator can use the $\widetilde{\mathsf{Enc}}$ and $\widetilde{\mathsf{Equiv}}$ algorithms to handle a further corruption of the sender if the receiver is initially corrupted, and use $\widetilde{\mathsf{EquivKey}}$ algorithm to handle a further corruption of the receiver if the sender is initially corrupted.

- **Simulating the communication with $\mathcal{Z}$.** The simulator $\mathcal{S}$ simulates the adversary $\mathcal{A}$ internally which can interact with the external environment $\mathcal{Z}$, i.e., whenever $\mathcal{A}$ and $\mathcal{Z}$ exchange messages with each other, the simulator will forward such messages between them. Further whenever $\mathcal{A}$ corrupts a party, $\mathcal{S}$ corrupts the corresponding dummy party.

- **Trusted setup.** Note that our $\mathcal{S}$ should be a setup-adaptive simulator which includes two parts $\mathcal{S}^{\text{setup}}$ and $\mathcal{S}^{\text{prot}}$. Here we give the construction of $\mathcal{S}^{\text{setup}}$. The simulator computes the CRS in the following way: generate $(crs_{\text{com}}, \tau_{\text{com}})$ for the UC commitment, and generate $(G, crs_{\text{sys}}, \tau_{\text{sys}}) \leftarrow \mathsf{PG}_{\text{sys}}(1^{\lambda})$, and further set crs_{com} and crs_{sys} as crs, and set the corresponding trapdoor τ_{com} and τ_{sys} as the trapdoor τ. When parties query $\mathcal{F}_{\text{CRS}}$, return $(\mathtt{CRS}, sid, crs)$. Note that the CRS can be learned by $\mathcal{A}$ even no party is corrupted, and no mode information has been committed by the simulator in the trusted setup stage. In next several items, we construct the other part of the simulator, i.e., $\mathcal{S}^{\text{prot}}$.
- **Simulation of the initially corrupted receiver case.** Given the adversary $\mathcal{A}$ is the second-corruption adaptive adversary, here we consider the case that the receiver is corrupted initially while the sender is honest, and the sender could be further corrupted at any point in the communication stage. The simulator chooses the messy mode and simulates the sender as follows.

  First, the simulator generates $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(0, G, crs_{\text{sys}}, \tau_{\text{sys}})$, where $(G, crs_{\text{sys}}, \tau_{\text{sys}})$ is generated in the trusted setup, and now the messy trapdoor $\tau_{\text{ot}} = (\tau_{\text{sys}}, \tau_{\text{tmp}})$ and the CRS $crs_{\text{ot}} = (crs_{\text{sys}}, crs_{\text{tmp}})$; then in the coin-tossing phase, the simulator computes s such that $crs_{\text{tmp}} = r + s$, where r is extracted from the commitment value from a corrupted receiver. Further, in the transfer phase, the simulator can obtain the sender's input $x_{1-\rho}$ for the non-messy branch $1 - \rho$ by querying the functionality $\mathcal{F}_{\text{OT}}$ with the input bit $1 - \rho$, i.e., the simulator in the name of corrupted receiver sends $(\mathtt{Receiver}, sid, 1 - \rho)$ to the functionality and obtain $(\mathtt{Output}, sid, x_{1-\rho})$ from the functionality; note that here ρ is extracted by using the messy trapdoor τ_{ot} from the

pk computed by the corrupted receiver; then the simulator computes $y_{1-\rho}$ honestly for $x_{1-\rho}$ by using randomly selected $\zeta_{1-\rho}$, and computes y_ρ by running the fake encryption algorithm, i.e., $(y_\rho, \omega_\rho) \leftarrow \widetilde{\mathsf{Enc}}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}}, pk, \rho)$. Later if the sender is corrupted, then the simulator based on the learned x_ρ runs the internal state reconstruction algorithm to compute ζ_ρ, i.e., $\zeta_\rho \leftarrow \widetilde{\mathsf{Equiv}}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}}, pk, \rho, y_\rho, \omega_\rho, x_\rho)$, and returns (ζ_0, ζ_1) as the sender's internals.

- **Simulation of the initially corrupted sender case.** Here we consider the case that the sender is corrupted initially while the receiver is honest, and the receiver could be further corrupted at any point in the communication stage. The simulator chooses the decryption mode and simulates the receiver as follows.

  First, in the coin-tossing phase, the simulator computes a fake commitment value based on $(crs_{\mathrm{com}}, \tau_{\mathrm{com}})$; then the simulator generates $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(1, G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$, where $(G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$ is previously generated in the trusted setup, now the decryption trapdoor $\tau_{\mathrm{ot}} = (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$ and the CRS $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$; after obtaining s, the simulator computes r such that $crs_{\mathrm{tmp}} = r + s$, where s is received from a corrupted sender; later the simulator equivocates the previous fake commitment value into r. Further, in the transfer part, the simulator runs $(pk, sk_0, sk_1) \leftarrow \widetilde{\mathsf{EquivKey}}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}})$. After receiving ciphertexts (y_0, y_1), the simulator decrypts them into (x_0, x_1), e.g., $x_b \leftarrow \mathsf{Dec}(crs_{\mathrm{ot}}, pk, sk_b, y_b)$ for $b = 0, 1$. Then $\mathcal{S}$ sends $(\mathtt{Sender}, sid, \langle x_0, x_1 \rangle)$ to the functionality $\mathcal{F}_{\mathrm{OT}}$. Later if the receiver is corrupted the simulator, based on the learned σ, reveals sk_σ as the receiver's internals.

- **Simulation of the remaining cases.** If both parties are corrupted, the simulator just follows the internal $\mathcal{A}$'s instructions to simulate the transcripts between the two corrupted parties. If both parties are hones then the simulator runs two parties honestly based on randomly selected inputs.

We next need to argue that no PPT environment can distinguish with non-negligible probability the interaction with the protocol in the CRS hybrid world and second-corruption adaptive adversary $\mathcal{A}$, or with the functionality and simulator $\mathcal{S}$ described above.

H1 This is the CRS hybrid world.

H2 Based on H1, we change the CRS setup, and run $(G, crs_{\text{sys}}, \tau_{\text{sys}}) \leftarrow \mathsf{PG}_{\text{sys}}(1^{\lambda})$. Now trapdoor τ_{sys} is known.

H1 and H2 are indistinguishable given that the system CRS are identically distributed and τ_{sys} is not used yet.

H3 Based on H2, we change the CRS setup, generate $(crs_{\text{com}}, \tau_{\text{com}})$ for the UC commitment, and the trapdoor τ_{com} is known. If it is the case that the sender is honest but the receiver is initially corrupted, we do the following further modification. We generate $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(0, G, crs_{\text{sys}}, \tau_{\text{sys}})$, where $(G, crs_{\text{sys}}, \tau_{\text{sys}})$ is, as generated in H1. Next in the coin-tossing phase, compute s such that $crs_{\text{tmp}} = r + s$, where r is extracted from the commitment transcripts from the corrupted receiver. Now the messy trapdoor $\tau_{\text{ot}} = (\tau_{\text{sys}}, \tau_{\text{tmp}})$ and the CRS $crs_{\text{ot}} = (crs_{\text{sys}}, crs_{\text{tmp}})$. If later receive a corruption command from the adversary that an honest sender is further corrupted, just reveal its input to the adversary.

H2 and H3 are indistinguishable given that the underlying commitment is UC secure and that the mode indistinguishability of the underlying enhanced dual mode encryption.

H4 Based on H3, if it is the case that the sender is honest but the receiver is initially corrupted, we do the following modification. Instead of honestly producing the ciphertexts (y_0, y_1) for both branches, we extract the messy branch number and use the faking encryption to produce the ciphertext for the messy branch, and later run the state reconstruction algorithm to compute the internals when the sender is further corrupted. For the non-messy branch, we compute the ciphertext honestly.

H3 and H4 are indistinguishable given the messy branch identification and ciphertext equivocation property of the underlying enhanced dual mode encryption.

H5 We make a further modification based on H4. Now we turn to consider the case that the sender is corrupted initially but the receiver is honest. In the coin-tossing phase, we fake the commitment transcripts from the receiver based on $(crs_{\text{com}}, \tau_{\text{com}})$ with the adversary. Further generate $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(1, G, crs_{\text{sys}}, \tau_{\text{sys}})$, where $(G, crs_{\text{sys}}, \tau_{\text{sys}})$ is previously generated in H1. After obtaining s from the adversary, compute r such that $crs_{\text{tmp}} = r + s$. Now the decryption trapdoor $\tau_{\text{ot}} = (\tau_{\text{sys}}, \tau_{\text{tmp}})$ and the CRS $crs_{\text{ot}} = (crs_{\text{sys}}, crs_{\text{tmp}})$. Later we can equivocate the previous faked commitment transcripts into r if the receiver is further corrupted.

H4 and H5 are indistinguishable given that the underlying commitment is UC secure and that the mode indistinguishability of the underlying enhanced dual mode encryption.

H6 Based on H5, if it is the case that the sender is corrupted initially but the receiver is honest, we do the following modification. Instead of honestly producing the *pk* based on the receiver's input, we use the dual key generation to produce *pk* and sk_0, sk_1, later reveals the corresponding decryption key as its internal state when the receiver is further corrupted.

H5 and H6 are indistinguishable given the encryption key duality property of the underlying enhanced dual mode encryption.

H7 Based on H6, now we consider the cases that both parties are initially corrupted and that both parties are honest. For the former, just follow the adversary's instruction, and for the latter generate the transcripts honestly based on randomly selected inputs and the real inputs are not used.

H6 and H7 are indistinguishable given that the underlying dual mode encryption is secure.

H8 The ideal world running with the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}_{\mathrm{OT}}$.

In H7, the inputs for honest parties are not used. So H7 and H8 are indistinguishable. Therefore the difference between the CRS hybrid world and the ideal world are negligible.

Based on the above argument, we conclude that the protocol in Figure 3.8 is second-corruption adaptively secure.

Next we need to show the constructed simulator is input-preserving. This can be easily verified. For the case with a corrupted sender and an honest receiver, if the corrupted sender honestly follows the protocol using input $\langle x_0, x_1 \rangle$, the input can be extracted as in the description of $\mathcal{S}$; further $\mathcal{S}$ must submit the extracted $\langle x_0, x_1 \rangle$ to the functionality, otherwise the environment can distinguish the two worlds based on the output from the honest receiver. Similarly for the case with an honest sender and a corrupted receiver, if the corrupted receiver honestly follows the protocol using input σ, the input can be extracted as in the description of $\mathcal{S}$; further $\mathcal{S}$ must submit the extracted bit σ to the functionality to learn x_σ, otherwise if $\mathcal{S}$ submit a different bit $1 - \sigma$ to the functionality and learn $x_{1-\sigma}$, the simulator has no idea of x_σ, and the environment can distinguish the two worlds based on the output from the corrupted receiver.

Together we show the protocol based on Figure 3.8 is semi-adaptively secure to realize $\mathcal{F}_{\mathrm{OT}}$ in the CRS hybrid model. This finishes the proof.

Chapter 4
BLIND SIGNATURES

4.1 Introduction

A blind signature is a cryptographic primitive that was proposed by Chaum [Cha82]; it is a digital signature scheme where the signing algorithm is split into a two-party protocol between a user (or client) and a signer (or server). The signing protocol's functionality is that the user can obtain a signature on a message that she selects in a blind fashion, i.e., without the signer being able to extract some useful information about the message from the protocol interaction. At the same time the existential unforgeability property of digital signatures should hold, i.e., after the successful termination of a number of n corrupted user instantiations, an adversary should be incapable of generating signatures for $(n+1)$ distinct messages.

A blind signature is a very useful privacy primitive that has many applications in the design of electronic-cash schemes, the design of electronic voting schemes as well as in the design of anonymous credential systems. Since the initial introduction of the primitive, a number of constructions have been proposed based on different intractability assumptions and security models with various communication and time complexities. The first formal treatment of the primitive in a stand-alone model and assuming random oracles (RO) was given by Pointcheval and Stern in [PS96].

Blind signatures have been implemented in real world Internet settings (e.g., in the Votopia [Kim04] voting system) and thus the investigation of more realistic attack models for blind signatures is of pressing importance. Juels, Luby and Ostrovsky [JLO97] presented a formal treatment of blind signatures that included the possibility for an adversary to launch attacks that use arbitrary concurrent interleaving of either user or signer protocols. Still, the design of schemes that satisfied such stronger modeling proved somewhat elusive. In fact, Lindell [Lin03] showed that unbounded concurrent security for blind signatures is impossible under a simulation-based security definition without any setup assumption; more recently in [HKKL07], the generic feasibility of blind signatures without setup assumptions was shown but using a game-based security formulation.

With respect to practical provably secure schemes (which is the focus of the present work), assuming random oracles or some setup assumption, various efficient constructions were proposed: for example, [BNPS03, Bol03] presented efficient two-move constructions in the RO model, while [KZ06, Oka06] presented efficient constant-round constructions without random oracles employing a common reference string (CRS) model that withstand concurrent attacks. While achieving security under concurrent attacks is an important property for the design of useful blind signatures, a blind signature scheme may still be insecure for a certain deployment. Game-based security definitions [PS96, JLO97, CKW04, KZ06, Oka06, HKKL07, CNS07] capture properties that are intuitively desirable. But the successive extensions of definitions in the literature and the differences between the various models in fact exemplify the following: on the one hand capturing all desirable properties of a complex cryptographic primitive such as a blind signature is a difficult task, while on the other, even if such properties are attained, a "provably secure" blind signature may still be insecure if deployed within a larger system. For this reason, it is important to consider the realization of blind signatures under a general simulation-based security formulation such as the one provided in the Universal Composability (UC) framework of Canetti [Can01] that enables us to formulate cryptographic primitives so that they remain secure under arbitrary deployments and interleavings of protocol instantiations.

In the UC setting, against static adversaries, it was shown how to construct blind signatures in the CRS model [Fis06] with two moves of interaction. Though the construction in [Fis06] is round-optimal, it is unknown whether it can admit concrete practical instantiations. In addition, security is argued only against static adversaries; and while it should be feasible to extend the construction of [Fis06] in the adaptive setting this can only exacerbate the difficulty of concretely realizing the basic design. Note that using the secure two party computation compiler of [CLOS02] one can derive adaptively secure blind signatures but this approach is also generic and does not suggest any concrete design.

4.1.1 Our Results

In this work we study the design of blind signatures in the UC framework against adaptive adversaries. We focus on maintaining a "practice-oriented" approach that entails the following: (i) a constant number of rounds, (ii) a choice of session scope that is consistent with how a blind signature would be implemented in practice, in particular a multitude of clients and one signer should be supported within a single session, (iii) a trusted setup string that is of constant length in the number of parties within a session, (iv) avoiding, if possible, cryptographic primitives that are "per-bit", such as bit-commitment, where one has to spend a communication length of $\Omega(\lambda)$ where λ is a security parameter per bit of private input. Our results are as follows:

Equivocal blind signatures. We introduce a new property for blind signatures, called equivocality that is suitable for arguing security against adaptive adversaries. In an equivocal blind signature there exists a simulator that has the power to construct the internal state of a client including all random tapes so that any simulated communication transcript can be mapped to any given valid message-signature pair. This capability should hold true even after a signature corresponding to the simulated transcript has been released to the adversary. Equivocality can be seen as a strengthening of the notion of blindness as typically defined in game-based security formulations of blind signatures: in an equivocal blind signature, signing transcripts can be simulated in an independent fashion to the message-signature pair they correspond to.

General methodology for building UC blind signatures. We present a general methodology for designing adaptively secure UC blind signatures. Our starting point is the notion of an *equivocal lite blind signature*: The idea behind "lite" blind signatures is that security properties should hold under the condition that the adversary deposits the private tapes of the parties he controls. This "open-all-private-tapes" approach simplifies the blind signature definitions substantially and allows one to separate security properties that relate to zero-knowledge compared to other necessary properties for blind signatures. Note that this is *not* an honest-but-curious type of adversarial formulation as the adversary is not required to be honestly simulating corrupted parties; in particular, the adversary may deviate from honest protocol specifications as long as he can present private tapes that match his communication transcripts.

We then demonstrate two instantiations of lite blind signature, one that is based on generic cryptographic primitives that is inspired by the blind signature construction of [Fis06] and one based on the design and the 2SDH assumption of [Oka06].

Study of the ZK requirements for UC blind signatures. Having demonstrated equivocal lite blind-signature as a feasible starting building block, we then focus on the formulation of the appropriate ZK-functionalities that are required for building blind signatures in the adaptive adversary setting. Interestingly, the user and the signer have different ZK "needs" in a blind signature. In particular the corresponding ZK-functionalities turn out to be simplifications of the standard multi-session ZK functionality $\mathcal{F}_{\mathrm{MZK}}$ that restrict the multi-sessions to occur either from many provers to a single verifier (we call this $\mathcal{F}_{\mathrm{SVZK}}$) or from a single prover to many verifiers (we call this $\mathcal{F}_{\mathrm{SPZK}}$). Note that this stems from our blind signature *session scope* that involves a multitude of users interacting with a single signer (this is consistent with the notion that a blind-signature signer is a server within a larger system and is expected that the number of such servers would be very small compared to a much larger population of users and verifiers).

First, regarding SVZK, the ZK protocol that users need to execute, we show that it can be realized in a commit-and-prove fashion using a commitment scheme that, as it is restricted in single-verifier setting, it does not require built-in non-malleability (while such property would be essential for general multi-session UC commitments). We thus proceed to realize $\mathcal{F}_{\mathrm{SVZK}}$ using mixed commitments [DN02, Nie03] with only a constant length common reference string (as opposed to linear in the number of parties that is required in the multi-session setting). Second, regarding SPZK, the ZK protocol that signer needs to execute towards the users, we find the rather surprising result that a much weaker version of SPZK, called *leaking SPZK* is enough for achieving adaptive security of our blind signature protocol! This enables a much more efficient realization design for SPZK as we can implement it using merely an extractable commitment and a Sigma protocol (alternatively, using an Ω-protocol [GMY06]). The intuition behind this result is that in a blind signature the signer is not interested in hiding his input in the same way that the user is: this can be seen by the fact that the verification-key itself leaks a lot of information about the signing-key to the adversary/environment, thus, using a full-fledged zero-knowledge instantiation is an overkill from the signer's point of view; this phenomenon was studied in the context of zero-knowledge in [KZ07].

Finally we note that our $\mathcal{F}_{\mathrm{SPZK}}$ functionality can be seen as a special instance of client-server computation as considered in [PS05] (where the relaxed non-malleability requirement of such protocols was also noted); interestingly $\mathcal{F}_{\mathrm{SVZK}}$ falls outside that framework (despite its client-server nature).

4.1.2 Recent Related Work

Fischlin [Fis06] proposed a blind signature functionality and gave an elegant round-optimal constructions against static corruption. Recently Abe et al. [AO09] proposed a very interesting relaxation of Fischlin's functionality, and gave practical round-optimal constructions to realize such relaxed functionality in the UC setting against adaptive corruption.

Our constructions are not round-optimal, but we note that by using Abe et al.'s techniques, we could improve the round complexity and communication complexity of the practical protocol in this work. We further note that the signatures obtained in our work are much shorter than those in [AO09, AHO10].

Regarding the definitions of blind signature functionality, ours is different from theirs. In our formulation, the corrupted signer is allowed to change its signing secret, while this is not captured in [Fis06, AO09]. Indeed, in the reality, an adversarial signer may initial such attacks (please also refer to [HK07]), and from this point of view, our formulation is more natural. We further note that, our formulation could be further relaxed in the same spirit in [AO09] to allow for more efficient constructions.

4.2 Blind Signature Schemes

In this section, we present the syntax and security definitions for blind signatures. We note that the definitions are based on the formulations in the literature.

