5-1-2016

Resource-Efficient and Robust Distributed Computing

Mahnush Movahedi Meimandi

Mahnush Movahedi Meimandi

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Jared Saia , Chairperson

David Evans

Shuang Luan

Maxwell Young

Resource-Efficient and Robust Distributed Computing

by

Mahnush Movahedi Meimandi

B.S., Computer Engineering, Amirkabir University of Technology
M.S., Computer Engineering, Amirkabir University of Technology
M.S., Computer Science, University of New Mexico

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico
Albuquerque, New Mexico

May 2016

Acknowledgments

I want to give my sincere thanks to my advisor, Professor Jared Saia, who has always been supportive and helpful to me. He taught me all sorts of skills of doing research, from selecting research topics to collaborating with other researcher and excelling in presentations. Without him, my Ph.D. would never be this productive.

I would like to give my deep gratitude to other members on my Ph.D. committee, Professor David Evans at University of Virginia, Professor Shuang Luan at UNM, and Professor Maxwell Young at University of Mississippi for serving on my committee and providing invaluable comments and feedback on my dissertation. I also thank Shafi Goldwasser at MIT and Seth Pettie at University of Michigan for their advices on my dissertation.

Lastly, I want to give special thanks to my husband, Mahdi, and my parents who has always helped me get through those toughest times. Not only have they been my first source of support. I cannot find words to express my gratitude to them. I feel truly lucky to have them in my life.

Abstract

There has been a tremendous growth in the size of distributed systems in the past three decades. Today, distributed systems, such as the Internet, have become so large that they require highly scalable algorithms; algorithms that have asymptotically-small communication, computation, and latency costs with respect to the network size. Moreover, systems with thousands or even millions of parties distributed throughout the world is likely in danger of faults from untrusted parties.

In this dissertation, we study scalable distributed algorithms that can tolerate faults from untrusted parties. We consider three different security models: *adversarial*, *rational*, and *cost-competitive*. Throughout this work, we balance two important and often conflicting characteristics of distributed protocols: security and efficiency.

Our first contribution is building an asynchronous algorithm for secure *Multi-Party Computation (MPC)* over n players. MPC is a fundamental problem in distributed computing that allows a set of players to jointly compute an arbitrary function of their inputs, without revealing these inputs to each other. We propose a scalable MPC algorithm assuming strictly less than a $1/8$ fraction of the players are controlled by a static adversary. For any function f over a field that can be computed by a circuit with m gates, our algorithm requires each player to send a number of field elements and perform an amount of computation that is $\tilde{O}(\frac{m}{n} + \sqrt{n})$. This significantly improves over traditional algorithms, which require each player to both send a number of messages and perform computation that is $\Omega(nm)$.

To achieve this result, we define and solve a new problem called *threshold counting problem* in the asynchronous setting. Our solution is load-balanced, with computation, communication, and latency complexity of $O(\log n)$, and may be useful for designing other load-balanced applications in the asynchronous communication model.

As an important building block of MPC algorithms in the rational setting, we study the classical secret sharing problem in a model where all players are rational. To the best of our knowledge, all known mechanisms for this problem require each player to send $O(n)$ messages in expectation.

As our second contribution in this dissertation, we describe mechanisms for rational secret sharing. Our first result is a mechanism for n-out-of-n rational secret sharing that is Nash equilibrium,

rather than just ϵ-Nash equilibrium. Our second result is a mechanism for rational t-out-of-n secret sharing that is everlasting ϵ-Nash equilibrium. Both our mechanisms are scalable in the sense that they require each player to send only an expected $O(\log n)$ bits. Furthermore, the latency of these mechanisms is $O(\log n)$ in expectation, compared to $O(n)$ expected latency of the best known result due to Kol and Naor [KN08]. Both of our mechanisms are non-cryptographic and are not susceptible to backward induction.

Finally, we propose a new algorithm for interactive communication when the noise rate is unknown. We achieve this result by using resource-competitive approach, where the adversary is assumed to pay a certain cost for each corruption he makes to the protocol. Most secure distributed algorithms pay a huge cost just due to the possible existence of an adversary. We conjecture by ensuring that a protocol does not pay more than the adversary, one can achieve a highly efficient algorithm protocol.

Table of Contents

List of Figures

Chapter 1

Introduction

Today, companies, governments, and consumers depend on secure and reliable information systems. A new generation of cyber-attacks is costing millions and straining the structure of the Internet. Unfortunately, as technology becomes more complex, security and privacy threats also become more complicated. The primary motivation of this dissertation is to design distributed algorithms that can provably resist sophisticated attacks conducted by powerful adversaries.

As a part of this dissertation, we develop techniques for securing large-scale distributed systems that consist of millions or even billions of nodes. Such systems may service many countries, and be used as the building blocks for communication, computation, and storage of information. When designing protocols for large systems, resources should be used carefully. Our first goal in this dissertation is to design secure algorithms that are scalable: communication, computation, and latency costs are asymptotically small with respect to the system size.

Moreover, in practice it is not always the case that the behavior and specifically the budget of the adversary for corruption is known by the algorithm in advance. Having this unknown factor in mind, we design an algorithm that adapts to the behavior of the adversary. Thus, our second goal in this dissertation is to design a protocol such that the cost of the algorithm is a slow-growing function of the budget of the adversary. Furthermore, we ensure that the algorithm is not paying too much if the adversary is not corrupting.

After describing our model, We briefly summarize the core contributions of this dissertation. These include scalable and efficient algorithms for three fundamental distributed problems: multi-party computation (MPC), rational secret sharing, and interactive communication.

1.1 Communication Models

Throughout this dissertation, we consider a network of n parties whose identities are common knowledge. We assume there is a channel between every pair of parties. We do not assume a broadcast channel. Communication can be via *synchronous* or *asynchronous* message passing. In a synchronous network, all parties have a common global clock and all the messages has pre-defined delays. In the asynchronous model, the sent messages may be arbitrarily delayed by the adversary. Latency in the asynchronous model is defined as the maximum length of any chain of messages (see [CD89, AW04]). We assume a synchronous model unless we explicitly say in the problem that the communication model is asynchronous.

1.2 Selfish and Adversarial Behavior

We study three secure distributed computing problems mentioned above in the three settings each with a different adversarial or selfish behavior.

1.2.1 Computation secure against an adversary

In this model, we assume the presence of an adversary who is willing to corrupt the protocol. Here we sketch some key parameters in this model.