Definition 4.2.1 (Blind Signature Schemes). Syntax of a blind digital signature scheme $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{CRS}, \mathsf{KG}, U, S, \mathsf{Vrf}\}$ is defined as follows:

- $(crs, \tau) \leftarrow \mathsf{CRS}(1^\lambda)$ is a PPT CRS generation algorithm which takes as an input a security parameter λ and outputs a pair $\langle crs, \tau \rangle$ of CRS and its trapdoor.
- $(vk, sk) \leftarrow \mathsf{KG}(crs)$ is a PPT key generation algorithm[1] which takes as an input crs and outputs a pair (vk, sk) of verification and signing keys.
- $(\langle \alpha, \sigma \rangle, \zeta; \beta, \eta) \leftarrow [\![U(crs, vk, m); S(crs, vk, sk)]\!]_{\mathrm{LR}}$ is an interactive protocol between two PPT parties, a user U and the signer S. At the end of protocol interaction, the user U

[1]Here we consider all random coins used in KG have been included in sk.

outputs a signature σ as well as a bit α where $\alpha = 1$ means that U's participation of the protocol is complete, and otherwise if U]s participation is not complete, then set $\alpha = 0$ and $\sigma = 0$. The signer S outputs a bit β where $\beta = 1$ means S's participation is complete and $\beta = 0$ otherwise. Here ζ and η denote the randomness used by U and S respectively.

– $\phi \leftarrow \mathsf{Vrf}(crs, vk, m, \sigma)$ is a deterministic polynomial time algorithm, where $\phi = 1$ means message-signature pair (m, σ) is valid, and $\phi = 0$ otherwise. □

Remark 4.2.2. Note that in the plain model, CRS is not employed, and the blind signature scheme $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{KG}, S, U, \mathsf{Vrf}\}$.

A signature scheme is naturally defined by a blind signature when party S and party U are same, i.e. $S = U$; in this case, the scheme $\Sigma(\mathsf{BS})$ can collapse to a plain signature scheme $\Sigma(\mathsf{SIG}) = \mathsf{SIG}.\{\mathsf{KG}, \mathsf{SG}, \mathsf{Vrf}\}$. Here KG and Vrf are same as that defined in $\Sigma(\mathsf{BS})$, and SG is the signature generation algorithm which can be used to generate signature σ for message m by computing $(\sigma, \zeta) \leftarrow \mathsf{SG}(vk, sk, m)$ (such algorithm is immediately from the collapse of S, U into a single unit). □

Definition 4.2.3 (Secure Blind Signature Schemes). A blind signature scheme $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{CRS}, \mathsf{KG}, U, S, \mathsf{Vrf}\}$ is <u>secure</u> if the following properties hold:

– Completeness: For all PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} crs \leftarrow \mathsf{CRS}(1^\lambda); (vk, sk) \leftarrow \mathsf{KG}(crs); m \leftarrow \mathcal{A}(vk); \\ (\langle \alpha, \sigma \rangle, \zeta; \beta, \eta) \leftarrow [\![U(crs, vk, m); S(crs, vk, sk)]\!]_{\mathrm{LR}}; \\ \phi \leftarrow \mathsf{Vrf}(crs, vk, m, \sigma) : \alpha = 0 \vee \phi = 0 \vee \beta = 0 \end{array}\right] \overset{c}{\approx} 0.$$

– Unforgeability: For all PPT $\mathcal{A}$, $\Pr[\mathbf{Unforge}_{\mathcal{A}}(\lambda) = 1] \overset{c}{\approx} 0$ where experiment $\mathbf{Unforge}_{\mathcal{A}}$ is defined as follows

```
$\mathbf{Unforge}_{\mathcal{A}}(\lambda)$
    $crs \leftarrow \mathsf{CRS}(1^{\lambda})$;
    $(vk, sk) \leftarrow \mathsf{KG}(crs)$;
    $(\langle \beta_1, \beta_2, \ldots, \beta_\ell \rangle, \eta) \leftarrow [\![\mathcal{A}(crs, vk); S(crs, vk, sk)]\!]^{\ell}_{\mathrm{R}}$;
    $((m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_{k+1}, \sigma_{k+1})) \leftarrow \mathcal{A}$;
    Return 1 iff all the following three conditions hold
        $(i)\ \sum_{i=1}^{\ell} \beta_i \le k$
        $(ii)\ m_i \neq m_j$ for all $1 \le i < j \le k+1$
        $(iii)\ \mathsf{Vrf}(crs, vk, m_i, \sigma_i) = 1$ for all $1 \le i \le k+1$.
```

– (Indistinguishable) Blindness: For all PPT $\mathcal{A}$, $\Pr[\mathbf{IndBlind}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{IndBlind}_{\mathcal{A}}(\lambda, 1)]$, where the experiment $\mathbf{IndBlind}_{\mathcal{A}}$ is defined as below:

```
$\mathbf{IndBlind}_{\mathcal{A}}(\lambda, b)$
    $crs \leftarrow \mathsf{CRS}(1^{\lambda})$;
    $(vk, m_0, m_1) \leftarrow \mathcal{A}(crs)$;
    $((\alpha_b, \sigma_b), \zeta_b) \leftarrow [\![U(crs, vk, m_b); \mathcal{A}]\!]_{\mathrm{L}}$;
    $((\alpha_{1-b}, \sigma_{1-b}), \zeta_{1-b}) \leftarrow [\![U(crs, vk, m_{1-b}); \mathcal{A}]\!]_{\mathrm{L}}$;
    If $\alpha_0 = 0 \vee \alpha_1 = 0$ then set $\sigma_0 = \sigma_1 = 0$;
    Return $b' \leftarrow \mathcal{A}(\sigma_0, \sigma_1)$.
```

□

In the literature, there is stronger security notion of unforgeability defined as below. In this thesis, we focus on the regular unforgeability above.

- STRONG UNFORGEABILITY: For all PPT $\mathcal{A}$, $\Pr[\mathbf{StrongUnforge}_{\mathcal{A}}(\lambda) = 1] \overset{c}{\approx} 0$ where experiment $\mathbf{StrongUnforge}_{\mathcal{A}}$ is defined same as experiment $\mathbf{StrongUnforge}_{\mathcal{A}}$ above except condition (ii) is modified into $(m_i, \sigma_i) \neq (m_j, \sigma_j)$ for all $1 \leq i < j \leq k+1$.

Above we give an indistinguishability style formulation of the blindness property. Please refer to [CNS07] for a stronger indistinguishability-based formulation considering the abort of the users. Below we give a simulation based formulation of such property.

- (SIMULATABLE) BLINDNESS: For all PPT $\mathcal{A}$, there exist a tuple of PPT algorithms $\widetilde{\mathsf{CRS}}$, $\widetilde{U}$,$\widetilde{\mathsf{SG}}$ such that $\Pr[\mathbf{SimBlind}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{SimBlind}_{\mathcal{A}}(\lambda, 1)]$, where the experiment $\mathbf{SimBlind}_{\mathcal{A}}(\lambda, b)$ is defined as below:

```
$\mathbf{SimBlind}_{\mathcal{A}}(\lambda, 1)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathbf{SimBlind}_{\mathcal{A}}(\lambda, 0)$
    $(crs, \tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs, vk, m)$
    $(\langle \alpha, \sigma \rangle, \zeta) \leftarrow [\![U(crs, vk, m); \mathcal{A}]\!]_{\mathrm{L}}$;
    Output $(\alpha, \sigma)$.

$\mathcal{O}_0(crs, \tau, vk, m)$
    $(\alpha, \xi_1) \leftarrow [\![\widetilde{U}(crs, \tau, vk); \mathcal{A}]\!]_{\mathrm{L}}$;
    $(\sigma, \xi_2) \leftarrow \widetilde{\mathsf{SG}}(crs, \tau, vk, m)$;
    If $\alpha = 0$ then set $\sigma = 0$;
    Output $(\alpha, \sigma)$.
```

Definition 4.2.4 (Equivocal Blind Signatures). We say a secure blind signature scheme $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{CRS}, \mathsf{KG}, U, S, \mathsf{Vrf}\}$ is equivocal if the following property holds:

- EQUIVOCALITY: For all PPT $\mathcal{A}$, there exist a tuple of PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{U}, \widetilde{\mathsf{SG}}, \widetilde{\mathsf{Equiv}}$ such that $\Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda, 0)] \overset{c}{\approx} \Pr[\mathbf{Equiv}_{\mathcal{A}}(\lambda, 1)]$, where the experiment $\mathbf{Equiv}_{\mathcal{A}}(\lambda, b)$ is defined as below:

```
$\mathbf{Equiv}_{\mathcal{A}}(\lambda,1)$
    $crs \leftarrow \mathsf{CRS}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathbf{Equiv}_{\mathcal{A}}(\lambda,0)$
    $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs, vk, m)$
    $(\langle\alpha,\sigma\rangle,\zeta) \leftarrow [\![U(crs,vk,m);\mathcal{A}]\!]_{\mathrm{L}}$;
    Output $(\alpha,\sigma,\zeta)$.

$\mathcal{O}_0(crs, \tau, vk, m)$
    $(\alpha,\xi_1) \leftarrow [\![\widetilde{U}(crs,\tau,vk);\mathcal{A}]\!]_{\mathrm{L}}$;
    $(\sigma,\xi_2) \leftarrow \widetilde{\mathsf{SG}}(crs,\tau,vk,m)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(\xi_1,\xi_2)$;
    If $\alpha = 0$ then set $\sigma = 0$;
    Output $(\alpha,\sigma,\zeta)$.
```

□

4.3 *Building Block: Lite Blind Signatures*

4.3.1 *Definitions*

In this section, we focus on blind signature scheme $\Sigma(\mathsf{2MBS}) = \mathsf{2MBS}.\{\mathsf{CRS}, \mathsf{KG}, \mathsf{Blind}, \mathsf{Sign}, \mathsf{SG}, \mathsf{Vrf}\}$. Here algorithms $\mathsf{CRS}, \mathsf{KG}, \mathsf{Vrf}$ are same as the ones defined in Definition 4.2.1;

algorithms Blind, Sign, SG consist of a special implementation of the signature generation protocol between the signer S and a user U, as presented in Figure 4.1.

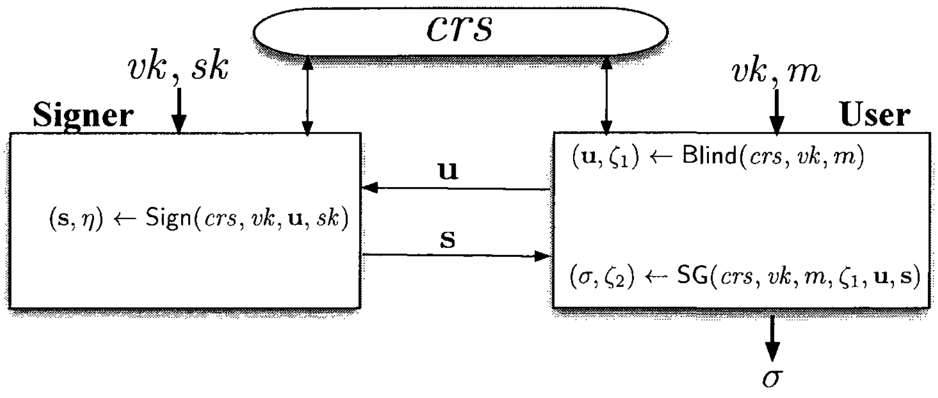

crs
vk, sk
Signer
vk, m
User
$(\mathbf{u}, \zeta_1) \leftarrow \mathsf{Blind}(crs, vk, m)$
$\mathbf{u}$
$(\mathbf{s}, \eta) \leftarrow \mathsf{Sign}(crs, vk, \mathbf{u}, sk)$
$\mathbf{s}$
$(\sigma, \zeta_2) \leftarrow \mathsf{SG}(crs, vk, m, \zeta_1, \mathbf{u}, \mathbf{s})$
σ

Figure 4.1: Outline of a two-move signature generation protocol 2MBS.

Definition 4.3.1 (Lite Blind Signatures). We say a secure blind signature scheme $\Sigma(\mathsf{2MBS}) = \mathsf{2MBS}.\{\mathsf{CRS}, \mathsf{KG}, \mathsf{Blind}, \mathsf{Sign}, \mathsf{SG}, \mathsf{Vrf}\}$ is lite if the following properties hold:

– Completeness: For all PPT $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} crs \leftarrow \mathsf{CRS}(1^\lambda); (vk, sk) \leftarrow \mathsf{KG}(crs); m \leftarrow \mathcal{A}(vk); (\mathbf{u}, \zeta_1) \leftarrow \mathsf{Blind}(crs, vk, m); \\ (\mathbf{s}, \eta) \leftarrow \mathsf{Sign}(crs, vk, \mathbf{u}, sk); (\sigma, \zeta_2) \leftarrow \mathsf{SG}(crs, vk, m, \zeta_1, \mathbf{u}, \mathbf{s}); \\ \phi \leftarrow \mathsf{Vrf}(crs, vk, m, \sigma) : \alpha = 0 \vee \phi = 0 \vee \beta = 0 \end{array}\right] \overset{c}{\approx} 0.$$

– Lite-Equivocality: For all PPT $\mathcal{A}$, there exist PPT algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Blind}}, \widetilde{\mathsf{SG}}, \widetilde{\mathsf{Equiv}}$ such that $\Pr[\mathbf{LiteEquiv}_{\mathcal{A}}(\lambda,0)] \overset{c}{\approx} \Pr[\mathbf{LiteEquiv}_{\mathcal{A}}(\lambda,1)]$, where $\mathbf{LiteEquiv}_{\mathcal{A}}(\lambda,b)$ is defined as:

```
$\mathbf{LiteEquiv}_{\mathcal{A}}(\lambda,1)$
    $crs \leftarrow \mathsf{CRS}(1^{\lambda})$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_1(crs,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_1(crs,vk,m)$
    $(\mathbf{u},\zeta_1) \leftarrow \mathsf{Blind}(crs,vk,m)$;
    $(\mathbf{s},\eta,sk) \leftarrow \mathcal{A}(\mathbf{u})$;
    $(\sigma,\zeta_2) \leftarrow \mathsf{SG}(crs,vk,m,\zeta_1,\mathbf{u},\mathbf{s})$;
    $\zeta \leftarrow (\zeta_1,\zeta_2)$;
    $\alpha \leftarrow \mathsf{Vrf}(crs,vk,m,\sigma)$;
    If well-formedness conditions hold
        $(i)$ $\mathbf{s} = \mathsf{Sign}(crs,vk,\mathbf{u},sk;\eta)$
        $(ii)$ $vk = \mathsf{KG}(crs,sk)$
      set $\alpha \leftarrow 0 \wedge \sigma \leftarrow 0$;
    Output $(\alpha,\sigma,\zeta)$.
```

```
$\mathbf{LiteEquiv}_{\mathcal{A}}(\lambda,0)$
    $(crs,\tau) \leftarrow \widetilde{\mathsf{CRS}}(1^{\lambda})$;
    $b' \leftarrow \mathcal{A}^{\mathcal{O}_0(crs,\tau,\cdot,\cdot)}(crs)$;
    Return $b'$.

$\mathcal{O}_0(crs,\tau,vk,m)$
    $(\mathbf{u},\xi_1) \leftarrow \widetilde{\mathsf{Blind}}(crs,vk,\tau)$;
    $(\mathbf{s},\eta,sk) \leftarrow \mathcal{A}(\mathbf{u})$; $\xi_1 \leftarrow (\xi_1,\mathbf{s},\eta,sk)$;
    $(\sigma,\xi_2) \leftarrow \widetilde{\mathsf{SG}}(crs,\tau,vk,sk,m)$;
    $\zeta \leftarrow \widetilde{\mathsf{Equiv}}(\xi_1,\xi_2)$;
    $\alpha \leftarrow \mathsf{Vrf}(crs,vk,m,\sigma)$;
    If well-formedness conditions hold
        $(i)$ $\mathbf{s} = \mathsf{Sign}(crs,vk,\mathbf{u},sk;\eta)$
        $(ii)$ $vk = \mathsf{KG}(crs,sk)$
      set $\alpha \leftarrow 0 \wedge \sigma \leftarrow 0$;
    Output $(\alpha,\sigma,\zeta)$.
```

– LITE-UNFORGEABILITY: For all PPT $\mathcal{A}$, $\Pr[\mathbf{LiteUnforge}_{\mathcal{A}}(\lambda)=1] \overset{c}{\approx} 0$ where $\mathbf{LiteUnforge}_{\mathcal{A}}$

is defined as follows

```
**LiteUnforge**$_{\mathcal{A}}(\lambda)$
    $crs \leftarrow \mathsf{CRS}(1^{\lambda}); (vk, sk) \leftarrow \mathsf{KG}(crs); state := \emptyset; i := 0;$
    while $i < \ell$
        $(m_i, \zeta_i, state) \leftarrow \mathcal{A}(state, crs, vk); (\mathbf{u}_i, \zeta_i) \leftarrow \mathsf{Blind}(crs, vk, m_i);$
        $(\mathbf{s}_i, \eta_i) \leftarrow \mathsf{Sign}(crs, vk, \mathbf{u}_i, sk); state \leftarrow state \| \mathbf{s}_i; i \leftarrow i+1;$
    $((m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_{k+1}, \sigma_{k+1})) \leftarrow \mathcal{A};$
    Return 1 iff all the following three conditions hold
        $(i)\ \sum_{i=1}^{\ell} \beta_i \leq k \quad (ii)\ m_i \neq m_j$ for all $1 \leq i < j \leq k+1$
        $(iii)\ \mathsf{Vrf}(crs, vk, m_i, \sigma_i) = 1$ for all $1 \leq i \leq k+1$.
```

□

Remark 4.3.2. For two-move signature generation protocols as in Figure 4.1, equivocality implies lite-equivocality because the adversary in the lite-equivalently is much weaker.

Recall that $(\sigma, \xi_2) \leftarrow \widetilde{\mathsf{SG}}(crs, \tau, vk, sk, m)$. The adversary might be able to generate different *sk*'s consistent to a single *vk*, and embed some information related to a *sk* into a signature string σ; the adversary later by matching σ with the corresponding *sk* identify the involved user. This types of attacks first identified by Horvitz and Katz, and please refer to [HK07] for more details. It seems difficult to argue that a scheme is lite-equivocal. To facilitate the security analysis, we here only consider algorithm KG such that for each *vk* there exists only a single *sk*. We also note that if *sk* is not used in $\widetilde{\mathsf{SG}}$ algorithm, then such requirement is not needed.

4.3.2 Generic Construction for Lite Blind Signatures

In this subsection, we present two lite blind signature constructions. The first construction is generic and is based on the blind signature design of [Fis06] whereas the second is a concrete number theoretic construction that is based on [Oka06].

Our first construction is based on [Fis06]; the main difference here is that we need the equivocality property (the original design employed two encryption steps for the user that are non-equivocal); in our setting, it is sufficient to have just one equivocal commitment (that is not extractable) in the first stage and then employ an extractable commitment in the second (that is not equivocal). Refer to the signature generation protocol in Figure 4.2: the 2MBS.CRS algorithm produces $crs = \langle crs_{\mathsf{EQC}}, crs_{\mathsf{EXC}}, crs_{\mathsf{NIZK}} \rangle$; here EQC and EXC are two commitments under crs_{EQC} and crs_{EXC} respectively; NIZK is an NIZK argument under crs_{NIZK}; SIG is a signature scheme. The language $\mathcal{L}_R \overset{\mathsf{def}}{=} \{x | (x, w) \in R\}$ where $R \overset{\mathsf{def}}{=} \{(crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \zeta_1, \zeta_2) | \mathbf{u} = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m; \zeta_1) \wedge \mathsf{SIG.Vrf}(vk, \mathbf{u}, \mathbf{s}) = 1 \wedge E = \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, \mathbf{u}, \mathbf{s}; \zeta_2)\}$. The algorithm 2MBS.Vrf given a message m and signature σ operates as follow: parse σ into E and ς, and check that $\mathsf{NIZK.Vrf}((crs, vk, E, m), \varsigma) =^? 1$.

Theorem 4.3.3. *The two-move signature generation protocol in Figure 4.2 is a lite blind signature. It satisfies lite-unforgeability provided that (i)* SIG *is unforgeable, (ii)* EQC *is binding, (iii)* EXC *is extractable, and (iv)* NIZK *is sound. It satisfies lite-equivocality provided that (i)* EQC *is equivocal, (ii)* EXC *is hiding, and (iii)* NIZK *is non-erasure zero-knowledge.*

Proof. **(I)** We first provide a sketch to show that the protocol in Figure 4.2 satisfies lite-unforgeability.

Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to attack the

$crs = \langle crs_{\mathsf{EQC}}, crs_{\mathsf{EXC}}, crs_{\mathsf{NIZK}} \rangle$

$\boxed{S}$		$\boxed{U}$
Verification Key = $\langle vk \rangle$		Verification Key = $\langle vk \rangle$
Signing Key = $\langle sk \rangle$		Plaintext = $\langle m \rangle$
		$(\mathbf{u}, \zeta_1) \leftarrow \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m)$
	$\xleftarrow{\mathbf{u}}$	
$(\mathbf{s}, \eta) \leftarrow \mathsf{SIG.SG}(vk, sk, \mathbf{u})$		
	$\xrightarrow{\mathbf{s}}$	
		$\mathsf{SIG.Vrf}(vk, \mathbf{u}, \mathbf{s}) \overset{?}{=} 1$
		$(E, \zeta_2) \leftarrow \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, \mathbf{u}, \mathbf{s})$
		$(\varsigma, \zeta_4) \leftarrow \mathsf{NIZK.Prv}((crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \zeta_1, \zeta_2);$
		$: \mathbf{u} = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m; \zeta_1)$
		$\wedge \mathsf{SIG.Vrf}(vk, \mathbf{u}, \mathbf{s}) = 1$
		$\wedge E = \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, \mathbf{u}, \mathbf{s}; \zeta_2))$
		$\sigma \leftarrow E \| \varsigma$

Figure 4.2: A generic signature generation protocol.

signature scheme SIG to produce a forgery. Note that $\mathcal{B}$ is given vk, and is allowed to query the signing oracle with $\mathbf{u}_i$ and obtain $\mathbf{s}_i$ such that $\mathsf{SIG.Vrf}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. $\mathcal{B}$'s goal is to obtain a pair $\langle \mathbf{u}^*, \mathbf{s}^* \rangle$ where $\mathbf{u}^*$ is not queried.

$\mathcal{B}$ runs a copy of $\mathcal{A}$ internally, and supplies $\mathcal{A}$ with crs and vk; note that $\mathcal{A}$ is allowed to query $(m_i, \zeta_{1,i})$ where $\mathbf{u}_i = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m_i; \zeta_{1,i})$; given such query, $\mathcal{B}$ queries his signing oracle with $\mathbf{u}_i$, obtains $\mathbf{s}_i$ and then gives such $\mathbf{s}_i$ to $\mathcal{A}$, where $\mathsf{SIG.Vrf}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. At some point, $\mathcal{A}$ produces a pair $\langle m^*, \sigma^* \rangle$ where $\sigma^* = E^* \| \varsigma^*$, and $\mathsf{NIZK.Vrf}(crs, vk, E^*, m^*; \varsigma^*) = 1$. Since NIZK is sound, there exist $\langle \mathbf{u}^*, \mathbf{s}^*, \zeta_1^*, \zeta_2^* \rangle$ such that $((crs, vk, E^*, m^*), (\mathbf{u}^*, \mathbf{s}^*, \zeta_1^*, \zeta_2^*)) \in R$.

Given that commitment scheme EQC is binding, we have only negligible probability to find $m_i' \neq m_i$ such that $\mathbf{u}_i = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m_i; \zeta_{1,i}) = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, m_i'; \zeta_{1,i}')$. Therefore m^* cannot be based on any queried $\mathbf{u}_i$. By using the extractable trapdoor τ_{EXC} of commitment scheme EXC, $\mathcal{B}$ can extract $\mathbf{u}^*$ and $\mathbf{s}^*$ from E. Note that $\mathsf{SIG.Vrf}(vk, \mathbf{u}^*, \mathbf{s}^*) = 1$, and $\mathbf{u}^*$ has never been queried, $\mathcal{B}$ can obtain a forgery $\langle \mathbf{u}^*, \mathbf{s}^* \rangle$ for SIG.

(II) Secondly, we prove that the protocol in Figure 4.2 satisfies lite-equivocality. To finish the proof, we need to construct a tuple of algorithms $\widetilde{\mathsf{CRS}}, \widetilde{\mathsf{Blind}}, \widetilde{\mathsf{SG}}, \widetilde{\mathsf{Equiv}}$ as required in Definition 4.3.1.

- $(crs, \tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$ is described as follows: $crs = \langle crs_{\mathsf{EQC}}, crs_{\mathsf{EXC}}, crs_{\mathsf{nizk}} \rangle$, $\tau = \langle \tau_{\mathsf{EQC}}, \tau_{\mathsf{nizk}} \rangle$ where τ_{EQC} is the trapdoor for equivocal commitment scheme EQC, and τ_{nizk} the trapdoor for non-erasure zero-knowledge NIZK scheme NIZK.
- $\widetilde{\mathsf{Blind}}$ is defined by $\mathsf{EQC}.\widetilde{\mathsf{Com}}$, i.e., $(\mathbf{u}, \xi_1) \leftarrow \mathsf{EQC}.\widetilde{\mathsf{Com}}(crs_{\mathsf{EQC}}, \tau_{\mathsf{EQC}})$.
- $\widetilde{\mathsf{SG}}$ is defined by $\mathsf{NIZK}.\widetilde{\mathsf{Prv}}$, i.e., $(\varsigma, \xi_2) \leftarrow \mathsf{NIZK}.\widetilde{\mathsf{Prv}}(crs_{\mathsf{NIZK}}, \tau_{\mathsf{NIZK}}, vk, E, m)$, where $(E, \zeta_2) \leftarrow \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, \mathbf{u}, \mathbf{s})$. We note that here sk is not used in $\widetilde{\mathsf{SG}}$ algorithm.
- $\widetilde{\mathsf{Equiv}}$ is defined by $\mathsf{EQC}.\widetilde{\mathsf{Equiv}}$ and $\mathsf{NIZK}.\widetilde{\mathsf{Equiv}}$, i.e., $\zeta_1 \leftarrow \mathsf{EQC}.\widetilde{\mathsf{Equiv}}(\mathbf{u}, \xi_1, m)$, $\zeta_4 \leftarrow \mathsf{NIZK}.\widetilde{\mathsf{Equiv}}(\varsigma, \xi_2, (\mathbf{u}, \mathbf{s}, \zeta_1, \zeta_2))$, and $\zeta \leftarrow (\zeta_1, \zeta_2, \zeta_4)$.

Based on the hiding property of EQC and EXC, and zero-knowledge property of NIZK, the protocol transcript $\mathbf{u}$ and the signature $\langle E, \varsigma \rangle$ cannot be distinguished by any adversary. In addition, based on the equivocality of EQC and the non-erasure property of NIZK, the internal state ζ cannot be distinguished.

□

4.3.3 *Concrete Construction for Lite Blind Signature from Okamoto Signatures*

In Figure 4.3 we present a lite blind signature Oka2MBS.{CRS, KG, Blind, Sign, SG, Vrf} that uses the 2SDH assumption and is based on Okamoto's blind signature scheme [Oka06]; the contribution here is our proof (cf. Theorem 4.3.4 below) that this design is in fact equivocal (instead of merely blind as shown in [Oka06]). In this scheme the CRS algorithm produces $crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \psi, u_2, v_2 \rangle$, where $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, $\mathbb{G}_1, \mathbb{G}_2$ are groups of order p, the KG algorithm produces a key-pair $vk = \langle X \rangle, sk = \langle x \rangle$ such that $X = g_2^x$, and the algorithm Vrf given a message m and signature $\sigma = \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$, responds as follows: check that $m, \beta \in \mathbb{Z}_p$, $\varsigma, V_1 \in \mathbb{G}_1$, $\alpha, V_2 \in \mathbb{G}_2$, $\varsigma \neq 1$, $\alpha \neq 1$ and $\hat{e}(\varsigma, \alpha) = \hat{e}(g_1, g_2^m u_2 v_2^\beta)$, $\hat{e}(V_1, \alpha) = \hat{e}(\psi(X), X) \cdot \hat{e}(g_1, V_2)$.

Theorem 4.3.4. *The two-move protocol of Figure 4.3 is a lite blind signature. It satisfies lite-unforgeability under the 2SDH assumption. It satisfies lite-equivocality unconditionally in the CRS model.*

Proof. **(I)** First, we show the proof for lite-unforgeability. We note that the proof here is very similar to the unforgeability proof of Okamoto signature (refer to the proof of theorem 2, page 14, in [Oka06]). Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to break the 2SDH assumption.

We consider two types of forgers, Type-1 forger and Type-2 forger. Let $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \psi, u_2, v_2, X \rangle$ be the verification key for the forger $\mathcal{A}$ and $v_2 = g_2^z$. Now $\mathcal{A}$ queries with $\langle m_i, t_i, s_i \rangle$ and is responded with $\langle Y_i, l_i, r_i \rangle$ for $i = 1, \ldots, L$. The two types of forgers are:

– Type-1 forger: outputs a forgery $\langle m^*, \sigma^* \rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^* \rangle$, and $m^* + \beta^* z \not\equiv$

$$crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \psi, u_2, v_2 \rangle$$

$$\begin{array}{lcl}
\boxed{S} & & \boxed{U} \\
vk = \langle X = g_2^x \rangle & & vk = \langle X = g_2^x \rangle \\
sk = \langle x \rangle & & m \in \mathbb{Z}_p \\
\hline
 & & t,s \overset{\$}{\leftarrow} \mathbb{Z}_p;\ W \leftarrow g_1^{mt} u_1^t v_1^{st} \\
 & \xleftarrow{\quad W \quad} & \\
r,l \overset{\$}{\leftarrow} \mathbb{Z}_p \text{ s.t. } x+r \neq 0 & & \\
Y \leftarrow (W v_1^l)^{\frac{1}{x+r}} & & \\
 & \xrightarrow{\quad Y,l,r \quad} & \\
 & & f,h \overset{\$}{\leftarrow} \mathbb{Z}_p;\ \varsigma \leftarrow Y^{\frac{1}{ft} \bmod p} \\
 & & \alpha \leftarrow X^f g_2^{fr};\ \beta \leftarrow s + \frac{l}{t} \bmod p \\
 & & V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h;\ V_2 \leftarrow X^{fh+r} g_2^{frh} \\
 & & \sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle \\
\hline
\end{array}$$

Figure 4.3: Signature generation protocol based on Okamoto digital signature.

$m_i + \beta_i z$ for $i = 1, \ldots, L$.

– Type-2 forger: outputs a forgery $\langle m^*, \sigma^* \rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^* \rangle$, and $m^* + \beta^* z \equiv m_i + \beta_i z$ for $i = 1, \ldots, L$. Note that in this case, $\beta^* \neq \beta_i$ because $m^* \neq m_i$.

$\mathcal{B}$ operates as follows:

1. $\mathcal{B}$ is given $\langle g_1, g_2, A, B, C_1, \ldots, C_L, a_1, \ldots, a_L, b_1, \ldots, b_L \rangle$ where $A = g_2^x, B = g_2^y, C_i = g^{\frac{y+b_i}{x+a_i}}$, $i = 1, \ldots, L$.

2. $\mathcal{B}$ randomly selects $c_{\text{type}} \in \{1, 2\}$.

3. If $c_{\text{type}} = 1$: $\mathcal{B}$ randomly selects $z \in \mathbb{Z}_p$, and $\mathcal{B}$ sets $X \leftarrow A = g_2^x$, $u_2 \leftarrow B = g_2^y$, and $v_2 \leftarrow g_2^z$; $\mathcal{B}$ then gives $\langle g_1, g_2, u_2, v_2, X \rangle$ to $\mathcal{A}$ as the verification key. When $\mathcal{A}$ queries with

$\langle m_i, t_i, s_i \rangle$, $\mathcal{B}$ computes $\beta_i \leftarrow \frac{b_i - m_i}{z}$, $r_i \leftarrow a_i$, $\varsigma \leftarrow \psi(C_i) = g_1^{\frac{y+b_i}{x+a_i}} = g_i^{\frac{m_i+y+\beta_i z}{x+r_i}}$, and $Y_i \leftarrow \varsigma_i^{t_i}$, $l_i \leftarrow t_i(\beta_i - s_i)$. Then $\mathcal{B}$ returns $\langle Y_i, l_i, r_i \rangle$. When $\mathcal{A}$ outputs a successful forgery $\langle m^*, \sigma^* \rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^* \rangle$, $\mathcal{B}$ checks $m^* + \beta^* z \not\equiv m_i + \beta_i z$ for all $i = 1, \ldots, L$. If not, $\mathcal{B}$ outputs `failure` and aborts. Else $\mathcal{B}$ sets $b^* \leftarrow m^* + \beta^* z$, and outputs $\langle \varsigma^*, \alpha^*, b^*, V_1^*, V_2^* \rangle$.

4. If $c_{\text{type}} = 2$: $\mathcal{B}$ randomly selects $x', y' \in \mathbb{Z}_p$ and computes $X \leftarrow g_2^{x'}$, $u_2 \leftarrow g_2^{y'}$, $v_2 \leftarrow A = g_2^x$. $\mathcal{B}$ then gives $\langle g_1, g_2, u_2, v_2, X \rangle$ to $\mathcal{A}$ as the verification key. The simulation of signing oracle can be achieved since $\mathcal{B}$ knows x'. When $\mathcal{A}$ outputs a successful forgery $\langle m^*, \sigma^* \rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^* \rangle$, $\mathcal{B}$ computes $z^* \leftarrow \frac{m_i - m^*}{\beta^* - \beta_i}$ for all $i = 1, \ldots, L$ such that $\beta_i \neq \beta^*$, and checks whether $A = g_2^{z^*}$. If it holds, $z^* = x$. $\mathcal{B}$ then outputs $\langle \varsigma, \alpha, d, V_1, V_2 \rangle$ where $c, d, \eta \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\varsigma \leftarrow (\psi(B))^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = (\psi(g_2)^y)^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = (g_1^y)^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = g_1^{\frac{y+d}{x+c}}$, $\alpha \leftarrow g_2^c$, $V_1 \leftarrow \psi(A) g_1^{\eta}$, $V_2 \leftarrow A^{\eta+c} g_2^{c\eta}$.

This completes the description of $\mathcal{B}$.

(II) Here we show the proof for lite-equivocality.

- $(crs, \tau) \leftarrow \widetilde{\mathsf{CRS}}(1^\lambda)$ is described as follows: $crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \psi, u_2, v_2 \rangle$, $\tau = \langle \tau_{u_1}, \tau_{v_1} \rangle$ such that $u_1 = g_1^{\tau_{u_1}}, v_1 = g_1^{\tau_{v_1}}$.
- $(vk, sk) \leftarrow \mathsf{KG}(crs)$, where $vk = X = g_2^x$ and $sk = x$. For each vk there exists a single sk.
- $\widetilde{\mathsf{Blind}}$ is defined as follows: select $\tilde{m}, \tilde{t}, \tilde{s} \overset{\$}{\leftarrow} \mathbb{Z}_p$, and compute $W \leftarrow g_1^{\tilde{m}\tilde{t}} u_1^{\tilde{t}} v_1^{\tilde{s}\tilde{t}}$.
- $\widetilde{\mathsf{SG}}$ is defined based on only sk but not any trapdoor of the CRS as follows: compute $\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$, where $\varsigma \leftarrow (g_1^m u_1 v_1^{\beta})^{\frac{1}{\widehat{f}x+\widehat{r}}}$, $\alpha \leftarrow g_2^{\widehat{f}x+\widehat{r}}$, $V_1 \leftarrow \psi(X)^{\frac{1}{\widehat{f}}} g_1^{\widehat{h}}$, $V_2 \leftarrow X^{\widehat{f}\widehat{h}+\frac{\widehat{r}}{\widehat{f}}} g_2^{\widehat{r}\widehat{h}}$, and $\beta, \widehat{f}, \widehat{r}, \widehat{h} \overset{\$}{\leftarrow} \mathbb{Z}_p$.
- $\widetilde{\mathsf{Equiv}}$ is described as follows.

First we compute t, s such that $W = g_1^{\widetilde{m}\widetilde{t}} u_1^{\widetilde{t}} v_1^{\widetilde{s}\widetilde{t}} = g_1^{mt} u_1^t v_1^{st}$ and $\beta = s + \frac{l}{t}$, i.e., based on equations $(m + \tau_{u_2} + s\tau_{v_2})t = (\widetilde{m} + \tau_{u_2} + \widetilde{s}\tau_{v_2})\widetilde{t}$ and $\beta = s + \frac{l}{t}$. Then we have $t \leftarrow \frac{(\widetilde{m}+\tau_{u_2}+\widetilde{s}\tau_{v_2})\widetilde{t}+\tau_{v_2}l}{m+\tau_{u_2}+\beta\tau_{v_2}}$, and $s \leftarrow \beta - l\frac{m+\tau_{u_2}+\beta\tau_{v_2}}{(\widetilde{m}+\tau_{u_2}+\widetilde{s}\tau_{v_2})\widetilde{t}+\tau_{v_2}l}$.

Second we compute f, h such that $\varsigma = (g_1^m u_1 v_1^{\beta})^{\frac{1}{\widehat{fx+r}}} = (g_1^m u_1 v_1^{s+\frac{l}{t}})^{\frac{1}{fx+fr}}$, $\alpha = g_2^{\widehat{fx+r}} = g_2^{fx+fr}$, $V_1 = \psi(X)^{\frac{1}{\widehat{f}}} g_1^{\widehat{h}} = \psi(X)^{\frac{1}{f}} g_1^h$, and $V_2 = X^{\widehat{fh}+\frac{\widehat{r}}{\widehat{f}}} g_2^{\widehat{rh}} = X^{fh+r} g_2^{frh}$. Reduce the redundance, and we can compute f, h based on just equations $\widehat{fx+r} = f(x+r)$, and $\frac{x}{\widehat{f}} + \widehat{h} = \frac{x}{f} + h$. We have $f \leftarrow \frac{\widehat{fx+r}}{x+r}$ and $h \leftarrow \frac{x}{\widehat{f}} + \widehat{h} - \frac{x(x+r)}{\widehat{fx+r}}$;

We note that the transcript W, and the signature σ, and also the internal states are identically distributed as those in the real protocol execution. $\square$

4.4 *Building Block: Single-Verifier/Prover Zero-Knowledge*

4.4.1 *Single-Verifier Zero-Knowledge*

We first describe the functionalities $\mathcal{F}_{\mathrm{SVZK}}$ in Figure 4.4.

Next we investigate how to realize the functionality $\mathcal{F}_{\mathrm{SVZK}}$ against adaptive adversaries for some relation R. We proceed as follows: first given $(x, w) \in R$, we will have the prover commit the witness w into value C; given that we have a single verifier in our setting (the signer), we find it convenient to base the commitment scheme that we employ based on the mixed commitment primitive of [DN02, Nie03]; using such a commitment one then can employ a non-erasure Sigma protocol to show the consistency of the witness between the commitment C and the statement x by performing a proof of language membership; finally to defend against a dishonest verifier, the Sigma protocol will have to be strengthened so

Functionality $\mathcal{F}^R_{\text{SVZK}}$

$\mathcal{F}^R_{\text{SVZK}}$, parameterized by a binary relation R, interacts with a single verifier V and a set $\mathcal{P}$ of provers, adversary $\mathcal{S}$. For each input or message, the functionality verifies that $sid = (V, sid')$ for some sid', and ignores it otherwise.

Proof:

- Upon receiving $(\textsc{svzkProve}, P, sid, \langle x, w\rangle)$ from a prover $P \in \mathcal{P}$, forward $(\textsc{svzkLeakProve}, P, sid, \langle x, \phi\rangle)$ to the adversary $\mathcal{S}$, and record $\langle x, w\rangle$ into $\mathsf{history}_P$, where $\phi = 1$ if $(x, w) \in R$, and $\phi = 0$ otherwise. Ignore future $(\textsc{svzkProve}, P, \ldots)$.
- Upon receiving $(\textsc{svzkInflProve}, V, sid, P)$ from the adversary $\mathcal{S}$, output $(\textsc{svzkProveReturn}, V, sid, \langle P, x, \phi\rangle)$ to the verifier V. Ignore future $(\textsc{svzkInflProve}, V, sid, P)$.

Corruption:

- Upon receiving $(\textsc{Corrupt}, P, sid)$ from the adversary $\mathcal{S}$, return it $(\textsc{CorruptReturn}, P, sid, \mathsf{history}_P)$.

  After the successful corruption of P, if no output $(\textsc{svzkProveReturn}, P, sid, \ldots)$ was returned to party V yet, upon receiving $(\textsc{Patch}, P, sid, \langle x', w'\rangle)$ from the adversary $\mathcal{S}$, output $(\textsc{svzkProveReturn}, V, sid, \langle P, x', \phi'\rangle)$ to party V, where $\phi' = 1$ if $(x', w') \in R$, and $\phi' = 0$ otherwise.

Figure 4.4: Single-verifier zero-knowledge functionality $\mathcal{F}_{\text{SVZK}}$.

that it can be simulated without knowing the witness; this e.g., can be based on Damgård's trick [Dam00] (or alternatively one may use an OR-proof).

An SVZK instantiation is presented in Figure 4.5: (1) it is based on a three-move mixed-commitment in the single-verifier setting, and EQC is an equivocal commitment which is used to strengthen the mixed-commitment by binding the committer from the beginning (please refer to section 9.4 in [Nie03] for more discussion); (2) a non-erasure Sigma protocol 3MZK = 3MZK.{Comt, Coins, Resp, Vrf} is used to show the consistency of the relation R and the strengthened mixed-commitment, i.e. for relation $R' = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R \wedge E = \mathsf{MIX.Com}(K, w; \zeta) \wedge C_0 = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, w; \eta)\}$, and a combination of the Sigma protocol with the committing step of mixed-commitment can be found in Figure 4.6; (3) COM is an equivocal commitment schemes for applying Damgård trick to the non-erasure Sigma protocol.

Theorem 4.4.1. *Given two equivocal commitment schemes* EQC *and* COM, *and a non-erasure zero-knowledge Sigma protocol* 3MZK.{Comt, Coins, Resp, Vrf}, *the SVZK protocol* $\pi_{\Sigma(\mathsf{SVZK})}$ *in Figure 4.5 securely realizes* $\mathcal{F}^R_{\mathrm{SVZK}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid model.*

4.4.2 Single-Prover Zero-Knowledge

We describe the functionality $\mathcal{F}_{\mathrm{SPZK}}$ Figure 4.7.

SPZK instantiations can also be constructed against adaptive corruption based on mix-commitments and non-erasure Sigma protocols. However, we surprisingly find that a weaker version of SPZK, called leaking SPZK, is enough for our UC blind signature protocol. We note that leaking SPZK protocols can be much more efficiently constructed,

Protocol $\pi_{\Sigma(\mathrm{SVZK})}$ in the $\mathcal{F}_{\mathrm{CRS}}$-Hybrid World

CRS generation: Set $crs \leftarrow (crs_{\mathsf{MIX}}, crs_{\mathsf{EQC}}, crs_{\mathsf{COM}})$ where $crs_{\mathsf{MIX}} \leftarrow \mathsf{MIX.CRS}(1^\lambda)$ and $crs_{\mathsf{EQC}} \leftarrow \mathsf{EQC.CRS}(1^\lambda)$ and $crs_{\mathsf{COM}} \leftarrow \mathsf{COM.CRS}(1^\lambda)$.

Proof: Please refer to Figure 4.4 for a pictorial presentation.

When party P is invoked with input (SPZKPROVE, $P, sid, \langle x, w\rangle$) by $\mathcal{Z}$, it verifies that $sid = (V, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R$, it computes $(C_0, \eta) \leftarrow \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, w)$ and $K_1 \leftarrow \mathsf{MIX.Coins}()$ and $(C_1, \mu_1) \leftarrow \mathsf{MIX.Com}(crs_{\mathsf{MIX}}, K_1)$, and then sends message (SVZK$_1$, $sid, \langle C_0, C_1\rangle$) to party V.

When party V is invoked with incoming SVZK$_1$ message, it selects $K_2 \leftarrow \mathsf{MIX.Coins}()$, and sends message (SVZK$_2$, sid, K_2) to party P.

When party P is invoked with incoming SVZK$_2$ message, it computes $K \leftarrow \mathsf{MIX.NewCRS}(crs_{\mathsf{MIX}}, K_1, K_2)$, $(E, \zeta) \leftarrow \mathsf{MIX.Com}(K, w)$, $(a, r_a) \leftarrow \mathsf{3MZK.Comt}((x, C_0, E), (w, \eta, \zeta))$ and $(c, r) \leftarrow \mathsf{COM.Com}(crs_{\mathsf{COM}}, a)$, and sends message (SVZK$_3$, $sid, \langle K_1, \mu_1, x, E, c\rangle$) to party P.

When party V is invoked with incoming SVZK$_3$ message, it verifies if $\mathsf{MIX.Vrf}(crs_{\mathsf{MIX}}, C_1, K_1, \mu_1) = 1$ holds. If the equation holds, then it selects $e \leftarrow \mathsf{3MZK.Coins}()$, and sends message (SVZK$_4$, sid, e) to party P.

When party P is invoked with incoming SVZK$_4$ message, it computes $z \leftarrow \mathsf{3MZK.Resp}((x, C_0, E), (w, \eta, \zeta), r_a, e)$, and sends message (SVZK$_5$, $sid, \langle a, r, z\rangle$) to party V.

When party V is invoked with incoming SVZK$_5$ message, it verifies $c = \mathsf{COM.Com}(crs_{\mathsf{COM}}, a; r)$ and $\mathsf{Vrf}((x, C_0, E), a, e, z) = 1$; if both hold, then it returns message (SVZKPROVERETURN, $V, sid, \langle P, x, 1\rangle$) to $\mathcal{Z}$. Otherwise return (SVZKPROVERETURN, $V, sid, \langle P, x, 0\rangle$) to $\mathcal{Z}$.

Figure 4.5: Single-verifier zero-knowledge protocol $\pi_{\Sigma(\mathrm{SVZK})}$ for relation R.

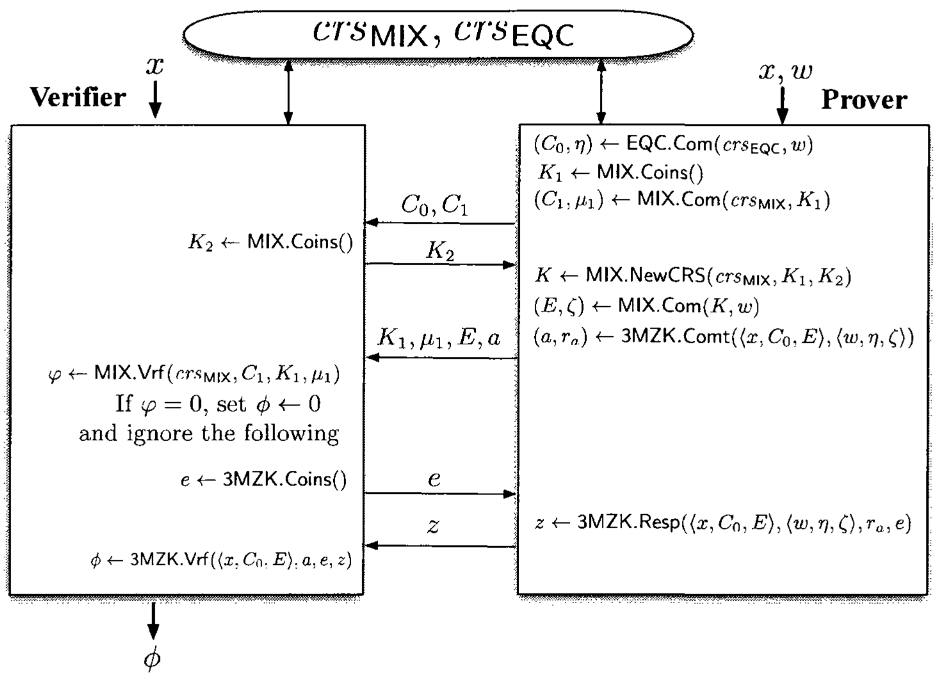

$crs_{\mathsf{MIX}}, crs_{\mathsf{EQC}}$
x
Verifier
x, w
Prover
$(C_0, \eta) \leftarrow \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, w)$
$K_1 \leftarrow \mathsf{MIX.Coins}()$
$(C_1, \mu_1) \leftarrow \mathsf{MIX.Com}(crs_{\mathsf{MIX}}, K_1)$
C_0, C_1
$K_2 \leftarrow \mathsf{MIX.Coins}()$
K_2
$K \leftarrow \mathsf{MIX.NewCRS}(crs_{\mathsf{MIX}}, K_1, K_2)$
$(E, \zeta) \leftarrow \mathsf{MIX.Com}(K, w)$
$(a, r_a) \leftarrow \mathsf{3MZK.Comt}(\langle x, C_0, E\rangle, \langle w, \eta, \zeta\rangle)$
K_1, μ_1, E, a
$\varphi \leftarrow \mathsf{MIX.Vrf}(crs_{\mathsf{MIX}}, C_1, K_1, \mu_1)$
If $\varphi = 0$, set $\phi \leftarrow 0$
and ignore the following
$e \leftarrow \mathsf{3MZK.Coins}()$
e
z
$z \leftarrow \mathsf{3MZK.Resp}(\langle x, C_0, E\rangle, \langle w, \eta, \zeta\rangle, r_a, e)$
$\phi \leftarrow \mathsf{3MZK.Vrf}(\langle x, C_0, E\rangle, a, e, z)$
ϕ

Figure 4.6: A combination of a committing step of a mixed commitment MIX with a non-erasure zero-knowledge Sigma protocol 3MZK for relation $R' = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R \wedge E = \mathsf{MIX.Com}(K, w; \eta) \wedge C_0 = \mathsf{EQC.Com}(crs_{\mathsf{EQC}}, w; \eta)\}$ in the single-verifier setting.

and can be based on extractable commitments (which are more efficient than mix commitments) plus Sigma protocols. Please refer to Section 4.7

4.5 Blind Signature Functionality

In this section we present our design methodology for constructing UC-blind signatures secure against adaptive adversaries, i.e., the protocol obtained by our method can UC-realize the blind signature functionality $\mathcal{F}_{\mathrm{BS}}$ (defined in Figure 4.9). A previous formaliza-

Functionality $\mathcal{F}^R_{\mathrm{SPZK}}$

$\mathcal{F}^R_{\mathrm{SPZK}}$, parameterized by a binary relation R, interacts with adversary $\mathcal{S}$, and a single prover P and a set $\mathcal{V}$ of verifiers. For each input or message, the functionality verifies that $sid = (P, sid')$ for some sid', and ignores it otherwise.

Proof:

- Upon receiving ($\textsc{spzkProve}, P, sid, \langle V, x, w\rangle$) from the prover P, where $V \in \mathcal{V}$, forward ($\textsc{spzkLeakProve}, P, sid, \langle V, x, \phi\rangle$) to the adversary $\mathcal{S}$, and record $\langle V, x, w\rangle$ into $\mathsf{history}_P$, where $\phi = 1$ if $(x, w) \in R$, and $\phi = 0$ otherwise. Ignore future ($\textsc{spzkProve}, P, sid, \langle V, \ldots\rangle$).
- Upon receiving ($\textsc{spzkInflProve}, V, sid$) from the adversary $\mathcal{S}$, output ($\textsc{spzkProveReturn}, V, sid, \langle V, x, \phi\rangle$) to party V. Ignore future ($\textsc{spzkInflProve}, V, sid$).

Corruption:

- Upon receiving ($\textsc{Corrupt}, P, sid$) from the adversary $\mathcal{S}$, return $\mathcal{S}$ ($\textsc{CorruptReturn}, P, sid, \mathsf{history}_P$).

  After the successful corruption of P, if no output ($\textsc{spzkProveReturn}, V, sid, \ldots$) was returned to party V yet, upon receiving ($\textsc{Patch}, P, sid, \langle V, x', w'\rangle$) from the adversary $\mathcal{S}$, output ($\textsc{spzkProveReturn}, V, sid, \langle x', \phi'\rangle$) to party V, where $\phi' = 1$ if $(x', w') \in R$, and $\phi' = 0$ otherwise.

Figure 4.7: Single-prover zero-knowledge functionality $\mathcal{F}_{\mathrm{SPZK}}$.

tion of the blind signature primitive in the UC setting was given by [Fis06]. In our definition of the ideal functionality we give an explicit description of how corruption is handled; as we deal with adaptive adversaries, being explicit in the way that the ideal functionality interacts with the adversary during corruption makes the security model crisper. One other difference is that our $\mathcal{F}_{\mathrm{BS}}$ does not require strong unforgeability from the underlying signing mechanism; this makes the presentation more general as strong unforgeability is not necessary for many applications of the blind signature primitive. Further, protocols realizing the functionality of [Fis06] requires a single "global trapdoor" that enables the functionality to produce a signature for a given message that will be valid for any given public-key; while this can be handy in the security proof, it is not a mandatory requirement for a UC-blind signature (which may allow for a different trapdoor to be used by the functionality in each occasion); we reflect this in our ideal functionality by allowing the adversary in the corrupted signer setting to "patch" the ideal functionality with a different signing key for each user. In a blind signature session we allow for a single signer (whose identity is hard-coded into the session identifier *sid*) and a multitude of users.

We define the ideal functionality $\mathcal{F}_{\mathrm{BS}}$ in Figure 4.9. Please also refer to a pictorial version in Figure 4.10. Here we provide the definition of UC blind signature protocol and show that protocols realizing such $\mathcal{F}_{\mathrm{BS}}$ will be secure in some existing model e.g. of [Oka06, HKKL07].

As defined in Definition 4.2.1, a blind signature scheme is a tuple $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{CRS}, \mathsf{KG}, U, S, \mathsf{Vrf}\}$ where U, S is an interactive protocol between the user and the signer. In each round of the protocol the user and the signer exchange messages (note that U, S may extend to many rounds). Given such protocol we can also define a Sign algorithm by collapsing the interactive protocol U, S into a non-interactive algorithm (that simulates both parties). Given such a scheme each party in the UC-framework will execute a program $\pi_{\Sigma(\mathsf{BS})}$ that is described in Figure 4.11.

Definition 4.5.1. A blind signature scheme $\Sigma(\mathsf{BS}) = \mathsf{BS}.\{\mathsf{CRS}, \mathsf{KG}, U, S, \mathsf{Vrf}\}$ is UC-secure in the CRS model if $\pi_{\Sigma(\mathsf{BS})}$ realizes the ideal functionality $\mathcal{F}_{\mathrm{BS}}$ in Figure 4.9 in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid

Functionality $\mathcal{F}_{\mathrm{BS}}$

$\mathcal{F}_{\mathrm{BS}}$ interacts with the adversary $\mathcal{S}$, and a set $\mathcal{P}$ of parties (including a single signer S, a set $\mathcal{U}$ of users, and a set $\mathcal{V}$ of verifiers). Initial $\mathsf{history} := \emptyset$ and $\mathsf{memory} := \emptyset$. For each input or message, the functionality verifies that $sid = (S, sid')$ for some sid', and ignores it otherwise.

Key generation:

- Upon receiving ($\textsc{KeyGen}, S, sid$) from party S, forward ($\textsc{LeakKeyGen}, S, sid$) to the adversary $\mathcal{S}$. Ignore future ($\textsc{KeyGen}, S, \ldots$).
- Upon receiving ($\textsc{InflKeyGen}, S, sid, \langle \mathsf{s}, \mathsf{v} \rangle$) from $\mathcal{S}$, record $\langle \mathsf{s}, \mathsf{v} \rangle$ in $\mathsf{history}_S$ and memory, and output ($\textsc{KeyGenReturn}, S, sid, \mathsf{v}$) to party S, where s is a probabilistic poly-time stateless algorithm, and v is a deterministic algorithm. Ignore future ($\textsc{InflKeyGen}, S, \ldots$).

Signature generation:

- Upon receiving ($\textsc{Sign}, U, sid, \langle m, \mathsf{v}' \rangle$) from party $U \in \mathcal{U}$, record $\langle m, \mathsf{v}' \rangle$ in $\mathsf{history}_U$, and send ($\textsc{LeakSign}, U, sid, \mathsf{v}'$) to the adversary $\mathcal{S}$. Ignore any future ($\textsc{Sign}, U, \ldots$).
- Upon receiving ($\textsc{InflSign}, S, sid, \langle U, \beta \rangle$) from $\mathcal{S}$, if $\mathsf{history}_U = \emptyset$ or $\langle m, \mathsf{v}', \alpha \rangle$ in $\mathsf{history}_U$, ignore the message. Else output ($\textsc{SignReturn}, S, sid, \langle U, \beta \rangle$) to party S, and record $\langle U, \beta \rangle$ in $\mathsf{history}_S$. Ignore future ($\textsc{InflSign}, S, sid, \langle U, \ldots \rangle$).
- Upon receiving ($\textsc{InflSign}, U, sid, \alpha$) from $\mathcal{S}$, if $\mathsf{history}_U = \emptyset$ or no $\langle U, \beta \rangle$ in $\mathsf{history}_S$, then ignore the message. Else update $\mathsf{history}_U$ into $\langle m, \mathsf{v}', \alpha \rangle$. If $\alpha = 0$, return ($\textsc{SignReturn}, U, sid, \langle 0, 0 \rangle$) to party U. If $\alpha = 1$, then compute $(\sigma, \zeta) \leftarrow \mathsf{s}(m)$ where s is from $\mathsf{history}_S$ and ζ is the randomness, and update $\mathsf{history}_U$ into $\langle m, \mathsf{v}', 1, \sigma, \zeta \rangle$, and return ($\textsc{SignReturn}, U, sid, \langle 1, \sigma \rangle$) to party U. Ignore future ($\textsc{InflSign}, U, \ldots$).

Figure 4.8: Blind signature functionality $\mathcal{F}_{\mathrm{BS}}$ (part 1).

Functionality $\mathcal{F}_{\mathrm{BS}}$

Signature verification:

- Upon receiving (Verify, V, sid, $\langle m, \sigma, \mathsf{v}' \rangle$) from party $V \in \mathcal{V}$, compute $\phi \leftarrow \mathsf{v}'(m, \sigma)$, and record $\langle m, \sigma, \mathsf{v}', \phi \rangle$ in $\mathsf{history}_V$. output (VerifyReturn, V, sid, ϕ) to party V. Ignore future (Verify, V, ...).

Corruption:

- Upon receiving (Corrupt, P, sid) from the adversary $\mathcal{S}$, mark P as corrupted, and return (CorruptReturn, P, sid, $\mathsf{history}_P$) to $\mathcal{S}$.

  When party U is corrupted at some point, upon receiving (Patch, U, sid, $\langle \tilde{m}, \tilde{\mathsf{v}} \rangle$) from $\mathcal{S}$, if neither $\langle m, \mathsf{v}', \alpha \rangle$ in $\mathsf{history}_U$ nor $\langle U, \beta \rangle$ in $\mathsf{history}_S$ then set $\mathsf{history}_U \leftarrow \langle \tilde{m}, \tilde{\mathsf{v}} \rangle$ otherwise ignore the message; ignore any future (Patch, U, ...) message.

  When party S is corrupted at some point, upon receiving (Patch, S, sid, $\langle \tilde{\mathsf{s}}, U \rangle$) from $\mathcal{S}$, then update $\mathsf{s} \leftarrow \tilde{\mathsf{s}}$ in $\mathsf{history}_S$, and record $\langle \tilde{\mathsf{s}}, \mathsf{v}' \rangle$ into memory where v' is from $\mathsf{history}_U$.

Halting conditions: At any moment, halts if any condition below holds:

(i) If party S is not corrupted, $\mathsf{v}' = \mathsf{v}$, and $\mathsf{v}(m, \sigma) = 1$, where v is from $\mathsf{history}_S$ and $\mathsf{history}_V = \langle m, \mathsf{v}', \sigma, 1 \rangle$ for some V, but there is no U such that m is recorded with any σ' in $\mathsf{history}_U$;

(ii) If party S is not corrupted, and $\mathsf{v}(m, \sigma) \neq 1$, where v is in $\mathsf{history}_S$, and m, σ in $\mathsf{history}_U$ for some U;

(iii) If $\mathsf{v}'(m, \sigma) \neq 1$, where $\mathsf{history}_U = \langle m, \mathsf{v}', 1, \sigma, \zeta \rangle$ for some U;

(iv) If $\beta \neq \alpha$, where α is in $\mathsf{history}_U$ for some U, and $\langle U, \beta \rangle$ in $\mathsf{history}_S$;

(v) If there are pairs $\langle \mathsf{s}_1, \mathsf{v}_1 \rangle$ and $\langle \mathsf{s}_2, \mathsf{v}_1 \rangle$ in memory such that $\mathsf{s}_1 \neq \mathsf{s}_2$.

Figure 4.9: Blind signature functionality $\mathcal{F}_{\mathrm{BS}}$ (part 2).