Control over the parties. In the *fail-stop* corruption model, parties may crash randomly. *Passive* adversary can eavesdrop on the internal state of the corrupted parties, trying to obtain some information he is not entitled to. *Byzantine* (malicious or Active) adversary can additionally make the corrupted parties deviate in an arbitrary manner from the protocol specification. These parties are *bad* (i.e., Byzantine or dishonest) and the remaining parties are *good* (i.e., honest or semi-honest). The good parties run our algorithm, but the bad parties may deviate from the protocol specification. Byzantine model is the strongest adversarial model.

Control over the channel. The adversary might have different abilities to corrupt or eavesdrop on the communication channel between the parties. In a *secure channel*, we assume the adversary has no control over the channel and cannot read the bits on the channel. In *private unauthenticated channel*, the adversary can only flip the bits of the channel, but he cannot eavesdrop the bits. In *authenticated channel*, the adversary can eavesdrop the bits, but he has not the ability to corrupt them. Finally, in *unauthenticated insecure channel*, the adversary has the ability to corrupt or eavesdrop the communication.

Computational power. The adversary can be computationally-bounded or computationally-unbounded. *Computationally bounded* adversary is capable of only polynomial-time computation. In *computationally unbounded* model, we cannot assume any bound on the adversary's computational powers, or any cryptographic hardness assumptions. Throughout this dissertation, we assume the adversary is computationally unbounded.

Adaptivity. The adaptive adversary can take over players at any point of the protocol. The static adversary must select the set of bad parties at the start of the algorithm. We assume that the adversary is static.

1.2.2 Computation robust against Rational Behavior

We deploy game theory to model the rational behavior of the parties while computing a protocol together. In this model, the parties are participating in a game together, thus we call parties, players interchangeably. We assume each player has a utility function and can computes its payoff at the end of the game (protocol) and its outcome based on his utility function. In this model, rationality implies that every player always maximizes his utility. In other words, every player is motivated by maximizing his own payoff.

In this setting, the goal is to design a protocol (mechanism) that can build an equilibrium between the players. We say a protocol (mechanism) is a *Nash equilibrium* in the sense that no player can improve its utility by deviating from the protocol, given that all other players are following the protocol. Similarly, we define ϵ-Nash equilibrium when no player can gain more than ϵ utility by unilaterally deviating from it. Furthermore, the equilibrium is *everlasting* if after any history that is consistent with all players following the protocol, following the protocol continues to be an ϵ-Nash equilibrium.

1.3 Contributions

1.3.1 Scalable Multi-Party Computation

In secure multi-party computation (MPC), a set of parties, each having a secret value, want to compute a common function over their inputs, without revealing any information about their inputs other than what is revealed by the output of the function. Usually, a malicious adversary who makes the parties deviate arbitrarily from the protocol controls a certain fraction of the parties. Although much theoretical progress has been made in the MPC literature to achieve scalability in

the malicious setting, practical progress is slower. In particular, most known schemes suffer from either poor or unknown communication and computation costs in practice. Another challenge is that most large-scale distributed systems are composed of nodes with limited resources. This makes it of extreme importance to balance the protocol load across all parties involved.

In the Section 2, we first propose an efficient MPC protocol for computing arithmetic circuits in the synchronous communication model. Our protocol is secure against a Byzantine (malicious) adversary who can corrupt less than a 1/3 fraction of the parties. To achieve efficiency, we reduce the communication cost of our protocol by performing local communications in logarithmic-size groups of parties called quorums, where the number of malicious parties in each quorum is at most a certain fraction. Our paper is the first publication in a line of research on the use of communication locality in multi-party computation.

We also adapted our protocol to the asynchronous communication model. The asynchronous protocol is secure against a malicious adversary corrupting less than a 1/8 fraction of the parties. To deal with asynchrony problems arising from delayed inputs, we propose a novel distributed data structure called threshold counter. Surprisingly, our MPC algorithm requires each party to send only a sub-linear number of bits with respect to the network size. This significantly improves over traditional works, which require each party to both send a number of messages and perform computation that is linear in the network size.

1.3.2 Rational Secret Sharing

Secret sharing is one of the most fundamental problems in computer security, and is an important primitive in many cryptographic protocols. In this problem, a secret message is converted into n pieces called shares, which are dealt to n parties. The shares are generated in such a way that makes it possible for all or a fraction of parties to reconstruct the secret later from the shares, but no smaller subset of parties can reconstruct the secret.

Recently, there has been an interest in solving the rational secret sharing problem, where all parties are rational: they want to maximize their pay-offs from a game-theoretic perspective. Unfortunately, to the best of our knowledge, all previous mechanisms for this problem require each agent to send $O(n)$ messages in expectation; they do not scale well to large networks. In Section 3, we address this issue by describing scalable mechanisms for rational secret sharing that are designed for large n in the sense that the amount of communication and the latency of the protocol are a slow growing function of the number of players.

Our first result is a non-cryptographic mechanism for n-out-of-n rational secret sharing that is Nash equilibrium, rather than just ϵ-Nash equilibrium. Our second result is a cryptographic

mechanism for rational t-out-of-n secret sharing that is everlasting ϵ-Nash equilibrium. Both our mechanisms require each player to send an expected $O(\log n)$ bits and has $O(\log n)$ latency in expectation.

1.3.3 Interactive Communication over Noisy Channels

In interactive communication, a network protocol is run by two parties over a communication channel in order to compute a desired functionality. Such a protocol usually considers the channel as a reliable medium that always transfers information sent by either party correctly. This assumption is, however, not always valid in practice; channels are usually subject to some random or adversarial noise corrupting some fraction of the symbols transmitted between the parties. In such cases, a mechanism is required to simulate the original protocol using another protocol over a noisy channel such that it achieves the same outcome as the original protocol over an abstract noise-free channel.

In Section 4, we argue that, in most practical scenarios, the exact resource budget of the adversary in corrupting the communication is finite but unknown to the parties in advance. We then propose a simple and efficient technique for simulating the protocol when the noise rate is unknown. The parties run one block of the original protocol, as if there was no noise. Then, they verify whether an error has occurred or not by checking the some fingerprints of the block sent along with the block. We design an adaptive algorithm during which parties estimate the number of bit-flips so far and adapt their running protocol based on that.

If an adversary flips T bits, our algorithm sends $L + O\left(\sqrt{L(T+1)\log L} + T\right)$ bits in expectation and succeeds with high probability in L. It does so without any *a* priori knowledge of T. Assuming a conjectured lower bound by Haeupler, our result is optimal up to logarithmic factors. Our algorithm critically relies on the assumption of a private channel. We show that such an assumption is necessary when the amount of noise is unknown.

Chapter 2

Secure Multi-Party Computation in Large Networks

In *secure multi-party computation (MPC)*, a set of parties, each having a secret value, want to compute a common function over their inputs, without revealing any information about their inputs other than what is revealed by the output of the function.

Recent years have seen a renaissance in MPC, but unfortunately, the distributed computing community is in danger of missing out. In particular, while new MPC algorithms boast dramatic improvements in latency and communication costs, none of these algorithms offer significant improvements in the highly *distributed* case, where the number of parties is large.

This is unfortunate, since MPC holds the promise of addressing many important problems in distributed computing. How can peers in BitTorrent auction off resources without hiring an auctioneer? How can we design a decentralized Twitter that enables provably anonymous broadcast of messages. How can we create deep learning algorithms over data spread among large clusters of machines?

Most large-scale distributed systems are composed of nodes with limited resources. This makes it of extreme importance to *balance* the protocol load across all parties involved. Also, large networks tend to have weak admission control mechanisms which makes them likely to contain malicious nodes. Thus, a key variant of the MPC problem that we consider will be when a certain hidden fraction of the nodes are controlled by a malicious adversary.

2.1 Our Contribution

In this chapter, we describe general MPC protocols for computing arithmetic circuits. In terms of communication and computation costs per party, our protocols scale sublinearly with the number of parties and linearly with the size of the circuit.

To achieve sublinear communication and computation costs, our protocols critically rely on the notion of *quorums*. A quorum is a set of $O(\log n)$ parties, where the number of corrupted parties in each quorum is guaranteed not to exceed a certain fraction. We describe an efficient protocol for creating a sufficient number of quorums in the asynchronous setting.

To adapt to the asynchronous setting, we introduce the general problem of *threshold counting*. We show how this problem relates to the problem of dealing with arbitrarily-delayed inputs in our asynchronous MPC protocol, and then propose an efficient protocol for solving it.

When a protocol is concurrently executed alongside other protocols (or with other instances of the same protocol), one must ensure this composition preserves the security of the protocol. We show that our protocols are secure under such concurrent compositions by proving its security in the *universal composability (UC) framework* of Canetti [Can01].

2.1.1 Model

Consider n parties $P_1,...,P_n$ in a fully-connected network with private and authenticated channels. In our asynchronous protocol, we assume communication is via asynchronous message passing, so that sent messages may be arbitrarily and adversarially delayed. Latency (or running time) of a protocol in this model is defined as the maximum length of any chain of messages sent/received throughout the protocol (see [CD89, AW04]).

We assume a *malicious* adversary who controls an unknown subset of parties. We refer to these parties as *corrupted* and to the remaining as *honest*. The honest parties always follow our protocol, but the corrupted parties not only may share information with other corrupted parties but also can deviate from the protocol in any arbitrary manner, *e.g.* , by sending invalid messages or remaining silent.

We assume the adversary is *static* meaning that it must select the set of corrupted parties at the start of the protocol. We assume that the adversary is computationally-unbounded; thus, we make no cryptographic hardness assumptions.

2.1.2 Problem Statement

Multi-Party Computation. In the MPC problem, n parties, each holding a private input, want to jointly evaluate a deterministic n-ary function f over their inputs while ensuring:

1. Each party learns the correct output of f; and
2. No party learns any information about other parties' inputs other than what is revealed from the output.

Constraints for the Asynchronous Model. Consider a simple setting, where, the n parties send their inputs to a trusted party P who then locally computes f and sends the result back to every party. In the asynchronous setting, the MPC problem is challenging even with such a trusted party. In particular, since the t corrupted parties can refrain from sending their inputs to P, it can only wait for $n-t$ inputs rather than n inputs. Then, it can compute f over n inputs consisting of $n-t$ values received from the parties and t dummy (default) values for the missing inputs. Finally, the trusted party sends the output back to the parties. The goal of asynchronous MPC is to achieve the same functionality as the above scenario but without the trusted party P.

Quorum Building. A quorum is a set of $\Theta(\log n)$ parties, where the fraction of corrupted parties in this set is at most $t/n+\epsilon$ for a small positive constant ϵ. In the quorum building problem, there are n parties up to $t<n$ of whom may be corrupted. The goal is to ensure all parties agree on a set of n quorums such that each party is mapped to $O(\log n)$ quorums.

Threshold Counting. In this problem, there are n honest parties each with a flag bit initially set to 0. At least $\tau<n$ of the parties will eventually set their bits to 1. The goal is for all the parties to learn when the number of bits set to 1 becomes greater than or equal to τ.

2.1.3 Our Results

The main results of this chapter are summarized by the following theorems proved in Section 2.5. We consider an n-ary function, f, represented as an arithmetic circuit of depth d with m gates.

Theorem 2.1.1. There exist a universally-composable protocol that with high probability solves the synchronous MPC problem and has the following properties:

- It is secure against $t<(1/3-\epsilon)n$ corrupted parties, for some fixed $\epsilon>0$.
- Each party sends $\tilde{O}(m/n+\sqrt{n})$ field elements.

- Each party performs $\tilde{O}(m/n + \sqrt{n})$ computations.
- The expected running time is $O(d \,\text{polylog}(n))$.

Theorem 2.1.2. There exist a universally-composable protocol that with high probability solves the asynchronous MPC problem and has the following properties:

- It is secure against $t < (1/8 - \epsilon)n$ corrupted parties, for some fixed $\epsilon > 0$.
- Each party sends $\tilde{O}(m/n + \sqrt{n})$ field elements.
- Each party performs $\tilde{O}(m/n + \sqrt{n})$ computations.
- The expected running time is $O(d \,\text{polylog}(n))$.

Chapter Organization. In Section 2.2, we discuss related work. In Section 3.6, we define our notation and discuss the building blocks used in our protocols. We present our MPC protocols in Section 2.4. In Section 2.5, we prove the security of our MPC protocols. Section 2.6 is a self-contained presentation of the threshold counting problem and our solution to this problem. In Section 2.7, we describe an asynchronous protocol for the quorum building problem. Finally, we conclude in Section 2.8 and discuss future directions.

2.2 Related Work

Due to the large body of work, we do not attempt a comprehensive review of the MPC literature here, but rather focus on seminal work and, in particular, schemes that achieve sublinear per-party communication costs. The MPC problem was first described by Yao [Yao82]. He described an algorithm for MPC with two parties in the presence of a semi-honest adversary. Goldreich *et al.* [GMW87] propose the first MPC protocol that is secure against a malicious adversary. This work along with [CDG88, GHY88] are all based on cryptographic hardness assumptions. These were later followed by several cryptographic improvements [BMR90, GRR98, CFGN96].

In a seminal work, Ben-Or *et al.* [BGW88] show that every function can be computed with information-theoretic security in the presence of a semi-honest adversary controlling less than half of the parties, and in the presence of a malicious adversary controlling less than a third of the parties. They describe a protocol for securely evaluating an arithmetic circuit that represents the function.