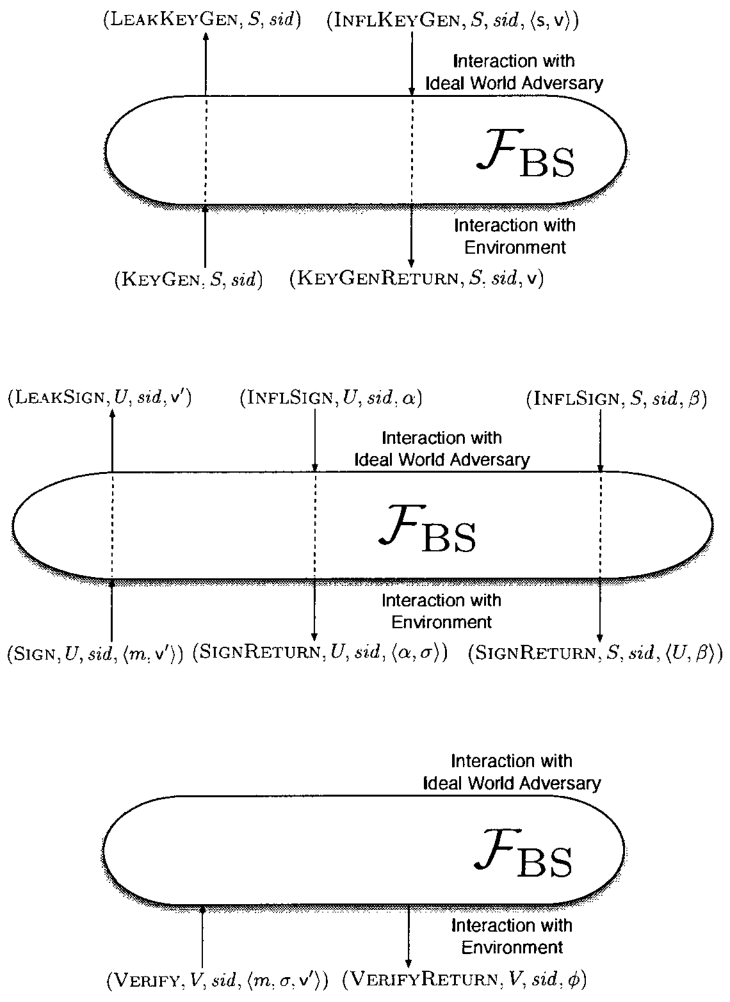

(LeakKeyGen, S, sid)
(InflKeyGen, S, sid, ⟨s, v⟩)
Interaction with Ideal World Adversary
$\mathcal{F}_{\mathrm{BS}}$
Interaction with Environment
(KeyGen, S, sid)
(KeyGenReturn, S, sid, v)
(LeakSign, U, sid, v′)
(InflSign, U, sid, α)
(InflSign, S, sid, β)
Interaction with Ideal World Adversary
$\mathcal{F}_{\mathrm{BS}}$
Interaction with Environment
(Sign, U, sid, ⟨m, v′⟩)
(SignReturn, U, sid, ⟨α, σ⟩)
(SignReturn, S, sid, ⟨U, β⟩)
Interaction with Ideal World Adversary
$\mathcal{F}_{\mathrm{BS}}$
Interaction with Environment
(Verify, V, sid, ⟨m, σ, v′⟩)
(VerifyReturn, V, sid, ϕ)

Figure 4.10: Pictorial presentation of the ideal functionality for actions KeyGen, Sign, and Verify, respectively.

world, where $\mathcal{F}_{\text{CRS}}$ is an implementation of the CRS algorithm as given in the specification of the scheme.

We note that each party that acts as a user in our framework is programmed to ask for a single signature; we make this choice without loss of generality and for the sake of keeping the description of the scheme simple. We prove that if a blind signature protocol securely realizes $\mathcal{F}_{\text{BS}}$, then the scheme is also secure in the stand alone model

The theorem below is a sanity-check that shows that the ideal functionality we propose is consistent with some of the previous modeling attempts. Naturally the intention is that realizing the ideal functionality goes much beyond the satisfaction of such previous game-based definitions. Still establishing results as the one below is important and non-trivial due to the fact that ideal functionalities interact with the ideal-world adversary substantially something that may (if they are badly designed) lead to disparities with game-based definitions.

Theorem 4.5.2. *If $\pi_{\Sigma(\text{BS})}$ can realize our $\mathcal{F}_{\text{BS}}$ in Figure 4.9, then $\Sigma(\text{BS})$ is a secure blind signature.*

Proof. The general plan of the proof is as follows: we first assume $\Sigma(\text{BS})$ is not secure according to one of the previous definitions. Then we construct an environment $\mathcal{Z}$ so that for all $\mathcal{S}$ it can distinguish the interaction with $\pi_{\Sigma(\text{BS})}$, from the interaction with the ideal world adversary $\mathcal{S}$ and $\mathcal{F}_{\text{BS}}$.

(I) We first assume that there is a successful forger F for $\Sigma(\text{BS})$. Then $\mathcal{Z}$ internally runs an instance of F. The environment $\mathcal{Z}$ invokes party S with (KeyGen, S, sid), and gives the returned verification algorithm v to F.

Blind Signature Protocol $\pi_{\Sigma(\mathsf{BS})}$

CRS generation: $crs \leftarrow \mathsf{CRS}(1^\lambda)$ where λ is the security parameter. Each party once invoked obtains the string crs immediately.

Key generation: When party S is invoked with input $(\textsc{KeyGen}, S, sid)$ from $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \mathsf{KG}(crs)$, lets the verification algorithm $\mathsf{v} := \mathsf{Vrf}(crs, vk, \cdot, \cdot)$, records $\langle sk, vk \rangle$ in $\mathsf{history}_S$, and outputs $(\textsc{KeyGenReturn}, S, sid, \mathsf{v})$ to $\mathcal{Z}$.

Signature generation: When party U is invoked with input $(\textsc{Sign}, U, sid, \langle m, \mathsf{v}' \rangle)$ from $\mathcal{Z}$ where $sid = (S, sid')$, it computes msg_1, and records the internal state in $\mathsf{history}_U$ including the randomness used, and the input m, v', and sends outgoing $(\textsc{bsig}_1, sid, msg_1)$ to party S (through $\mathcal{A}$). If party U fails to compute msg_1, then outputs $(\textsc{SignReturn}, U, sid, \langle 0, 0 \rangle)$ to $\mathcal{Z}$, and halts.

When party U is invoked with incoming $(\textsc{bsig}_i, sid, msg_i)$ from party S, where $sid = (S, sid')$, it computes msg_{i+1}, updates its internal state in $\mathsf{history}_U$, and sends outgoing $(\textsc{bsig}_{i+1}, sid, msg_{i+1})$ to party S; if msg_{i+1} is the last message generated by U, then U further computes the signature σ, updates its internal state in $\mathsf{history}_U$, and outputs $(\textsc{SignReturn}, U, sid, \langle 1, \sigma \rangle)$ to $\mathcal{Z}$ and halts. If U fails to compute msg_{i+1} or the signature σ, then outputs $(\textsc{SignReturn}, U, sid, \langle 0, 0 \rangle)$ to $\mathcal{Z}$, and halts.

When party S is invoked with incoming $(\textsc{bsig}_j, sid, msg_j)$ from party U, where $sid = (S, sid')$, it computes msg_{j+1}, records its internal state in $\mathsf{history}_S$, and sends outgoing $(\textsc{bsig}_{j+1}, sid, msg_{j+1})$ to party U; if msg_{j+1} is the last message generated by S, then S also returns $(\textsc{SignReturn}, S, sid, \langle U, 1 \rangle)$ to $\mathcal{Z}$. If S fails to compute msg_{j+1}, then outputs $(\textsc{SignReturn}, S, sid, \langle U, 0 \rangle)$ to $\mathcal{Z}$, and halts.

Signature verification: When party V is invoked with input $(\textsc{Verify}, V, sid, \langle m, \sigma, \mathsf{v}' \rangle)$ from $\mathcal{Z}$ where $sid = (S, sid')$, it outputs $(\textsc{VerifyReturn}, V, sid, \mathsf{v}'(m, \sigma))$ to $\mathcal{Z}$.

Corruption: When party J is invoked with $(\textsc{Corrupt}, J, sid)$ by $\mathcal{Z}$, it sends $(\textsc{CorruptReturn}, J, sid, \mathsf{history}_J)$ to $\mathcal{Z}$.

Figure 4.11: Blind signature protocol $\pi_{\Sigma(\mathsf{BS})}$.

When the simulated F outputs BSIG_1 message on behalf of some party U, $\mathcal{Z}$ creates party U; $\mathcal{Z}$ then corrupts party U and forces it to send his first outgoing message (through $\mathcal{A}$) to party S; when F outputs its i-th message where $i > 1$, $\mathcal{Z}$ forces party U to send the corresponding message to party S; when the corrupted party U receives an incoming message from the signer, $\mathcal{Z}$ forwards it to the simulated F. $\mathcal{Z}$ uses C_1 to count the number of successful final signer messages from the party S. $\mathcal{Z}$ uses C_2 to count the number of $(\text{SignReturn}, S, sid, \langle U, 1\rangle)$ messages obtained from party S (as this party returns this value to the subroutine output tape of $\mathcal{Z}$). When the simulated F outputs a number of, say L, forged message-signature pairs, $\mathcal{Z}$ activates L verifiers with $(\text{Verify}, V, sid, \langle \widehat{m}_i, \widehat{\sigma}_i, \mathsf{v}\rangle)$, where $i = 1, \ldots, L$; the verifiers will return 1, i.e. $(\text{VerifyReturn}, V, sid, 1)$, for the successful forged pairs. $\mathcal{Z}$ uses C_3 to count the number of 1's. If $\mathcal{Z}$ finds that $C_3 \leq C_2$ it returns 0 otherwise it returns 1.

In the real world, because F is a successful forger, with non-negligible probability, $\mathcal{Z}$ will observe $C_3 > C_1$. Moreover, note that $C_1 = C_2$ will hold in the real world, which is based on the fact: when party S sends his last outgoing message to party U, he also sends output $(\text{SignReturn}, S, sid, \langle U, 1\rangle)$ to $\mathcal{Z}$. It follows that when $\mathcal{Z}$ operates in the real world it returns 1 with non-negligible probability.

Next we turn to the ideal world. In the verification stage, when a verifier receives $(\text{Verify}, V, sid, \langle \widehat{m}_i, \widehat{\sigma}_i, \mathsf{v}\rangle)$, he will forward such message to $\mathcal{F}_{\text{BS}}$, and $\mathcal{F}_{\text{BS}}$ will check if the message is a forgery using the information he possesses. Based on the definition of $\mathcal{F}_{\text{BS}}$, such check will return $\mathsf{v}(\widehat{m}_i, \widehat{\sigma}_i)$ to the verifier, as long $\widehat{m}_i$ is recorded with a signature. In case the message is not recorded with a signature, $\mathcal{F}_{\text{BS}}$ would either return $(\text{VerifyReturn}, V,$

$sid, 0)$ or halt. It follows that C_1 will be incremented only due to messages recorded with done. Given that all users are corrupted on start there is only one possibility that a message is recorded with done within $\mathcal{F}_{\text{BS}}$: $\mathcal{S}$ has patched the particular message into some corrupted user instance inside $\mathcal{F}_{\text{BS}}$. Note that a message is recorded with done by $\mathcal{F}_{\text{BS}}$ only after a message $(\textsc{InflSign}, S, sid, \langle U, 1\rangle)$ is received from $\mathcal{S}$. The environment tracks the $(\textsc{SignReturn}, S, sid, \langle U, 1\rangle)$ messages into C_2 thus we can be certain that in the ideal world it would be impossible to have $C_3 > C_2$. It follows that in the ideal world, $C_3 \leq C_2$; thus it follows that the environment always returns 0 and it is a distinguisher between the real and the ideal world for any implementation of $\mathcal{S}$.

(II) Next assume there is a successful blindness distinguisher D for $\Sigma(\text{BS})$. Then $\mathcal{Z}$ internally runs an instance of D. Such D can be viewed as a signer inside $\mathcal{Z}$. When D outputs the verification algorithm v and two messages $\langle m_0, m_1\rangle$, $\mathcal{Z}$ activates two parties U_L and U_R with $(\textsc{Sign}, U, sid, \langle m_b, \mathsf{v}\rangle)$ and $(\textsc{Sign}, U, sid, \langle m_{1-b}, \mathsf{v}\rangle)$, respectively, where $b \xleftarrow{\$} \{0,1\}$ is randomly chosen by $\mathcal{Z}$.

Subsequently $\mathcal{Z}$ relays all messages communicated between the party D and the two user protocols. Next if $\mathcal{Z}$ obtains two valid responses $(\textsc{SignReturn}, U_L, sid, \langle \alpha_b, \sigma_b\rangle)$ and $(\textsc{SignReturn}, U_R, sid, \langle \alpha_{1-b}, \sigma_{1-b}\rangle)$ from the two parties U_L and U_R respectively. If $\alpha_0 = \alpha_1 = 1$, then D is supplied with $\langle 1, \sigma_0; 1, \sigma_1\rangle$, else with $\langle 0, 0; 0, 0\rangle$. Finally D returns b^* as the guess of the random coin b which is chosen by $\mathcal{Z}$. Here $\mathcal{Z}$ returns $b^* \stackrel{?}{=} b$.

Consider now that D is a successful blindness distinguisher for $\Sigma(\text{BS})$; in the real world, D will guess the coin b with non-negligible advantage. However, in the ideal world, no matter how the simulator $\mathcal{S}$ is implemented we observe that the bit b remains secured in

$\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\text{BS}}$ does not communicate any information related to b to $\mathcal{S}$. Therefore, $\mathcal{S}$ cannot help D to figure out b.

Due to an attack in [HK07], still D may use two different signing algorithms s_L and s_R for the interactions with the users U_L and U_R respectively; note that both s_L and s_R correspond to the same verification algorithm v; once D receives $\langle \sigma_0, \sigma_1 \rangle$ from $\mathcal{Z}$, he may use s_L and s_R to relate the message-signature pairs to the users correctly and figure out b. However, once D uses two different signing algorithms, both of them will be patched into $\mathcal{F}_{\text{BS}}$; now both $\langle U_L, \mathsf{s}_L, \mathsf{v} \rangle$ and $\langle U_R, \mathsf{s}_R, \mathsf{v} \rangle$ will be recorded which will halt the execution, and $\mathcal{Z}$ can easily distinguish the two worlds. This follows that the coins b are independent from D, and even an unbounded D cannot guess such b with probability better than 1/2. It follows that $\mathcal{Z}$ is a distinguisher between the real and the ideal worlds.

$\square$

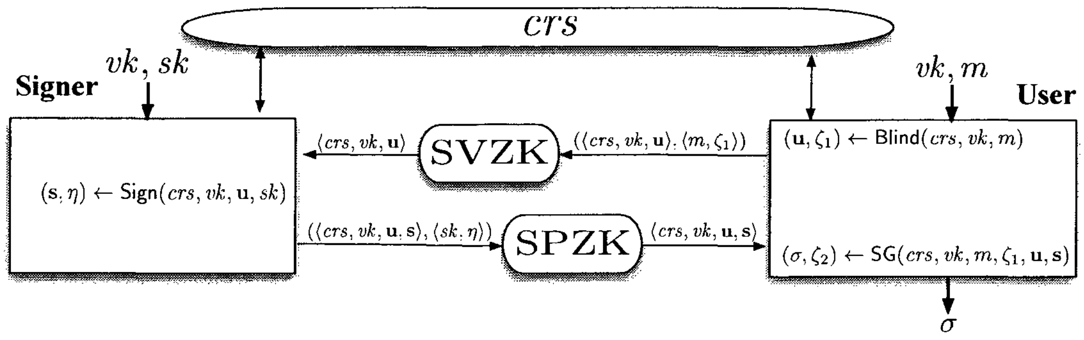
4.6 *Generic Construction for Adaptively Secure Blind Signatures*

Our design is modular and delineates the components required for designing UC blind signatures in the adaptive security setting. We present our methodology in two steps. First, we employ a lite blind signature scheme and we operate in a hybrid world where the following ideal functionalities exist: $\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}$. Here $\mathcal{F}_{\text{CRS}}$ will be an appropriate common reference string functionality; on the other hand, $\mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}$ will be two *different* zero-knowledge functionalities that are variations of the standard multi-session ZK functionality. This reflects the fact that the ZK "needs" of the user and the signer are different in a blind signature. (1) $\mathcal{F}_{\text{SVZK}}^{R_U}$ is the "single verifier zero-knowledge functionality for the relation R_U" where the user will be the prover (please refer to Figure 4.4) and, (2) $\mathcal{F}_{\text{SPZK}}^{R_S}$ is the "single-prover zero-knowledge functionality for the relation R_S" where the signer will be the prover (refer to Figure 4.7). The two functionalities differ from the multi-session zero-knowledge ideal functionality $\mathcal{F}_{\text{MZK}}$ (e.g., see $\hat{\mathcal{F}}_{\text{ZK}}$ in figure 7, page 49, in [CLOS02]) in the following manner: $\mathcal{F}_{\text{SVZK}}$ assumes that there is only a single verifier that potentially many provers wish to prove to it a certain type of statements; on the other hand, $\mathcal{F}_{\text{SPZK}}$ assumes that only a single prover exists that potentially wishes to convince many verifiers regarding a certain type of statement. Our setting is different from previous UC-formulations of ZK where multiple provers wish to convince multiple verifiers at the same time; while we could use such stronger primitives in our design, recall that we are interested in the simplest possible primitives that can instantiate our methodology as these highlight minimum sufficient requirements for blind signature design in the UC setting.

4.6.1 *Construction in the ($\mathcal{F}_{\text{CRS}}$,$\mathcal{F}_{\text{SVZK}}$,$\mathcal{F}_{\text{SPZK}}$)-Hybrid World*

In this section we describe our blind signature construction in the hybrid world. In Figure 4.13, we describe a UC blind signature protocol in the ($\mathcal{F}_{\text{CRS}}$,$\mathcal{F}_{\text{SVZK}}^{R_U}$,$\mathcal{F}_{\text{SPZK}}^{R_S}$)-hybrid world that is based on a lite blind signature protocol. The signature generation part can be pictorially described in Figure 4.12.

The relations parameterized with the ZK functionalities are $R_U = \{((crs, vk, \mathbf{u}), (m, \zeta_1)) \mid \mathbf{u} = \text{2MBS.Blind}(crs, vk, m; \zeta_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \eta)) \mid \mathbf{s} = \text{2MBS.Sign}(crs,\ vk, \mathbf{u}, sk; \eta) \wedge vk = \text{KG}(crs, sk)\}$. We remark that in the protocol in Figure 4.13 all communication between users and the signer is relayed through the ideal functionalities. We prove the following



crs
Signer
vk, sk
$(\mathsf{s}, \eta) \leftarrow \mathsf{Sign}(crs, vk, \mathbf{u}, sk)$
$\langle crs, vk, \mathbf{u} \rangle$
SVZK
$(\langle crs, vk, \mathbf{u} \rangle, \langle m, \zeta_1 \rangle)$
$(\langle crs, vk, \mathbf{u}, \mathsf{s} \rangle, \langle sk, \eta \rangle)$
SPZK
$\langle crs, vk, \mathbf{u}, \mathsf{s} \rangle$
vk, m
User
$(\mathbf{u}, \zeta_1) \leftarrow \mathsf{Blind}(crs, vk, m)$
$(\sigma, \zeta_2) \leftarrow \mathsf{SG}(crs, vk, m, \zeta_1, \mathbf{u}, \mathsf{s})$
σ

Figure 4.12: A pictorial description of signature generation.

main theorem:

Theorem 4.6.1. *Given a signature generation protocol that is a lite blind signature, the protocol $\pi_{\Sigma(\mathsf{BS})}$ in Figure 4.13 securely realizes $\mathcal{F}_{\mathrm{BS}}$ in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S})$-hybrid model.*

4.6.2 The Proof of Main Theorem

Before the proof of Theorem 4.6.1, we introduce a useful lemma as below which will help us to organize the proof. Note that we can extend such lemma to general setting.

Lemma 4.6.2. *Assume that*

(1) $\exists \mathcal{S}_1, \forall \mathcal{Z}$, $\mathrm{EXEC}_{\pi_{\Sigma(\mathsf{BS})}, \mathcal{Z}}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S}} = \mathrm{EXEC}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}^{\mathcal{F}_1}$, where $\mathcal{F}_1$ is dummy blind signature functionality (cf. Figure 4.14);

(2) for $\mathcal{S}_i$, $\exists \mathcal{S}_{i+1}$, $\forall \mathcal{Z}$, $\left|\mathrm{EXEC}_{\pi_d, \mathcal{S}_i, \mathcal{Z}}^{\mathcal{F}_i} - \mathrm{EXEC}_{\pi_d, \mathcal{S}_{i+1}, \mathcal{Z}}^{\mathcal{F}_{i+1}}\right| \le \epsilon_i$, where $1 \le i \le 4$;

(3) $\mathcal{F}_5 = \mathcal{F}_{\mathrm{BS}}$.

Protocol $\pi_{\Sigma(\mathsf{BS})}$ in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S})$-Hybrid Model

CRS generation: $crs \leftarrow \mathsf{CRS}(1^\lambda)$ where λ is the security parameter. Each party once activated obtains the string crs.

Key generation: When party S is invoked with input $(\textsc{KeyGen}, S, sid)$ by $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \mathsf{KG}(crs)$, stores sk, defines the verification algorithm $\mathsf{v} := \mathsf{Vrf}(crs, vk, \cdot, \cdot)$, and sends output $(\textsc{KeyGenReturn}, S, sid, \mathsf{v})$ to $\mathcal{Z}$.

Signature generation: Please also refer to Figure 4.12 for a pictorial description of this stage.

On input $(\textsc{Sign}, S, sid, \langle m, \mathsf{v}' \rangle)$ from $\mathcal{Z}$ where $sid = (S, sid')$, the party U obtains vk' from v', selects random ζ_1 and computes $(\mathbf{u}, \zeta_1) \leftarrow \mathsf{2MBS.Blind}(crs, vk', m)$ and sends $(\textsc{svzkProve}, U, sid, \langle (crs, vk', \mathbf{u}), (m, \zeta_1) \rangle)$ to $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$.

Upon receiving $(\textsc{svzkProveReturn}, S, sid, \langle U, (crs', vk', \mathbf{u}) \rangle)$ from $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, the party S verifies $crs' = crs$ and $vk' = vk$. If not, then the party S outputs $(\textsc{SignReturn}, S, sid, \langle U, 0 \rangle)$ to $\mathcal{Z}$. Else the party S selects random η and computes $(\mathbf{s}, \eta) \leftarrow \mathsf{2MBS.Sign}(crs, vk, \mathbf{u}, sk)$ and sends $(\textsc{spzkProve}, S, sid, \langle U, (crs, vk, \mathbf{u}, \mathbf{s}), (sk, \eta) \rangle)$ to $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$, and outputs $(\textsc{SignReturn}, S, sid, \langle U, 1 \rangle)$ to $\mathcal{Z}$.

Upon receiving $(\textsc{svzkProveReturn}, S, sid, \langle U, 0 \rangle)$ from $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, the party S outputs $(\textsc{SignReturn}, S, sid, \langle U, 0 \rangle)$ to $\mathcal{Z}$.

Upon receiving $(\textsc{spzkProveReturn}, U, sid, (crs', vk'', \mathbf{u}', \mathbf{s}))$ from $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$, the party U verifies that $crs' = crs$ and $vk'' = vk'$ and $\mathbf{u}' = \mathbf{u}$. If not, then party U outputs $(\textsc{SignReturn}, U, sid, \langle 0, 0 \rangle)$ to $\mathcal{Z}$. Else, the party U selects random ζ_2 and computes $(\sigma, \zeta_2) \leftarrow \mathsf{2MBS.SG}(crs, vk', m, \zeta_1, \mathbf{u}, \mathbf{s})$, and outputs $(\textsc{SignReturn}, U, sid, \langle 1, \sigma \rangle)$ to $\mathcal{Z}$.

Upon receiving $(\textsc{spzkProveReturn}, U, sid, 0)$ from $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$, party U outputs $(\textsc{SignReturn}, U, sid, \langle 0, 0 \rangle)$ to $\mathcal{Z}$.

Signature verification: When party V is invoked with input $(\textsc{Verify}, V, sid, \langle m, \sigma, \mathsf{v}' \rangle)$ by $\mathcal{Z}$ where $sid = (S, sid')$, it outputs $(\textsc{VerifyReturn}, V, sid, \mathsf{v}'(m, \sigma))$ to $\mathcal{Z}$.

Corruption: When party J is invoked with $(\textsc{Corrupt}, J, sid)$ by $\mathcal{Z}/\mathcal{A}$, it sends $(\textsc{CorruptReturn}, J, sid, \mathsf{history}_J)$ to $\mathcal{Z}/\mathcal{A}$.

Figure 4.13: Blind signature protocol $\pi_{\Sigma(\mathsf{BS})}$ in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S})$-hybrid model based on a lite-blind signature scheme $\mathsf{2MBS}.\{\mathsf{CRS}, \mathsf{KG}, \mathsf{Blind}, \mathsf{Sign}, \mathsf{SG}, \mathsf{Vrf}\}$.

Then we have $\exists\mathcal{S}_5, \forall\mathcal{Z}, \left|\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BS}}}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}\right| \leq \sum_{i=1}^{4}\epsilon_i$.

Proof. The proof is straightforward. Recall that $\exists\mathcal{S}_1, \forall\mathcal{Z}, \mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}}$. For such $\mathcal{S}_1$, we have $\exists\mathcal{S}_2, \forall\mathcal{Z}, \left|\mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}\right| \leq \epsilon_1$. Therefore, we have the following: $\exists\mathcal{S}_2, \forall\mathcal{Z}, \left|\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}\right| \leq \epsilon_1$. Similarly, we have $\exists\mathcal{S}_5, \forall\mathcal{Z}$, $\left|\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}\right| \leq \epsilon_1 + \cdots + \epsilon_4$. Note that $\mathcal{F}_5 = \mathcal{F}_{\mathrm{BS}}$. We complete the proof. □

Now we turn to the proof of Theorem 4.6.1.

Proof. In order to prove that $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BS}}}_{\pi_d,\mathcal{S},\mathcal{Z}}$, we use the proof strategy explored in Lemma 4.6.2. We develop several bridge hybrid worlds between the $(\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}})$-hybrid world and the ideal world, and define the ensemble of random variables of $\mathcal{Z}$'s output of each bridge hybrid worlds as $\mathrm{EXEC}^{\mathcal{F}_i}_{\pi_d,\mathcal{S}_i,\mathcal{Z}}$, $i = 1, 2, \ldots, 5$, where π_d is the dummy protocol same as that in the ideal world. Next we prove $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}^{R_U}_{\mathrm{SVZK}},\mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})},\mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} \approx \cdots \approx \mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BS}}}_{\pi_d,\mathcal{S},\mathcal{Z}}$. In our sequence of games we introduce a sequence of functionalities called "vault" which gradually becomes from dummy blind signature functionality $\mathcal{F}_1$ into the ideal functionality $\mathcal{F}_{\mathrm{BS}} = \mathcal{F}_5$ across a sequence of five steps. At same time the corresponding simulator becomes from $\mathcal{S}_1$ into ideal world simulator $\mathcal{S} = \mathcal{S}_5$. We also explicitly present the construction of $\mathcal{S}$ after the proof.

Note that we assume the underlying lite blind signature satisfies both lite-unforgeability and lite-equivocality.

$\mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}}$. Here the vault $\mathcal{F}_1$, i.e. the dummy blind signature functionality, is between the dummy parties S, U, V and the simulator $\mathcal{S}_1$; the vault $\mathcal{F}_1$ just forwards the mes-

Dummy Blind Signature Functionality $\mathcal{F}_1$

Initialization is same as in $\mathcal{F}_{\text{BS}}$.

Key generation:

- Upon receiving ($\textsc{KeyGen}, S, sid$) from party S, forward ($\textsc{LeakKeyGen}, S, sid$) to the adversary $\mathcal{S}$. Ignore future ($\textsc{KeyGen}, S, \ldots$).
- Upon receiving ($\textsc{InflKeyGen}, S, sid, \mathsf{v}$) from $\mathcal{S}$, record v in $\mathsf{history}_S$ and memory, and output ($\textsc{KeyGenReturn}, S, sid, \mathsf{v}$) to party S, where v is a deterministic algorithm. Ignore future ($\textsc{InflKeyGen}, S, \ldots$).

Signature generation:

- Upon receiving ($\textsc{Sign}, U, sid, \langle m, \mathsf{v}' \rangle$) from party $U \in \mathcal{U}$, record $\langle m, \mathsf{v}' \rangle$ in $\mathsf{history}_U$, and send ($\textsc{LeakSign}, U, sid, \langle m, \mathsf{v}' \rangle$) to the adversary $\mathcal{S}$. Ignore any future ($\textsc{Sign}, U, \ldots$).
- Upon receiving ($\textsc{InflSign}, S, sid, \langle U, \beta \rangle$) from $\mathcal{S}$, output ($\textsc{SignReturn}, S, sid, \langle U, \beta \rangle$) to party S, and record $\langle U, \beta \rangle$ in $\mathsf{history}_S$. Ignore future ($\textsc{InflSign}, S, sid, \langle U, \ldots \rangle$).
- Upon receiving ($\textsc{InflSign}, U, sid, \langle \alpha, \sigma \rangle$) from $\mathcal{S}$, return ($\textsc{SignReturn}, U, sid, \langle \alpha, \sigma \rangle$) to party U, and update $\mathsf{history}_U$ into $\langle m, \mathsf{v}', \alpha, \sigma \rangle$. Ignore future ($\textsc{InflSign}, U, \ldots$).

Signature verification:

- Upon receiving ($\textsc{Verify}, V, sid, \langle m, \sigma, \mathsf{v}' \rangle$) from party $V \in \mathcal{V}$, record $\langle m, \sigma, \mathsf{v}' \rangle$ in $\mathsf{history}_V$, and send ($\textsc{LeakVerify}, V, sid, \langle m, \sigma, \mathsf{v}' \rangle$) to the adversary $\mathcal{S}$. Ignore future ($\textsc{Verify}, V, \ldots$).
- Upon receiving ($\textsc{InflVerify}, V, sid, \phi$) from $\mathcal{S}$, update $\mathsf{history}_V = \langle m, \sigma, \mathsf{v}', \phi \rangle$, and output ($\textsc{VerifyReturn}, V, sid, \phi$) to party V.

Corruption:

- Upon receiving ($\textsc{Corrupt}, P, sid$) from the adversary $\mathcal{S}$, mark P as corrupted.

Figure 4.14: Dummy blind signature functionality $\mathcal{F}_1$ in Lemma 4.6.2.

sages between the dummy parties and $\mathcal{S}_1$, and at same time does some "basic" recording. Please refer to Figure 4.14.

The simulator $\mathcal{S}_1$ simulates exactly the protocol $\pi_{\Sigma(\text{BS})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model except that all inputs/outputs of the parties of protocol $\pi_{\Sigma(\text{BS})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model are from/to the vault $\mathcal{F}_1$ instead of from/to $\mathcal{Z}$.

Analysis:

Note that $\mathcal{S}_1$ restates the whole execution in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world. So we have $\text{EXEC}_{\pi_{\Sigma(\text{BS})},\mathcal{Z}}^{\mathcal{F}_{\text{CRS}},\mathcal{F}_{\text{SVZK}}^{R_U},\mathcal{F}_{\text{SPZK}}^{R_S}} = \text{EXEC}_{\pi_d,\mathcal{S}_1,\mathcal{Z}}^{\mathcal{F}_1}$.

$\text{EXEC}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}^{\mathcal{F}_2}$. Here the vault $\mathcal{F}_2$, operating like the vault $\mathcal{F}_1$, forwards the messages between the dummy parties and the simulator $\mathcal{S}_2$, and records some basic information. Furthermore, $\mathcal{F}_2$ needs to deal with the "extraction" of m used by a corrupted U as that in $\mathcal{F}_{\text{BS}}$: When party U is corrupted at some point, upon receiving $(\text{PATCH}, U, sid, \langle \tilde{m}, \tilde{\mathsf{v}} \rangle)$ from $\mathcal{S}$, if neither $\langle m, \mathsf{v}', \alpha \rangle$ in $\mathsf{history}_U$ nor $\langle U, \beta \rangle$ in $\mathsf{history}_S$ then set $\mathsf{history}_U \leftarrow \langle \tilde{m}, \tilde{\mathsf{v}} \rangle$ otherwise ignore the message; ignore any future $(\text{PATCH}, U, \ldots)$ message.

$\mathcal{S}_2$ is same as $\mathcal{S}_1$ to simulate the whole $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world except: in the case that the user is corrupted, $\mathcal{Z}$ instructs $\mathcal{A}$ to send $((crs, \tilde{vk}, \mathbf{u}), (\tilde{m}, \zeta_1))$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$ on the behalf of the corrupted user U; if $((crs, \tilde{vk}, \mathbf{u}), (\tilde{m}, \zeta_1)) \in R_U$, then $\mathcal{S}_2$ patches $\langle \tilde{m}, \tilde{\mathsf{v}} \rangle$ into $\mathcal{F}_2$, where algorithm $\tilde{\mathsf{v}}$ is obtained from the verification key $\tilde{vk}$.

Analysis:

This is a preparation step for the next step and the modification has no effect on $\mathcal{Z}$'s output. So $\text{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}$.

$\text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Here, we enable the functionality to be able to verify signatures as in $\mathcal{F}_{\text{BS}}$. Now the vault $\mathcal{F}_3$ is same as $\mathcal{F}_2$ except the signature verification stage is same as in $\mathcal{F}_{\text{BS}}$ and the halting conditions (i)-(iv) are included.

Analysis:

Given that the underlying 2MBS is complete, the halting (ii)-(iv) will not be violated. We still need to argue that the halting condition (i) will not be violated. First we note that once the signer S is corrupted, then halting condition (i) will not be violated. Next, we argue when the signer S is honest, the halting condition (i) will not be violated given the underlying 2MBS is lite-unforgeable.

We define **E** as the event that in the $\mathcal{F}_2$-hybrid world, the signer has generated the verification algorithm v, and some party V is activated with a verification request $(\text{VERIFY}, V, sid, \langle m, \sigma, \text{v}\rangle)$, where $\text{v}(m,\sigma) = 1$, and S is honest at this moment, and m has not been signed. Note that event **E** can also be defined in the $\mathcal{F}_3$-hybrid world. The only difference between the two worlds, i.e. the $\mathcal{F}_2$-hybrid world and the $\mathcal{F}_3$-hybrid world, is the verification stage. If event **E** does not occur, then $\text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Based on the difference lemma (refer to [Sho04]), $|\text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} - \text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}| \leq \Pr[\mathbf{E}]$. Now we still need to argue that $\Pr[\mathbf{E}]$ is negligible.

We now construct an algorithm $\mathcal{B}$ to output $(k+1)$ message-signature pairs after k queries to the signing oracle. Here the algorithm $\mathcal{B}$ is supplied with a verification key vk and is allowed to access to the signing oracle as defined in the lite-unforgeability model.

$\mathcal{B}$ runs a simulated $\mathcal{S}_2$ and a simulated $\mathcal{F}_2$. Note that now the simulator $\mathcal{S}_2$ has to simulate party S without knowing the signing key sk.

- When $\mathcal{Z}$ activates some party S with input $(\textsc{KeyGen}, S, sid)$ with $sid = (S, sid')$ for some sid', $\mathcal{B}$ returns v to $\mathcal{Z}$, where $\mathsf{v} \overset{\text{def}}{=} \mathsf{Vrf}(crs, vk, \cdot, \cdot)$. Note that vk is from the input of $\mathcal{B}$ as described before.
- When the simulated S receives $(\textsc{svzkProveReturn}, S, sid, \langle U, (crs, vk', \mathbf{u})\rangle)$ from $\mathcal{F}^{R_U}_{\text{SVZK}}$, in the case that $vk' = vk$, $\mathcal{S}_2$ needs to simulate $\mathcal{F}^{R_S}_{\text{SPZK}}$ to return $(\textsc{spzkProve}, U, sid, (crs, vk, \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$. However $\mathcal{S}_2$ cannot produce $\mathbf{s}$ by itself because now $\mathcal{S}_2$ does not have the signing key sk. Note that $\mathcal{S}_2$ simulates $\mathcal{F}^{R_U}_{\text{SVZK}}$ and $\langle m, \zeta_1\rangle$ can always be obtained from $\mathcal{Z}$; then $\mathcal{S}_2$ queries the signing oracle with $\langle m, \zeta_1\rangle$ and obtains $\mathbf{s}$.
- After finishing the above step, the simulator $\mathcal{S}_2$ lets $\mathcal{F}^{R_S}_{\text{SPZK}}$ return $(\textsc{spzkProve}, U, sid, (crs, vk, \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$. In the case that the user is honest, the user will produce signature σ for m; $\mathcal{B}$ records such (m, σ) pairs. In the case that the user is corrupted, $\mathcal{B}$ computes $(\sigma, \zeta_2) = \mathsf{2MBS.SG}(crs, vk, m, \zeta_1, \mathbf{u}, \mathbf{s})$ by randomly selecting ζ_2; $\mathcal{B}$ records (m, σ).
- When $\mathcal{Z}$ activates some party V with input $(\textsc{Verify}, V, sid, \langle m, \sigma, \mathsf{v}'\rangle)$, $\mathcal{B}$ checks whether (m, σ) is a forgery, i.e. if $\mathsf{v}' = \mathsf{v}$, $\mathsf{v}'(m, \sigma) = 1$, and m has never been queried to the signing oracle. If (m, σ) is a forgery, $\mathcal{B}$ outputs the pair and all recorded pairs, say the number is k, and halts. If $\mathcal{S}_2$ is asked by $\mathcal{Z}$ to corrupt the signer then $\mathcal{B}$ halts.

Note that whenever the event $\mathbf{E}$ occurs, algorithm $\mathcal{B}$ can produce a successful one-more forgery. Since the underlying 2MBS is lite-unforgeable, $\Pr[\mathbf{E}] \overset{c}{\approx} 0$, and we have $\mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$.

$\mathrm{EXEC}^{\mathcal{F}_4}_{\pi_d,\mathcal{S}_4,\mathcal{Z}}$. Here we let the simulator $\mathcal{S}_4$ send the vault $\mathcal{F}_4$ the pair $\langle \mathsf{s}, \mathsf{v}\rangle$ in the key generation where $\mathsf{s} \overset{\mathsf{def}}{=} \widetilde{\mathsf{SG}}(crs, \tau, vk, sk, \cdot)$. The first difference between $\mathcal{F}_4$ and $\mathcal{F}_3$ is that the key generation stage in $\mathcal{F}_4$ is same as that in $\mathcal{F}_{\mathrm{BS}}$.

The second difference is as follows. In the case that the signer is corrupted, inside $\mathcal{S}_4$, when $\mathcal{Z}$ instructs $\mathcal{A}$ to send $((crs, vk, \mathbf{u}, \mathsf{s}), (\tilde{sk}, \eta))$ to $\mathcal{F}^{R_S}_{\mathrm{SPZK}}$ on behalf of the corrupted signer S, $\mathcal{S}_4$ defines $\tilde{\mathsf{s}} \overset{\mathsf{def}}{=} \widetilde{\mathsf{SG}}(crs, \tau, vk, \tilde{sk}, \cdot)$ and patches $\tilde{\mathsf{s}}$ into $\mathcal{F}_4$. Now $\mathcal{F}_4$ deals with the patching for $\tilde{\mathsf{s}}$ as in $\mathcal{F}_{\mathrm{BS}}$.

Analysis:

This is a preparation step for the next step and the modification has no effect on $\mathcal{Z}$'s output. So $\mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_4}_{\pi_d,\mathcal{S}_4,\mathcal{Z}}$.

$\mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}$. First, $\mathcal{F}_5$ include the halting condition (v). Second, when the vault $\mathcal{F}_5$ receives $(\textsc{Sign}, U, sid, \langle m, \mathsf{v}'\rangle)$ from each dummy user U, $\mathcal{F}_5$ "blocks" m and sends $(\textsc{LeakSign}, U, sid, \mathsf{v}')$ to $\mathcal{S}_5$; now $\mathcal{S}_5$ needs to simulate the user U without the real m (as opposed to the real m used in $\mathcal{S}_4$). Note that the underlying 2MBS is lite-equivocal, so now $\mathcal{S}_5$ can take advantage of the algorithm $\widetilde{\mathsf{Blind}}$ and s (i.e., $\widetilde{\mathsf{SG}}$) to finish the simulation. First $\mathcal{S}_5$ computes $(\mathbf{u}, \xi_1) \leftarrow \widetilde{\mathsf{Blind}}(crs, vk, \tau)$, and once the user is corrupted, $\mathcal{S}_5$ obtains from $\mathcal{F}_5$ the value ζ which includes the randomness and the signed plaintext, and then $\mathcal{S}_5$

computes the consistent internal states as in the $\mathcal{F}_4$-hybrid world by running $\widehat{\mathsf{Equiv}}$ algorithm.