This work was later improved in terms of both communication and computation costs in [CCD88, Bea91, GRR98]. Unfortunately, these methods all have poor communication scalability. In particular, if there are n parties involved in the computation, and the function f is represented by

a circuit with m gates, then these algorithms require each party to send a number of messages and perform a number of computations that is $\Omega(nm)$.

These were followed by several improvements to the cost of MPC, when m (*i.e.* , the circuit size) is much larger than n [DI06, DN07, DIK+08]. For example, the protocol of Damgård *et al.* [DIK+08] incurs computation and communication costs that are $\tilde{O}(m)$ plus a polynomial in n. Unfortunately, the additive polynomial in these algorithms is large (at least $\Omega(n^6)$) making them impractical for large n. One may argue that for large circuits the circuit-dependent complexity dominates the polynomial complexity. However, we believe there are many useful circuits such as the ones used in [MSZ15, HKI+12] which have relatively small number of gates.

Asynchronous MPC. Foundational work in asynchronous MPC was presented by Ben-Or *et al.* [BCG93]. They adapt the protocol of [BGW88] to the asynchronous setting and show that asynchronous MPC is possible for up to $n/3$ fail-stop faults and up to $n/4$ malicious faults. Improvements were made by Srinathan and Rangan [SR00] and Prabhu *et al.* [PSR02] with a final communication cost of $O(n^3)$ per multiplication achieved by Beerliová-Trubíniová and Hirt [BTH07] for perfectly-secure asynchronous MPC with the optimal resiliency bound of up to $n/4$.

Damgård *et al.* [DGKN09] describe a perfectly-secure MPC that guarantees termination only when the adversary allows a preprocessing phase to terminate. However, their protocol is not fully asynchronous, as they assume a few synchronization points; hence, they can achieve a resiliency bound of up to $n/3$.

Choudhury *et al.* [CHP13] propose an amortized asynchronous MPC protocols with linear communication complexity per multiplication gate meaning that the communication done by an individual party for each gate does not grow with the number of parties. This protocol is unconditionally-secure against up to $n/4$ corrupted parties with a small failure probability. In this chapter, we are directly addressing the third open problem of [CHP13] as we quote here:

"If one is willing to reduce the resilience t from the optimal resilience by a constant fraction, then by using additional techniques like packed secret sharing, committee election and quorum forming, one can achieve additional efficiency in the synchronous MPC protocols, as shown in [...]. It would be interesting to see whether such techniques can be used in the asynchronous settings to gain additional improvements."

MPC with Sublinear Overhead. We first introduced the notion of using quorums to decrease message cost in MPC in a brief announcement [DKMS12]. In that paper, we described a synchronous protocol with bit complexity of $\tilde{O}(m/n + \sqrt{n})$ per party that can tolerate a computationally

unbounded adversary who controls up to $(1/3-\epsilon)$ fraction of the parties for any fixed positive ϵ. As network size scales, it becomes infeasible to require each party to communicate with all other parties.

The current chapter is the detailed version of our later extended abstract [DKMS14], where we described algorithms to improve [DKMS12] by handling asynchronous communication. One important challenge in the asynchronous communication model is to ensure that at least $n-t$ inputs are committed to, before the circuit evaluation. To address this issue we introduce and solve the *threshold counting problem.*

Boyle *et al.* [BGT13] describe a synchronous MPC protocol for evaluating arithmetic circuits. The protocol is computationally-secure against an adversary corrupting up to $(1/3-\epsilon)$ fraction of parties, for some fixed positive ϵ. Similar to [DKMS12], the protocol of [BGT13] also uses quorums to achieve sublinear per-party communication cost. Interestingly, the communication cost of this protocol is independent of circuit size. This is achieved by evaluating the circuit over encrypted values using a *fully-homomorphic encryption (FHE)* scheme [Gen09]. Unfortunately, the protocol is not fully load-balanced as it evaluates the circuit using only one quorum (called the supreme committee). The protocol requires each party to send $\mathsf{polylog}(n)$ messages of size $\tilde{O}(n)$ bits and requires $\mathsf{polylog}(n)$ rounds.

Chandran *et al.* [CCG+14] address two limitations of the protocol of [BGT13]: tolerating an adaptive adversary and achieving optimal resiliency (*i.e.*, $t < n/2$ malicious parties). They replace the common reference string assumption of [BGT13] with a different setup assumption called symmetric-key infrastructure, where every pair of parties share a uniformly-random key that is unknown to other parties. The authors also show how to remove the SKI assumption at a cost of increasing the communication locality by $O(\sqrt{n})$. Although this protocol provides small communication locality, the bandwidth cost seems to be super-polynomial due to large message sizes.

Boyle *et al.* [BCP14] describe a scalable technique for secure computation of RAM programs [GO96] in large networks by performing local communications in quorums of parties. For securely evaluating a RAM program Π, their protocol incurs a total communication and computation of $\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$ while requiring $\tilde{O}(|x|+Space(\Pi)/n)$ memory per party, where $Time(\Pi)$ and $Space(\Pi)$ are time and space complexity of Π respectively, and $|x|$ denotes the input size.

Table 2.1: Recent MPC results with sublinear communication costs

Protocol	Security	Resiliency Bound	Async?	Assumes Broadcast Channel?	Total Message Complexity	Total Computation Complexity	Latency	Msg Size	Load-Balanced?
[BGT13]	Crypto	$(1/3-\epsilon)n$	No	No	$\tilde{O}(n)$	$\tilde{\Omega}(n)+\tilde{\Omega}(\kappa m d^3)^{\dagger}$	$\tilde{O}(1)$	$O(n\ell\cdot \mathsf{polylog}(n))$	No
[BCP14]	Perfect	$(1/3-\epsilon)n$	No	Yes	$\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$	$\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$	$\tilde{O}(Time(\Pi))$	$O(\ell)$	Yes
[CCG+14]	Crypto‡	$n/2$	No	No	$O(n\log^{1+\epsilon} n)$ or $O(n\sqrt{n}\log^{1+\epsilon} n)$	$\Omega(n\log^{1+\epsilon} n)$ or $\Omega(n\sqrt{n}\log^{1+\epsilon} n)$	$O(\log^{\epsilon'} n)$	$\Omega(\log^{\log n} n)$ or $\Omega(\sqrt{n}^{\log n})$	Yes
Our result (sync)	Perfect	$(1/3-\epsilon)n$	No	No	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d+\mathsf{polylog}(n))$	$O(\ell)$	Yes
Our result (async)	Perfect	$(1/8-\epsilon)n$	Yes	No	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d+\mathsf{polylog}(n))$	$O(\ell)$	Yes

Parameters: n is the number of parties; ℓ is the size of a field element; d is the depth of the circuit; κ is the the security parameter; ϵ,ϵ' are the positive constants; $Time(\Pi)$ is the worst-case running time of RAM program Π.