Analysis:

Given that the algorithm KG generates key pair (vk, sk) such that for each vk there exists only a single sk, the halting condition (v) in $\mathcal{F}_5$ will not be violated. Futhermore, notice that the $\mathcal{F}_4$-hybrid and $\mathcal{F}_5$-hybrid worlds can be viewed as the oracles $\mathcal{O}_1$ and $\mathcal{O}_0$ respectively in the lite-equivocality definition, and the underlying 2MBS is lite-equivocal. We can conclude that the two worlds are computationally indistinguishable, i.e., $\mathrm{EXEC}^{\mathcal{F}_4}_{\pi_d, \mathcal{S}_4, \mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d, \mathcal{S}_5, \mathcal{Z}}$.

Note that the vault $\mathcal{F}_5$ is exactly the functionality $\mathcal{F}_{\mathrm{BS}}$ and $\mathcal{S}_5$ is same as $\mathcal{S}$ in the ideal world. So $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{BS}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d, \mathcal{S}_5, \mathcal{Z}}$. Based on all discussions above, we have $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{BS}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}^{R_U}_{\mathrm{SVZK}}, \mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BS})}, \mathcal{Z}}$. □

4.7 Leaking ZK and its Application to Blind Signatures

4.7.1 Leaking SPZK Functionality and its Implementation

The efficiency of our UC blind signature protocol presented in the previous subsection can be significantly improved. We find the SPZK functionality $\mathcal{F}_{\mathrm{SPZK}}$ in the protocol can be replaced by a much weaker *Leaking SPZK* functionality $\mathcal{F}_{\mathrm{LSPZK}}$. The reason is that in the UC blind signature security proof, the simulator knows the signing secret which means the witness for $\mathcal{F}_{\mathrm{SPZK}}$ is known by the simulator, and thus no equivocation of dishonestly simulated transcripts is ever necessary!

We next give the description of functionality $\mathcal{F}_{\text{LSPZK}}$ in Figure 4.15.

Functionality $\mathcal{F}_{\text{LSPZK}}^{R,\mathcal{N}}$

$\mathcal{F}_{\text{LSPZK}}^{R,\mathcal{N}}$, parameterized by a binary relation R and a non-information oracle $\mathcal{N}$, interacts with adversary $\mathcal{S}$, and a single prover P and a set $\mathcal{V}$ of verifiers. The oracle $\mathcal{N}$ is instantiated as follows: upon a plaintext w, $\mathcal{N}$ returns $\langle c, \xi \rangle$ where $(c, \xi) \leftarrow \mathsf{Com}(pk, w)$, Com is a commitment scheme with key pk. For each input or message, the functionality verifies that $sid = (P, sid')$ for some sid', and ignores it otherwise.

Proof:

- Upon receiving $(\textsc{spzkProve}, P, sid, \langle V, x, w \rangle)$ from the prover P, where $V \in \mathcal{V}$, obtain $\langle c, \xi \rangle$ from $\mathcal{N}$, forward $(\textsc{spzkLeakProve}, P, sid, \langle V, x, \phi, c \rangle)$ to the adversary $\mathcal{S}$, and record $\langle V, x, w, c, \xi \rangle$ into $\mathsf{history}_P$, where $\phi = 1$ if $(x, w) \in R$, and $\phi = 0$ otherwise. Ignore future $(\textsc{spzkProve}, P, sid, \langle V, \ldots \rangle)$.
- Upon receiving $(\textsc{spzkInflProve}, V, sid)$ from the adversary $\mathcal{S}$, output $(\textsc{spzkProveReturn}, V, sid, \langle V, x, \phi \rangle)$ to party V. Ignore future $(\textsc{spzkInflProve}, V, sid)$.

Corruption:

- Upon receiving $(\textsc{Corrupt}, P, sid)$ from the adversary $\mathcal{S}$, return $\mathcal{S}$ $(\textsc{CorruptReturn}, P, sid, \mathsf{history}_P)$.

  After the successful corruption of P, if no output $(\textsc{spzkProveReturn}, V, sid, \ldots)$ was returned to party V yet, upon receiving $(\textsc{Patch}, P, sid, \langle V, x', w', c', \xi' \rangle)$ from the adversary $\mathcal{S}$, output $(\textsc{spzkProveReturn}, V, sid, \langle x', \phi' \rangle)$ to party V, where $\phi' = 1$ if $(x', w') \in R$ and $c' = \mathsf{Com}(pk, w'; \xi')$, and $\phi' = 0$ otherwise.

Figure 4.15: Leaking single-prover zero-knowledge functionality $\mathcal{F}_{\text{LSPZK}}$.

An instantiation for leaking SPZK can be found in Figure 4.17: (1) it is based on an extractable commitment EXC; (2) a Sigma protocol 3MZK = 3MZK.{Comt, Coins, Resp, Vrf,} is used to show the consistency of the relation R and the extractable commitment, i.e. for

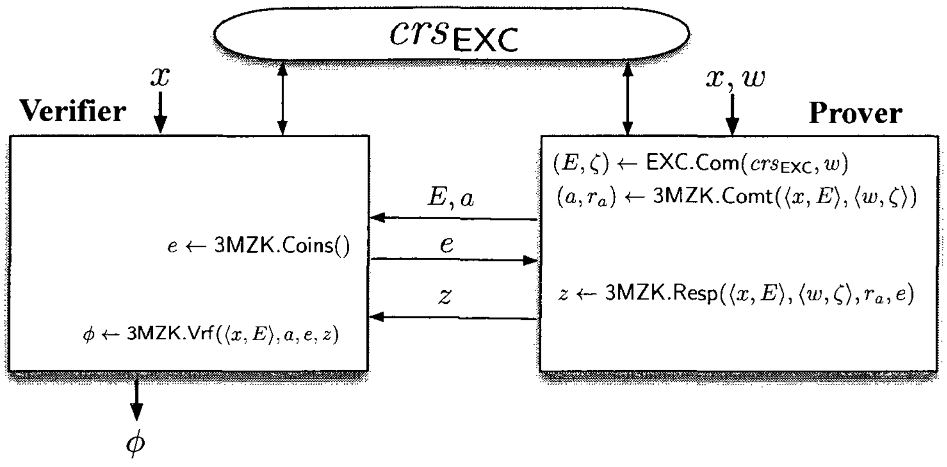

crs_{EXC}
x
Verifier
x, w
Prover
$(E,\zeta) \leftarrow \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, w)$
$(a, r_a) \leftarrow \mathsf{3MZK.Comt}(\langle x, E\rangle, \langle w, \zeta\rangle)$
E, a
$e \leftarrow \mathsf{3MZK.Coins}()$
e
z
$z \leftarrow \mathsf{3MZK.Resp}(\langle x, E\rangle, \langle w, \zeta\rangle, r_a, e)$
$\phi \leftarrow \mathsf{3MZK.Vrf}(\langle x, E\rangle, a, e, z)$
ϕ

Figure 4.16: A combination of a committing step with a Sigma protocol for relation $R' = \{(crs, x, E), (w, \zeta) \mid (x, w) \in R \wedge E = \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, w; \zeta)\}$ in the single-prover setting.

relation $R' = \{(crs, x, E), (w, \zeta) \mid (x, w) \in R \wedge E = \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, w; \zeta)\}$, and a combination of the Sigma protocol with the committing step of extractable commitment can be found in Figure 4.16; (3) also, COM is an equivocal commitment schemes for applying Damgård trick to the Sigma protocol.

Theorem 4.7.1. *Given an extractable commitment scheme* EXC, *an equivocal commitment scheme* COM, *and a non-erasure Sigma protocol* 3MZK.{Comt, Coins, Resp, Vrf}, *the Leaking SPZK protocol* $\pi_{\Sigma(\mathrm{LSPZK})}$ *in Figure 4.17 securely realizes* $\mathcal{F}_{\mathrm{LSPZK}}^{R,\mathcal{N}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid model.*

4.7.2 Alternative Implementation for Leaking SPZK

Here we present an alternative realization of $\mathcal{F}_{\mathrm{LSPZK}}$ with the potential of reducing communication rounds complexity; please refer to Figure 4.18. In Section 4.8, by using this protocol, we can save 2 moves of communication in our final concrete construction of blind

Protocol $\pi_{\Sigma(\mathrm{LSPZK})}$ in the $\mathcal{F}_{\mathrm{CRS}}$-Hybrid World

CRS generation: Set $crs \leftarrow (crs_{\mathsf{EXC}}, crs_{\mathsf{COM}})$ where $crs_{\mathsf{EXC}} \leftarrow \mathsf{EXC.CRS}(1^{\lambda})$ and $crs_{\mathsf{COM}} \leftarrow \mathsf{COM.CRS}(1^{\lambda})$.

Proof: Please refer to Figure 4.17 for a pictorial presentation. When party P is invoked with input $(\textsc{spzkProve}, P, sid, \langle V, x, w\rangle)$ from $\mathcal{Z}$, it verifies that $sid = (P, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R$, it computes $(E, \zeta) \leftarrow \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, w)$ and $(a, r_a) \leftarrow \mathsf{3MZK.Comt}((x, E), (w, \zeta))$ and $(c, r) \leftarrow \mathsf{COM.Com}(crs_{\mathsf{COM}}, a)$, and then sends message $(\textsc{spzk}_1, sid, \langle x, E, c\rangle)$ to party V (through $\mathcal{A}$).

When party V is invoked with incoming $\textsc{spzk}_1$ message, it selects $e \leftarrow \mathsf{3MZK.Coins}()$, and sends message $(\textsc{spzk}_2, sid, e)$ to party P (through $\mathcal{A}$).

When party P is invoked with incoming $\textsc{spzk}_2$ message, it computes $z \leftarrow \mathsf{3MZK.Resp}((x, E), (w, \zeta), r_a, e)$, and sends message $(\textsc{spzk}_3, sid, \langle a, r, z\rangle)$ to party V (through $\mathcal{A}$).

When party V is invoked with incoming $\textsc{spzk}_3$ message, it verifies $c = \mathsf{COM.Com}(crs_{\mathsf{COM}}, a; r)$ and $\mathsf{Vrf}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\textsc{spzkProveReturn}, V, sid, \langle V, x, 1\rangle)$ to $\mathcal{Z}$. Otherwise return $(\textsc{spzkProveReturn}, V, sid, \langle V, x, 0\rangle)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\textsc{Corrupt}, P, sid)$ by $\mathcal{Z}$, it sends outgoing $(\textsc{CorruptReturn}, P, sid, \mathsf{history}_P)$ to $\mathcal{Z}$.

Figure 4.17: Leaking single-prover zero-knowledge protocol $\pi_{\Sigma(\mathrm{LSPZK})}$ for relation R in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world.

signatures.

Protocol $\pi'_{\Sigma(\mathrm{SPZK})}$ in the $\mathcal{F}_{\mathrm{CRS}}$-Hybrid World

CRS generation: Set $crs \leftarrow (crs_{\mathsf{EXC}}, crs_{\mathsf{COM}})$ where $crs_{\mathsf{EXC}} \leftarrow \mathsf{EXC.CRS}(1^{\lambda})$ and $crs_{\mathsf{COM}} \leftarrow \mathsf{COM.CRS}(1^{\lambda})$.

Proof: When party P is invoked with input $(\textsc{spzkProve}, P, sid, \langle V, x, w\rangle)$ by $\mathcal{Z}$, it verifies that $sid = (P, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R$, it computes $(E, \zeta) \leftarrow \mathsf{EXC.Com}(crs_{\mathsf{EXC}}, w)$, $(a, r_a) \leftarrow \mathsf{3MZK.Comt}((x, E), (w, \zeta))$ and $(c, r) \leftarrow \mathsf{COM.Com}(crs_{\mathsf{COM}}, x, E, a)$, and then sends message $(\textsc{spzk}_1, sid, c)$ to party V (through $\mathcal{A}$).

When party V is invoked with incoming $\textsc{spzk}_1$ message, it selects $e \leftarrow \mathsf{3MZK.Coins}()$, and sends message $(\textsc{spzk}_2, sid, e)$ to party P.

When party P is invoked with incoming $\textsc{spzk}_2$ message, it computes $z \leftarrow \mathsf{3MZK.Resp}((x, E), (w, \zeta), r_a, e)$, and sends message $(\textsc{spzk}_3, sid, \langle x, E, a, r, z\rangle)$ to party V.

When party V is invoked with incoming $\textsc{spzk}_3$ message, it verifies $c = \mathsf{COM.Com}(crs_{\mathsf{COM}}, a; r)$ and $\mathsf{3MZK.Vrf}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\textsc{spzkProveReturn}, V, sid, \langle V, x, 1\rangle)$ to $\mathcal{Z}$. Otherwise returns message $(\textsc{spzkProveReturn}, V, sid, \langle V, x, 0\rangle)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\textsc{Corrupt}, P, sid)$ by $\mathcal{Z}$, it sends outgoing $(\textsc{CorruptReturn}, P, sid, \mathsf{history}_P)$ to $\mathcal{Z}$.

Figure 4.18: Single-prover zero-knowledge protocol $\pi'_{\Sigma(\mathrm{SPZK})}$ for relation R in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world.

Theorem 4.7.2. *Given an extractable commitment scheme* EXC, *an equivocal commitment scheme* COM, *and a non-erasure Sigma protocol* $\mathsf{3MZK} = \mathsf{3MZK.}\{\mathsf{Comt}, \mathsf{Coins}, \mathsf{Resp}, \mathsf{Vrf}\}$, *the Leaking SPZK protocol* $\pi'_{\Sigma(\mathrm{LSPZK})}$ *in Figure 4.18 securely realizes* $\mathcal{F}^{R,\mathcal{N}}_{\mathrm{LSPZK}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid* *model.*

4.8 Concrete Construction from Okamoto Signatures

In this section, we demonstrate how it is possible to derive an efficient UC blind signature instantiation based on Theorem 4.6.1 and the realization of its hybrid world with the related ZK-functionalities. Note that we opt for minimizing the overall communication complexity as opposed to round complexity. We need three ingredients: (1) an equivocal lite blind signature scheme, (2) a UC-realization of the ideal functionality $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, (3) a UC-realization of the ideal functionality $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$. Regarding (1) we will employ the lite blind signature scheme of Figure 4.3. Regarding the two ZK functionalities we will follow the design strategy outlined in Section 4.6. Recall that in Figure 4.13, $R_U = \{((crs, vk, \mathbf{u}), (m, \zeta_1)) \mid \mathbf{u} = \mathsf{2MBS.Blind}(crs, m; \zeta_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \eta)) \mid \mathbf{s} = \mathsf{2MBS.Sign}(crs, vk, \mathbf{u}, sk; \eta) \wedge vk = \mathsf{KG}(crs, sk)\}$. Instantiating these relations for the protocol of Figure 4.3 we have that $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$ and $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

4.8.1 *Efficient Instantiation for SVZK*

Based on the SVZK protocol discussed in Section 4.6 (presented in Figure 4.5), and considering the underlying equivocal lite blind signature in Figure 4.3, we can realize $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$ efficiently. We instantiate the equivocal commitment COM with a hashed Pedersen commitment [Ped91]. For the relation R_U, it is easy to derive an efficient Sigma protocol Σ^{R_U} (the reader may refer to Figure 4.19). In the resulting construction, the value W that is present within the statement proved by the user can play the role of an equivocal commitment and thus we can take advantage of this fact and save somewhat in the communication

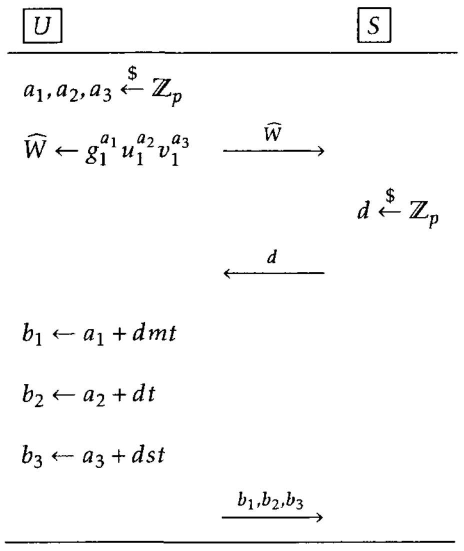

$\boxed{U}$
$\boxed{S}$
$a_1, a_2, a_3 \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$
$\xrightarrow{\widehat{W}}$
$d \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\xleftarrow{d}$
$b_1 \leftarrow a_1 + dmt$
$b_2 \leftarrow a_2 + dt$
$b_3 \leftarrow a_3 + dst$
$\xrightarrow{b_1, b_2, b_3}$

Figure 4.19: Σ^{R_U}-protocol, where $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$.

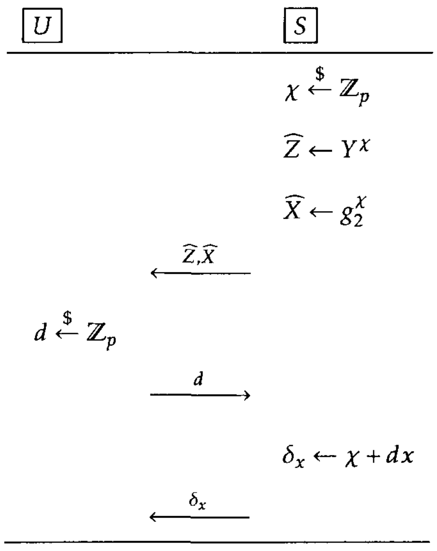

$\boxed{U}$
$\boxed{S}$
$\chi \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\widehat{Z} \leftarrow Y^{\chi}$
$\widehat{X} \leftarrow g_2^{\chi}$
$\xleftarrow{\widehat{Z}, \widehat{X}}$
$d \overset{\$}{\leftarrow} \mathbb{Z}_p$
$\xrightarrow{d}$
$\delta_x \leftarrow \chi + dx$
$\xleftarrow{\delta_x}$

Figure 4.20: Σ^{R_S}-protocol, where $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (Wv_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

complexity (this is reflected in the full protocol description that is presented in Figure 4.23 and Figure 4.24). The resulting protocol consists of 5 moves.

4.8.2 *Efficient Instantiation for Leaking SPZK*

Based on the Leaking SPZK protocol discussed before (presented in Figure 4.17) we instantiate the extractable commitment by a Paillier encryption [Pai99]; an efficient Sigma protocol Σ^{R_S} for relation R_S is easy to be constructed (we present such protocol explicitly in Figure 4.20); the required equivocal commitment that is needed to employ Damgård's trick [Dam00] is instantiated with a hashed Pedersen commitment. The resulting protocol has 3 moves.

4.8.3 Communication Rounds Optimization

Based on the protocols put forth above, we can obtain an 8-move blind signature protocol by having the user make the first proof and the signer respond (refer to Figure 4.21). Nevertheless, it is possible to achieve a 6-move protocol by carefully interleaving the communication transcripts of the two sides (refer to Figure 4.22). This is possible by modifying the UC protocol that realizes $\mathcal{F}_{\mathrm{LSPZK}}$ in the following manner: the signer (who plays the role of the prover in Leaking SPZK) starts the proof prior to receiving the final communication of the user's protocol; this naturally puts the signer at risk as Theorem 4.6.1 does not apply directly anymore (the signer starts the proof prior to the user completing her part). We mitigate the problem by modifying the signer's ZK-protocol so that it leaks no information in the first move; this enables an early start for the signer; the trick relies on a commitment that is performed by the signer in the first step. The resulting protocol is also a realization of $\mathcal{F}_{\mathrm{LSPZK}}$ (and is presented in Figure 4.18). This trick may be of general interest as it has the potential of reducing the number of rounds when two parties are bilaterally proving to each other statements in zero-knowledge.

For the final 6-move blind signature protocol, please refer to Figure 4.23 and Figure 4.24.