Notes:

†The cost is calculated based on the FHE scheme of [BGV12].

‡Assumes a symmetric-key infrastructure. However, unlike the rest, this protocol is secure against an adaptive adversary.

In Table 2.1, we review recent MPC results that provide sublinear communication locality. All of these results rely on some quorum building technique for creating a set of quorums each with honest majority.

Counting Networks. The threshold counting problem can be solved in a load-balanced way using *counting networks* that were first introduced by Aspnes *et al.* [AHS91]. Counting networks are constructed from simple two-input two-output computing elements called *balancers* connected to one another by wires. A counting network can count any number of inputs even if they arrive at arbitrary times, are distributed unevenly among the input wires, and propagate through the network asynchronously.

Aspnes *et al.* [AHS91] establish an $O(\log^2 n)$ upper bound on the depth complexity of counting networks. Since the latency of counting is dependent on the depth of the network, minimizing this depth has been the goal of many papers in this area. A simple explicit construction of an $O(c^{\log^* n} \log n)$-depth counting network (for some positive constant c), and a randomized construction of an $O(\log n)$-depth counting network that works with high probability are described by Klugerman and Plaxton in [KP92, Klu95]. These constructions use the AKS sorting network [AKS83] as a building block. While this sorting network and the resulting counting networks have $O(\log n)$ depth and require each party (or gate in their setting) to send $O(\log n)$ messages, large hidden constants render them impractical.

2.3 Preliminaries

In this section, we define standard terms, notation, and known building blocks used throughout this chapter.

Notation. We denote the set of integers $\{1,\ldots,n\}$ by $[n]$. We say an event occurs *with high probability*, if it occurs with probability at least $1 - 1/n^c$, for some $c > 0$ and sufficiently large n. A protocol is called *t-private* if no coalition of t corrupted parties can learn anything more than what is implied by their private inputs and the protocol output. A protocol is called *t-resilient* if no set of t or less parties can influence the correctness of the outputs of the remaining parties.

We also assume that all arithmetic operations in the circuit are carried out over a finite field $\mathbb{F}$. The size of $\mathbb{F}$ depends on the specific function to be computed and is always $\Omega(\log n)$. All of the messages transmitted by our protocol are logarithmic in $\mathbb{F}$ and n.

Let r be a value chosen uniformly at random from $\mathbb{F}$ and $\widehat{x} = x + r$, for any $x \in \mathbb{F}$. In this case, we say x *is masked with* r and we refer to r and $\widehat{x}$ as the *mask* and the *masked value* respectively.

Universal Composability Framework. When a protocol is executed several times possibly concurrently with other protocols, one requires to ensure this composition preserves the security of the

protocol. This is because an adversary attacking several protocols that run concurrently can cause more harm than by attacking a *stand-alone* execution, where only a single instance of one of the protocols is executed.

One way to ensure this is to show the security of the protocol in the *universal composability (UC)* framework of Canetti [Can01]. A protocol that is secure in the UC framework is called *UC-secure*. We describe this framework in Section 2.5.

Verifiable Secret Sharing. An (n,t)-*secret sharing* scheme is a protocol in which a dealer who holds a secret value shares it among n parties such that any set of $t < n$ parties cannot gain any information about the secret, but any set of at least $t+1$ parties can reconstruct it. An (n,t)-*verifiable secret sharing (VSS)* scheme is an (n,t)-secret sharing scheme with the additional property that after the sharing stage, a dishonest dealer is either disqualified or the honest parties can reconstruct the secret, even if shares sent by dishonest parties are spurious. When we say a set of shares of a secret are *valid*, we mean the secret can be uniquely reconstructed solely from the set of shares distributed among the parties.

In this chapter, we use the $(\lceil n/3 \rceil - 1)$-resilient VSS scheme of Ben-Or *et al.* [BGW88] for the synchronous setting and the $(\lceil n/4 \rceil - 1)$-resilient VSS scheme of Ben-Or *et al.* [BCG93] for the asynchronous setting. When run among n parties, both protocols incur $\text{poly}(n)$ communication cost and $O(1)$ latency. We refer to the sharing stages of these protocols as VSS-Share and AVSS-Share, and to their reconstruction stages as VSS-Reconst and AVSS-Reconst, respectively.

Classic MPC. Our main protocols rely on the classic $(\lceil n/3 \rceil - 1)$-resilient MPC protocol of Ben-Or *et al.* [BGW88] for the synchronous setting and the classic $(\lceil n/4 \rceil - 1)$-resilient MPC protocol of Ben-Or *et al.* [BCG93] for the asynchronous setting. When run among n parties to compute a circuit with d gates, both protocols send $\text{poly}(n)$ bits and incur a latency of $O(d)$. We refer to the former protocol as CMPC and to the latter as ACMPC.

In this chapter, we use the above VSS and classic MPC protocols only among logarithmic-size groups of parties and only for computing logarithmic-size circuits. Thus, the communication overhead per invocation of these protocols will be $\text{polylog}(n)$.

Byzantine Agreement. In the *Byzantine agreement* problem, each party is initially given an input bit. All honest parties must agree on a bit which coincides with at least one of their input bits.

When parties only have access to secure pairwise channels, a protocol is required to ensure secure (reliable) broadcast. This guarantees all parties receive the same message even if the

broadcaster (dealer) is dishonest and sends different messages to different parties. Every time a broadcast is required in our protocols, we use the Byzantine agreement algorithms of Feldman and Micali [FM88]. We refer to their $(\lceil n/3 \rceil - 1)$-resilient synchronous algorithm as BA and to their $(\lceil n/4 \rceil - 1)$-resilient asynchronous algorithm as ABA. When all parties participating in a run of a broadcast protocol receive the same message, we say these messages are *consistent*.

2.4 Our Protocols

We now describe our protocols for scalable MPC in large networks. Throughout this section, we consider the network model defined in Section 2.1.1. We first describe our synchronous protocol, and then adapt this protocol to the asynchronous setting.

We assume that the parties have an arithmetic circuit C computing f; the circuit consists of m addition and multiplication gates. For convenience of presentation, we assume each gate has in-degree and out-degree 2.[1] For any two gates x and y in C, if the output of x is input to y, we say that x is a *child* of y and that y is a *parent* of x. We assume the gates of C are numbered $1,2,\ldots,m$, where the gate numbered 1 is the output (root) gate.

2.4.1 Synchronous MPC

The high-level idea behind our protocols is to first create a sufficient number of quorums and assign to each gate in the circuit one of these quorums. Then, for each party P_i holding an input $x_i \in \mathbb{F}$, P_i secret-shares x_i among all parties in the quorum associated with the i-th input gate. We refer to such a quorum as an *input quorum*.