Finally, we can obtain the corollary below:

Corollary 4.8.1. *Under the DCR assumption, the DLOG assumption, and the 2SDH assumption, and assuming existence of collision resistant hash function, there exists a blind signature protocol that securely realizes $\mathcal{F}_{\mathrm{BS}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model.*

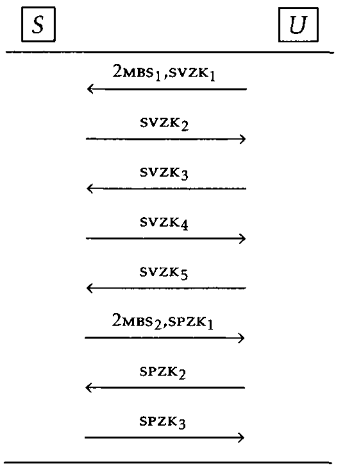


Figure 4.21: 8-move BSIG protocol based on 2MBS, SVZK, and SPZK.

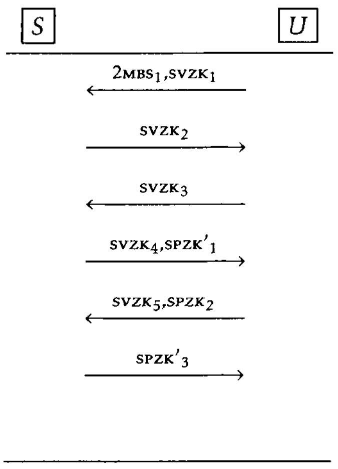


Figure 4.22: 6-move BSIG protocol based on 2MBS, SVZK, and SPZK. Here 2MBS_2 is committed inside SPZK'_1, and will be opened until SPZK'_3; except this difference, SPZK'_1 and SPZK'_3 are same as SPZK_1, SPZK_3 in Figure 4.21.

4.8.4 Description of the Concrete UC Blind Signature

Based on the design in Section 4.8, we obtain a concrete UC blind signature protocol described in Figure 4.23 and Figure 4.24. Now we describe the CRS generation, the choice of parameters and also communication efficiency.

Common reference string generation. The common reference string $crs = \{\mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H}\}$. Here $\langle \mathsf{n}, \mathsf{g} \rangle$ is a public key for Paillier encryption; K is a public key for an equivocal commitment; $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$ is a part of public key for Okamoto signature; $\langle Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H} \rangle$ is Pedersen commitments public key. The parameters generated as follows.

$$crs = \langle \mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H} \rangle$$

$\boxed{S}$		$\boxed{U}$
$vk = \langle X = g_2^x \rangle$		$vk = \langle X = g_2^x \rangle$
$sk = \langle x \rangle$		$m \in \mathbb{Z}_p$
		$K_1 \xleftarrow{\$} \mathcal{Z}^*_{\mathsf{n}^2};\ \mu_1 \xleftarrow{\$} \mathcal{Z}^*_{\mathsf{n}}$
		$C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$
	$\xleftarrow{\quad C_1, W \quad}$	$t, s \xleftarrow{\$} \mathbb{Z}_p; W \leftarrow g_1^{mt} u_1^t v_1^{st}$
$K_2 \xleftarrow{\$} \mathcal{Z}^*_{\mathsf{n}^2}$		
	$\xrightarrow{\quad K_2 \quad}$	$K \leftarrow K_1 K_2 \bmod \mathsf{n}^2$
		$\theta \leftarrow (mt \bmod p) + t2^{\kappa} + (st \bmod p)2^{2\kappa}$
		$A_\theta, B_\theta \xleftarrow{\$} \mathcal{Z}^*_{\mathsf{n}};\ E_\theta \leftarrow K^{\theta}(A_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$
		$a_i \xleftarrow{\$} \pm[0, 2^{\lambda_0 + \lambda_U + \ell_p}],\ i = 1, 2, 3$
		$\vartheta \leftarrow a_1 + a_2 2^{\kappa} + a_3 2^{2\kappa}$
		$\widehat{E}_\theta \leftarrow K^{\vartheta}(B_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$
		$\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$
		$\mu_2 \xleftarrow{\$} \mathcal{Z}_Q; \omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$
$C_1 \overset{?}{=} \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$	$\xleftarrow{\langle K_1, \mu_1 \rangle, \langle E_\theta, C_2 \rangle}$	$C_2 \leftarrow \mathbf{g}^{\omega_2} \mathbf{h}_2^{\mu_2}$
$d_U \xleftarrow{\$} \{0, 1\}^{\lambda_U}$		
$r \xleftarrow{\$} \mathbb{Z}_p$ s.t. $x + r \not\equiv 0 \bmod p$		
$\chi \xleftarrow{\$} \pm[0, 2^{\lambda_0 + \lambda_S + \ell_p}]$		
$A_x, B_x \xleftarrow{\$} \mathcal{Z}^*_{\mathsf{n}}, l \xleftarrow{\$} \mathbb{Z}_p$		
$Y \leftarrow (W v_1^l)^{\frac{1}{x+r}}, \widehat{Z} \leftarrow Y^{\chi}$		
$\widehat{X} \leftarrow g_2^{\chi}$		
$E_x \leftarrow \mathsf{g}^x (A_x)^{\mathsf{n}} \bmod \mathsf{n}^2$		
$\widehat{E}_x \leftarrow \mathsf{g}^{\chi} (B_x)^{\mathsf{n}} \bmod \mathsf{n}^2$		
$\mu_3 \xleftarrow{\$} \mathcal{Z}_Q;\ \omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x)$		
$C_3 \leftarrow \mathbf{g}^{\omega_3} \mathbf{h}_3^{\mu_3}$		

(to be continued in Figure 4.24)

Figure 4.23: Blind signature generation protocol (part 1).

$$
\begin{array}{lcl}
\boxed{S} & & \boxed{U} \\
\hline
\textit{(continued from Figure 4.23)} & & \\
 & \xrightarrow{\langle E_x, C_3\rangle, d_U} & b_1 \leftarrow a_1 + d_U \cdot (mt \bmod p) \\
 & & b_2 \leftarrow a_2 + d_U \cdot t \\
 & & b_3 \leftarrow a_3 + d_U \cdot (st \bmod p) \\
 & & F_\theta \leftarrow B_\theta (A_\theta)^{d_U} \bmod \mathsf{n} \\
\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W});\ C_2 \stackrel{?}{=} \mathbf{g}^{\omega_2}\mathbf{h}_2^{\mu_2} & \xleftarrow{\langle b_1, b_2, b_3, F_\theta\rangle, \langle \widehat{E}_\theta, \widehat{W}, \mu_2\rangle, d_S} & d_S \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda_S} \\
E_\theta \in^? \mathcal{Z}^*_{\mathsf{n}^2},\ W \in^? \mathbb{G}_1 & & \\
b_i \in^? \pm[0, 2^{\lambda_0 + \lambda_U + \ell_p + 1}],\ i = 1,2,3 & & \\
g_1^{b_1} u_1^{b_2} v_1^{b_3} \stackrel{?}{=} \widehat{W} W^{d_U} & & \\
K \leftarrow K_1 K_2 \bmod \mathsf{n}^2 & & \\
\eta \leftarrow b_1 + b_2 2^{\kappa} + b_3 2^{2\kappa} & & \\
K^{\eta}(F_\theta)^{\mathsf{n}} \stackrel{?}{=} \widehat{E}_\theta (E_\theta)^{d_U} \bmod \mathsf{n}^2 & & \\
\delta_x \leftarrow \chi + d_S \cdot x, & & \\
F_x \leftarrow B_x (A_x)^{d_S} \bmod \mathsf{n} & \xrightarrow{\langle \delta_x, F_x\rangle, \langle Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x, \mu_3\rangle} & \omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x) \\
 & & C_3 \stackrel{?}{=} \mathbf{g}^{\omega_3}\mathbf{h}_3^{\mu_3} \\
 & & E_x \in^? \mathcal{Z}^*_{\mathsf{n}^2}; \delta_x \in^? \pm[0, 2^{\lambda_0 + \lambda_S + \ell_p + 1}] \\
 & & g_2^{\delta_x} \stackrel{?}{=} \widehat{X} X^{d_S} \\
 & & \mathsf{g}^{\delta_x}(F_x)^{\mathsf{n}} \stackrel{?}{=} \widehat{E}_x (E_x)^{d_S} \bmod \mathsf{n}^2 \\
 & & Y^{\delta_x} \stackrel{?}{=} \widehat{Z}(W v_1^l Y^{-r})^{d_S} \\
 & & f, h \stackrel{\$}{\leftarrow} \mathbb{Z}_p;\ \varsigma \leftarrow Y^{\frac{1}{ft} \bmod p} \\
 & & \alpha \leftarrow X^f g_2^{fr};\ \beta \leftarrow s + \frac{l}{t} \bmod p \\
 & & V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h;\ V_2 \leftarrow X^{fh+r} g_2^{frh} \\
 & & \sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle \\
 & & \mathsf{Vrf}(crs, vk, m, \sigma) \stackrel{?}{=} 1 \\
\hline
 & & \text{output } (m, \sigma)
\end{array}
$$

Figure 4.24: Blind signature generation protocol (part 2).

First, we generate parameters for Paillier encryption: let p and q be random primes for which it holds $\mathsf{p} \neq \mathsf{q}$, $|\mathsf{p}| = |\mathsf{q}|$ and $\gcd(\mathsf{pq}, (\mathsf{p}-1)(\mathsf{q}-1)) = 1$; let $\mathsf{n} \leftarrow \mathsf{pq}$, and $\mathsf{g} \leftarrow (1+\mathsf{n})$; set $\langle \mathsf{n}, \mathsf{g} \rangle$ as a Paillier public key, and $\langle \mathsf{p}, \mathsf{q} \rangle$ as the X-trapdoor. Second, randomly select $\tau_{\mathsf{K}} \overset{\$}{\leftarrow} \mathcal{Z}_{\mathsf{n}}$ and compute $\mathsf{K} \leftarrow (\tau_{\mathsf{K}})^{\mathsf{n}^2} \bmod \mathsf{n}^3$; set K as an E-key, and τ_{K} as the E-trapdoor. Third, let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups as defined in Section 2.6. Randomly select $g_2 \overset{\$}{\leftarrow} \mathbb{G}_2$, $\tau_{u_2}, \tau_{v_2} \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $u_2 \leftarrow g_2^{\tau_{u_2}}, v_2 \leftarrow g_2^{\tau_{v_2}}$, and set $g_1 \leftarrow \psi(g_2)$, $u_1 \leftarrow \psi(u_2)$ and $v_1 \leftarrow \psi(v_2)$. Set $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$ as the public information, and $\langle \tau_{u_2}, \tau_{v_2} \rangle$ as the trapdoor. Fourth, we generate parameters for a Pedersen-like [Ped91] commitment scheme over an elliptic curve group: let $\mathbf{G} = \langle \mathbf{g} \rangle$ be a cyclic elliptic curve group of prime order Q; select $\tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3} \overset{\$}{\leftarrow} \mathcal{Z}_Q$ and compute $\mathbf{h}_2 \leftarrow \mathbf{g}^{\tau_{\mathbf{h}_2}}$, $\mathbf{h}_3 \leftarrow \mathbf{g}^{\tau_{\mathbf{h}_3}}$; select $\mathcal{H}$ from a collision-resistent hash family $\mathcal{H}$, i.e. $\mathcal{H} \leftarrow \mathcal{H}$, such that $\mathcal{H} : \{0,1\}^* \rightarrow \mathcal{Z}_Q$; set $\langle Q, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathbf{G}, \mathcal{H} \rangle$ as public information, and $\tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3}$ as the trapdoor. Finally, set $\mathsf{CRS} = \{\mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H}\}$, and discard the corresponding trapdoors $\{\mathsf{p}, \mathsf{q}; \tau_{\mathsf{K}}; \tau_{u_2}, \tau_{v_2}; \tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3}\}$.

Choice of parameter lengths. Let the length of each parameter p, n, Q be ℓ_p, ℓ_{n}, ℓ_Q respectively. In the protocol, $d_U < p$, $d_S < p$, i.e. $\lambda_U < \ell_p$, $\lambda_S < \ell_p$, and $\kappa > \lambda_0 + \lambda_U + \ell_p + 3$, $\ell_{\mathsf{n}} \geq 3\kappa$. The parameters should be selected so that the following requirements: (i) The 2SDH assumption holds over the bilinear group parameter $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$, (ii) The DLOG assumption holds over the elliptic curve cyclic group $\mathbf{G}$, (iii) The DCR assumption holds over $\mathcal{Z}^*_{\mathsf{n}^2}$. Based on the present state of the art with respect to the solvability of the above problems, a possible choice of the parameters is for example $\ell_p = 171$ bits, $\ell_{\mathsf{n}} = 1024$ bits, $\ell_Q = 171$ bits. Notice that we should avoid using elliptic curves that have small $p+1$

divisors and $p-1$ divisors apart from 2, which suffer from a recent attack on SDH-related assumptions by Cheon [Che06].

Communication efficiency. We count the bits of six flows for generating a UC blind signature as:

flow1: $3\ell_{\mathrm{n}} + \ell_p$

flow2: $2\ell_{\mathrm{n}}$

flow3: $2\ell_{\mathrm{n}} + \ell_{\mathrm{n}} + 2\ell_{\mathrm{n}} + \ell_Q$

flow4: $2\ell_{\mathrm{n}} + \ell_Q + \lambda_U$

flow5: $3(\lambda_0 + \lambda_U + \ell_p + 1) + \ell_{\mathrm{n}} + 2\ell_{\mathrm{n}} + \ell_p + \ell_Q + \lambda_S$

flow6: $(\lambda_0 + \lambda_S + \ell_p + 1) + \ell_{\mathrm{n}} + 5\ell_p + 2\ell_{\mathrm{n}} + \ell_Q$

Based on the parameters: $\lambda_0 = \lambda_U = \lambda_S = 80$bits, $\ell_p = 171$bits, $\ell_{\mathrm{n}} = 1024$bits, $\ell_Q = 171$bits, $\kappa = 341$bits, and each element in $\mathbb{G}_1$ is with length of ℓ_p, we can compute the whole communication which is about 22.3 Kbits, i.e. less than 3 Kbytes.

Signature length. The length of signature $\sigma = \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$ is: $\ell_p + 6 \cdot \ell_p + \ell_p + \ell_p + 6 \cdot \ell_p = 15\ell_p = 2565$bits, i.e. about 2.6 Kbits. Here using the families of curves in [BLS04], we use group $\mathbb{G}_1$ where each element is 171bits and group $\mathbb{G}_2$ where each element 6×171bits.

BIBLIOGRAPHY

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. In *Cryptology ePrint Archive: Report 2010/133*, 2010. Available at http://eprint.iacr.org/2010/133.

[AO09] Masayuki Abe and Miyako Ohkubo. A framework for universally composable non-committing blind signatures. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 435–450. Springer, 2009.

[Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.

[Bea97] Donald Beaver. Plug and play encryption. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 1997.

[Bea98] Donald Beaver. Adaptively secure oblivious transfer. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 1998.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptology, 17(4):297–319, 2004.*

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003. Preliminary version appears in Financial Cryptography 2001.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–235. IEEE Computer Society, 2004. Full version at http://eprint.iacr.org/2003/239/.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Cryptology ePrint Archive, Report 2000/067*, December 2005. Latest version at http://eprint.iacr.org/2000/067/.

[Can07] Ran Canetti. Obtaining universally compoable security: Towards the bare bones of trust. In *ASIACRYPT*, pages 88–112, 2007. http://eprint.iacr.org/2007/475.

[CCK$^+$05] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using probabilistic i/o automata to analyze an oblivious transfer protocol. In *Cryptology ePrint Archive: Report 2005/452*, 2005. Available at `http://eprint.iacr.org/2005/452`.

[CCK$^+$07a] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic I/O automata. In *MIT CSAIL Technical Report MIT-CSAIL-TR-2006-060*, June 2007. Available at `http://people.csail.mit.edu/lcheung/task-pioa/task-PIOA-TR.pdf`.

[CCK$^+$07b] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Nancy A. Lynch, and Olivier Pereira. Compositional security for task-PIOAs. In *CSF*, pages 125–139. IEEE Computer Society, 2007.

[CCK$^+$08] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Analyzing security protocols using time-bounded task-PIOAs. *Discrete Event Dynamic Systems*, 18(1):111–159, 2008.

[CCLP07] Ran Canetti, Ling Cheung, Nancy Lynch, and Olivier Pereira. On the role of scheduling in simulation-based security. In *7th International Workshop on Issues in the Theory of Security (WITS'07)*, 2007.

[CDD$^+$04] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *J. Cryptology*, 17(3):153–207, 2004.

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2009.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, pages 199–203. Plemum Press, 1982.

[Che06] Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003.

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient blind signatures without random oracles. In Carlo Blundo and Stelvio Cimato, editors, *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2004.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002. Full version at http://eprint.iacr.org/2002/140/.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590. Springer, 2007.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430. Springer, 2000.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000. Preliminary version appears at STOC 1991.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *STOC*, pages 426–437. ACM, 2003. Full version at http://www.brics.dk/~jg/.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, pages 644–654, 1976.

[DKM$^+$04] Anupam Datta, Ralf Küsters, John C. Mitchell, Ajith Ramanathan, and Vitaly Shmatikov. Unifying equivalence-based definitions of protocol security. In *2004 IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security (WITS 2004)*, 2004.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92. Springer, 2000.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2000.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002. Full version at http://www.brics.dk/RS/01/41/.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multi-party computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

[DNO08] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2008. Available at http://eprint.iacr.org/2008/220/.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2006. Full version at http://www.fischlin.de/.

[GKZ10] Juan Garay, Aggelos Kiayias, and Hong-Sheng Zhou. A framework for the sound specification of cryptographic tasks. In *CSF*, 2010. To appear. Available at http://eprint.iacr.org/2008/132/.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GMY04] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and universally composable committed oblivious transfer and applications. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2004.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *J. Cryptology*, 19(2):169–209, 2006. Preliminary version appears in Eurocrypt 2003.

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 505–523. Springer, 2009.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2007.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptology*, 13(1):31–60, 2000.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1997.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2007.

[KDMR08] Ralf Küsters, Anupam Datta, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. *J. Cryptology*, 21(4):492–546, 2008.

[Kim04] Kwangjo Kim. Lessons from Internet voting during 2002 FIFA WorldCup Korea/Japan(TM). In *DIMACS Workshop on Electronic Voting – Theory and Practice*, 2004.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2004.

[Küs06] Ralf Küsters. Simulation-based security with inexhaustible interactive Turing machines. In *CSFW*, pages 309–320. IEEE Computer Society, 2006.

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2006. Long version at `http://eprint.iacr.org/2005/435/`.

[KZ07] Aggelos Kiayias and Hong-Sheng Zhou. Trading static for adaptive security in universally composable zero-knowledge. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 2007.

[KZ08] Aggelos Kiayias and Hong-Sheng Zhou. Equivocal blind signatures and adaptive uc-security. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

[Lin03] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC*, pages 683–692. ACM, 2003. Full version at `http://www.cs.biu.ac.il/~lindell/`.

[Lin08] Andrew Y. Lindell. Efficient fully-simulatable oblivious transfer. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 52–70. Springer, 2008.

[Lin09] Andrew Y. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2009. Available at `http://eprint.iacr.org/2009/031/`.

[LMMS98] Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In Michael Mitzenmacher, editor, *STOC*, pages 179–188. ACM, 2009.

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Compositionality for probabilistic automata. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2003.

[LZ09] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 183–201. Springer, 2009.

[MMS03] Paulo Mateus, John C. Mitchell, and Andre Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 323–345. Springer, 2003.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991. Long version at `http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-511.pdf`.

[MRST06] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[Nie03] Jesper Buus Nielsen. On protocol security in the cryptographic model. *Dissertation Series DS-03-8, BRICS*, 2003. `http://www.brics.dk/DS/03/8/`.

[Nie05] Jesper Buus Nielsen. Universally composable zero-knowledge proof of membership. *Manuscript*, 2005.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99. Springer, 2006. Long version at `http://eprint.iacr.org/2006/102/`.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 1996.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251. ACM, 2004.

[PS05] Manoj Prabhakaran and Amit Sahai. Relaxing environmental security: Monitored functionalities and client-server computation. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 104–127. Springer, 2005.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008. Available at `http://www.cc.gatech.edu/~cpeikert/`.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, 2001.

[Sho04] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. In *Cryptology ePrint Archive: Report 2004/332*, 2004. Available at http://shoup.net/papers/.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions. In *FOCS 1982*, pages 80–91. IEEE, 1982.