Next, the protocol evaluates the circuit gate-by-gate starting from input gates. Each gate is jointly evaluated by parties of the quorum associated with this gate over the secret-shared inputs provided by its children. In a similar way, the result of the gate is then used as the input to the computation of the parent gate. Finally, the quorum associated with the root gate, constructs the final result and sends it to all parties via a binary tree of quorums.

This high-level idea relies on solutions to the following main problems.

Quorum Building. Creating a sufficient number of quorums. In Section 2.7, we describe a randomized protocol called Build-Quorums that achieves this goal with high probability.

[1] Our protocol works, with minor modifications, for gates with arbitrary constant fan-in and fan-out.

Protocol 1 Synchronous MPC

1. **Quorum Building.** All parties run Build-Quorums to agree on n good quorums $Q_1,...,Q_n$. The i-th gate of C is assigned to $Q_{(i \bmod n)}$, for all $i \in [m]$.
2. **Input Commitment.** For all $i \in [n]$, party P_i holding an input value $x_i \in \mathbb{F}$ runs the following steps concurrently:
   a) Pick a uniformly random element $r_i \in \mathbb{F}$, set $\widehat{x} = x_i + r_i$, and broadcast $\widehat{x}$ to Q_i.
   b) Run VSS-Share to secret-share r_i in Q_i.
3. **Circuit Evaluation.** All parties participate in a run of Circuit-Eval to securely evaluate C.
4. **Output Reconstruction.** For the output gate z, parties in Q_z,
   a) Run VSS-Reconst to reconstruct r_z from its shares.
   b) Set the circuit output message: $y \leftarrow \widehat{y}_z - r_z$.
   c) Send y to all parties in the Q_2 and Q_3.
5. **Output Propagation.** For every $i \in \{2,...,n\}$, parties in Q_i perform the following steps:
   a) Receive y from the $Q_{\lfloor i/2 \rfloor}$.
   b) Send y to all parties in Q_{2i} and Q_{2i+1}.

Circuit Evaluation. Securely evaluating each gate over secret-shared inputs by the parties inside a quorum. In Section 2.4.1.2, we describe a protocol called Circuit-Eval that achieves this goal.

Share Renewal. Sending the result of one quorum to another without revealing any information to any individual party or to any coalition of corrupted parties in both quorums. We solve this as part of our gate evaluation protocol described in Section 2.4.1.2.

Protocol 1 is our main protocol. When we say a party *VSS-shares* (or *secret-shares*) a value s in a quorum Q (or among a set of parties), we mean the party participates as the dealer with input s in the protocol VSS-Share with all parties in Q (or in the set of parties).

The protocol starts by running Build-Quorums to create n quorums $Q_1,...,Q_n$. Then, it assigns the gates of C to these quorums in the following way. The output gate of C is assigned to Q_1; then, every gate in C numbered i (other than the output gate) is assigned to $Q_{(i \bmod n)}$. For each gate $u \in C$, we let Q_u denote the quorum associated with u, y_u denote the output of u, r_u be a random element from $\mathbb{F}$, and $\widehat{y}_v$ denote the masked output of u, where $\widehat{y}_u = y_u + r_u$.

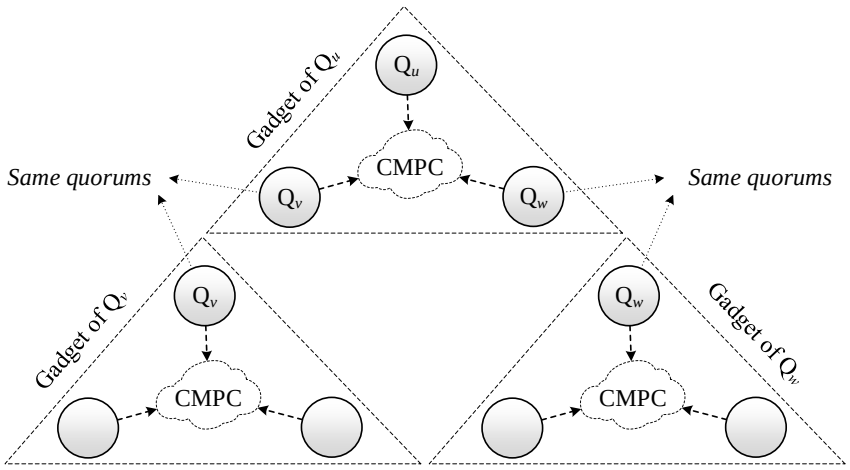

Figure 2.1: The gate gadgets for gate u and its left and right children

2.4.1.1 Input Commitment

Let Q_i be the quorum associated with party P_i who holds input x_i. At the start of our protocol, P_i samples a value r_i uniformly at random from $\mathbb{F}$, sets $\widehat{x} = x_i + r_i$, and broadcasts $\widehat{x}$ to all parties in Q_i. Next, P_i runs VSS-Share to secret-share r_i among all parties in Q_i.

2.4.1.2 Circuit Evaluation

The main idea for reducing the amount of communication required in evaluating the circuit is quorum-based gate evaluation. If each party participates in the computation of the whole circuit, it must communicate with all other parties. Instead, in quorum-based gate evaluation, each gate of the circuit is computed by a *gate gadget*. A gate gadget (see Figure 2.1) consists of three quorums: two *input quorums* and one *output quorum*. Input quorums are associated with the gate's children which serve inputs to the gate. The output quorum is associated with the gate itself and is responsible for creating a shared random mask and maintaining the output of the quorum for later use in the circuit. As depicted in Figure 2.1, these gate gadgets connect to form the entire circuit. In particular, for any gate u, the output quorum of u's gate gadget is the input quorum of the gate gadget for all of u's parents.

The parties in each gate gadget run CMPC among themselves to compute the gate operation. To ensure privacy is preserved, each gate gadget maintains the invariant that the value computed by the gadget is the value that the corresponding gate in the original circuit would compute, masked by a uniformly random element of the field. This random element is not known to any individual party.

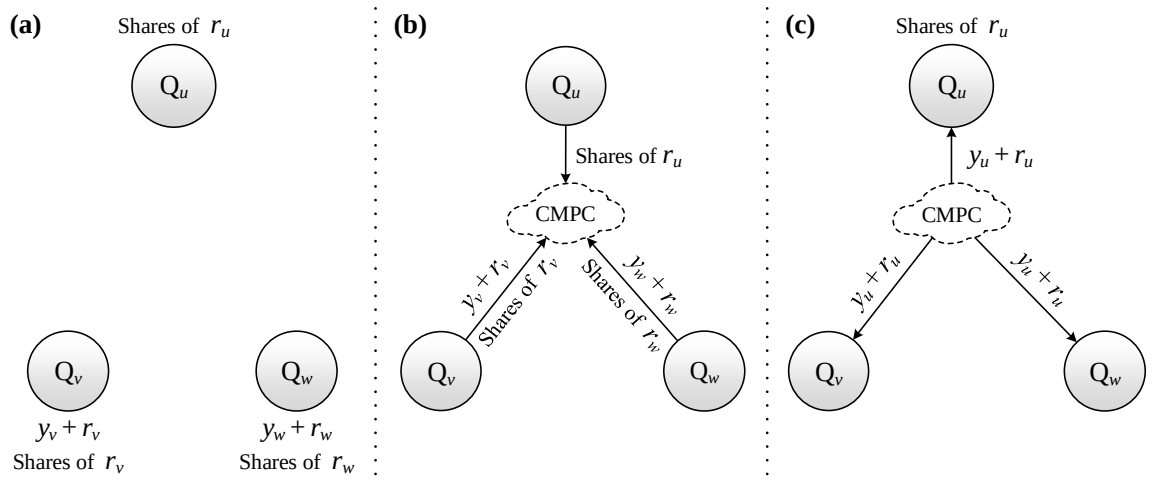

Figure 2.2: Evaluation of gate u: (a) generating r_u, (b) providing inputs to CMPC, (c) receiving the masked outputs

---

Protocol 2 Circuit-Eval

Goal. Given a circuit C, the protocol securely evaluates C.

For every gate $u \in C$ with children $v, w \in C$, parties in Q_u, Q_v, and Q_w perform the following steps to compute the gate functionality:

1. **Mask Generation.** Parties in Q_u run Gen-Rand to jointly generate a secret-shared random value $r_u \in \mathbb{F}$.
2. **MPC in Quorums.** The following parties participate in a run of CMPC with their corresponding inputs:
   - Every party in Q_u with his share of r_u.
   - Every party in Q_v with his input $(\widehat{y}_v$, his share of $r_v)$.
   - Every party in Q_w with his input $(\widehat{y}_w$, his share of $r_w)$.

Instead, shares of it are held by the members of the output quorum. Thus, the output quorum can participate as an input quorum for the evaluation of any parent gate and provide both the masked version of the inputs and shares of the mask. The gate gadget computation is performed in the same way for all gates in the circuit until the final output of the whole circuit is computed. After the input commitment step, for each input gate u, parties in Q_u know the masked input $\widehat{y}_u$, and each has a share of the mask r_u.

The first step of the circuit evaluation is to generate shares of uniformly random field elements for all gates. If a party is in a quorum at gate u, it generates shares of r_u, a uniformly random field element, by participating in the Gen-Rand protocol. These shares are needed as inputs to the subsequent run of CMPC.

Protocol 3 Gen-Rand

Goal. A set of parties $P_1,...,P_N$ in a quorum want to agree on a secret-shared value r chosen uniformly at random from $\mathbb{F}$.

1. For all $i \in [N]$, party P_i chooses $\rho_i \in \mathbb{F}$ uniformly at random and VSS-shares it among all N parties.
2. For every $j \in [N]$, let N' be the number of shares P_j receives from the previous step, and $\rho_{1j},...,\rho_{N'j}$ be these shares. P_j computes $r_j = \sum_{k=1}^{N'} \rho_{kj}$.

Next, parties form the gadget for each gate u to evaluate the functionality of the gate using Circuit-Eval. Let v and w be the left and right children of u respectively. The gate evaluation process is shown in Figure 2.2. The values y_v and y_w are the inputs to u, and y_u is the its output as it would be computed by a trusted party. Each party in Q_u has a share of the random element r_u via Gen-Rand. Every party in Q_v has the masked value $y_v + r_v$ and a share of r_v (respectively for Q_w).

As shown in Part (b) of Figure 2.2, all parties in the three quorums participate in a run of CMPC, using their inputs, in order to compute $\widehat{y}_u = y_u + r_u$. Part (c) of the figure shows the output of the gate evaluation after participating in CMPC. Each party in Q_u now learns $\widehat{y}_u$ as well a share of r_u. Therefore, parties in Q_u now have the input required for performing the computation of parents of u (if any). Note that both y_u and r_u remain unknown to any individual.

The gate evaluation is performed for all gates in C starting from the bottom to the top. The output of the quorum associated with the output gate in C is the output of the entire algorithm. This quorum will unmask the output via the output reconstruction step. The last step of the algorithm is to send this output to all parties. We do this via a complete binary tree of quorums, rooted at the output quorum.

2.4.1.3 Implementing the Gate Circuit

For every gate $u \in C$, the Circuit-Eval protocol requires a circuit (as we denote by C_u) for unmasking the masked inputs $\widehat{y}_v$ and $\widehat{y}_w$, computing u's functionality f_u over the unmasked inputs, and masking the output with the gate's random value r_u. This circuit is securely evaluated using the CMPC protocol by the quorum associated with u.

For unmasking an input, C_u requires a *reconstruction circuit*, which given a set of shares, outputs the corresponding secret. Since dishonest parties may send spurious shares, the circuit implements the error-correcting algorithm of Berlekamp and Welch [BW86] to fix such corruptions. Then, the resulting shares are given to an *interpolation circuit* which implements a simple polynomial interpolation. Figure 2.3 depicts the circuit for gate u.

We now briefly describe the error correcting algorithm of Berlekamp and Welch [BW86]. Let

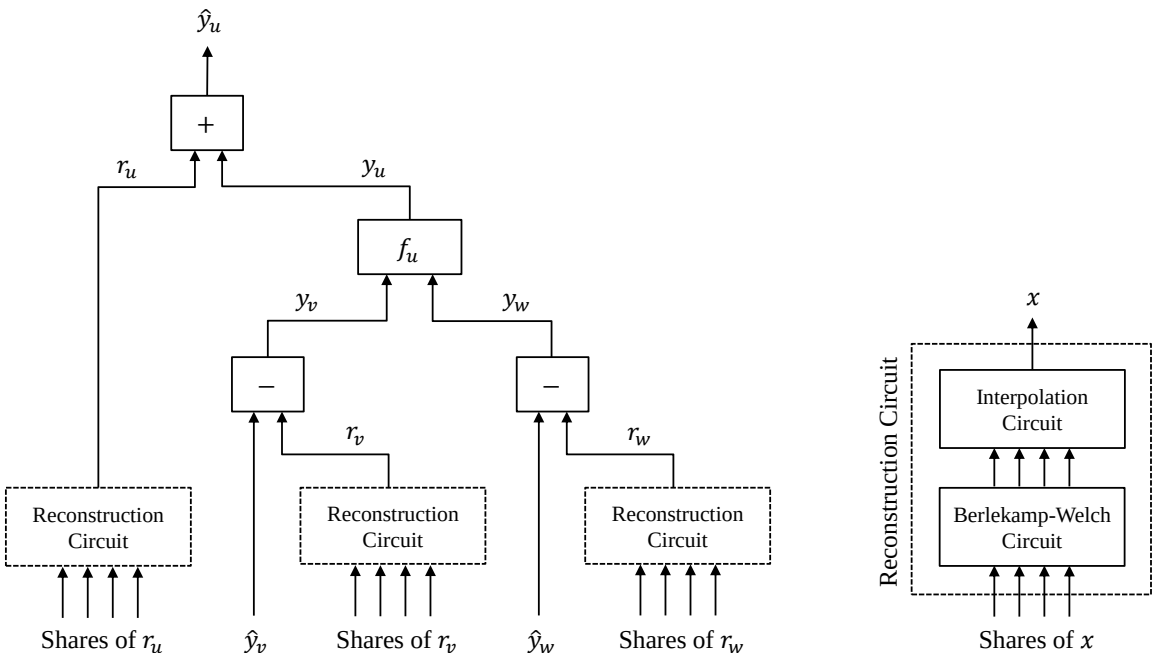

Figure 2.3: Circuit of gate u

$\mathbb{F}_p$ denote a finite field of prime order p, and $S = \{(x_1, y_1) \mid x_i, y_i \in \mathbb{F}_p\}_{i=1}^{\eta}$ be a set of η points, where $\eta - \varepsilon$ of them are on a polynomial $y = P(x)$ of degree τ, and the rest $\varepsilon < (\eta - \tau + 1)/2$ points are erroneous. Given the set of points S, the goal is to find the polynomial $P(x)$. The algorithm proceeds as follows. Consider two polynomials $E(x) = e_0 + e_1 x + ... + e_\varepsilon x^\varepsilon$ of degree ε, and $Q(x) = q_0 + q_1 x + ... + q_k x^k$ of degree $k \leq \varepsilon + \tau - 1$ such that $y_i E(x_i) = Q(x_i)$ for all $i \in [\eta]$. This defines a system of η linear equations with $\varepsilon + k = \eta$ variables $e_0, ..., e_\varepsilon, q_0, ..., q_k$ that can be solved efficiently using Gaussian elimination technique to get the coefficients of $E(x)$ and $Q(x)$. Finally, calculate $P(x) = Q(x)/E(x)$.

Since the Gaussian elimination algorithm over finite fields has $O(n^3)$ arithmetic complexity [Far88], the corresponding circuit has at most $O(n^3)$ levels. Since the interpolation circuit consists of at most $O(n^2)$ arithmetic operations (using the Lagrange's method [Abr74]), the overall depth of the reconstruction circuit will be $O(n^3)$.

2.4.2 Asynchronous MPC

We now adapt our synchronous protocol to the asynchronous communication model. We do this by modifying the following parts of Protocol 1:

1. We replace the synchronous subprotocols VSS-Share, VSS-Reconst, and CMPC with their

corresponding asynchronous versions AVSS-Share, AVSS-Reconst, and ACMPC, respectively. In Section 2.7, we describe a technique for adapting Build-Quorums to the asynchronous setting.

2. At the end of the Input Commitment stage, the protocol should wait for at least $n-t$ inputs before proceeding to the Circuit Evaluation stage. To this end, we introduce a new subprotocol called Wait-For-Inputs and invoke it right after step (b) of the Input Commitment stage. This protocol is described in Section 2.4.2.1.

3. Although the protocol ACMPC terminates with probability one, its actual running time (*i.e.* , the number of rounds until it terminates) is a random variable with expected value $O(D \log N)$, where N is the number of parties participating in the MPC and D is the circuit depth [BCG93]. Since we run m instances of ACMPC (one for each gate of C), we need a method that allows us to bound the running time of each gate, and thus to bound the expected running time of our asynchronous MPC protocol. We describe a simple method for achieving this in Section 2.4.2.2.

4. In the second step of Gen-Rand (Protocol 3), each party may receive less than N shares. This is because

2.4.2.1 Implementing Wait-For-Inputs

The protocol Wait-For-Inputs counts the number of inputs that are successfully received by their corresponding input quorums. This can be achieved using a solution to the threshold counting problem: Count the number of inputs successfully received by each input quorum and return once this number becomes greater than or equal to $n-t$. As a result of returning from Wait-For-Inputs, the main protocol resumes and starts the circuit evaluation procedure.

In Section 2.6, we provide a solution to the threshold counting problem. We refer to this protocol as Thresh-Count. This protocol creates a distributed tree structure called the *count tree* which is known to all parties and determines how the parties communicate with each other to count of the number of inputs.

Protocol 4 implements Wait-For-Inputs using our Thresh-Count algorithm. In Wait-For-Inputs, the role of each party in Thresh-Count (*i.e.* , each node in the count tree) is played by a quorum of parties. Once Thresh-Count terminates, the parties in each input quorum decide whether or not the corresponding inputs are part of the computation.

Protocol 4 Wait-For-Inputs

Goal. For every input quorum Q, all parties in a quorum Q wait until $n-t$ inputs are received by the input quorums. For each party $P_i \in Q$, P_i is initially holding two values $\widehat{x}$ and r_i, the i-th share of a random value r.

Each party $P_i \in Q$ does the following:

1. Run Thresh-Count with flag bit b_i initially set to zero.
2. If $\widehat{x}$ and r_i's are consistent and valid (based on the Byzantine agreement protocol and the verification stage of AVSS-Share respectively), set $b_i \leftarrow 1$ in Step $2(a)$ of Thresh-Count.
3. Upon receiving $\langle$Done$\rangle$ from the parent quorum, run ACMPC using b_i as the input. $d \leftarrow$ True if a $5/8$-fraction of the parties in Q have their b_i's set to one. Otherwise, $d \leftarrow$ False.
4. If $d =$ False, then $\widehat{x} \leftarrow$ Default and $r_i \leftarrow 0$.

When run among quorums, Thresh-Count requires the quorums to communicate with each other. We say a quorum Q sends a message M to quorum Q', when every (honest) party in Q sends M to every party in Q'. A party in Q' is said to have received M from Q if it receives M from at least $7/8$ of the parties in Q. When we say a party *broadcasts* a message M to a quorum Q, we mean the party sends M to every party in Q, and then, all parties in Q run BA over their messages to ensure they all hold the same message.

2.4.2.2 Bounding the Expected Running Time

Consider N parties in a quorum who want to jointly compute a circuit of depth D using the protocol ACMPC. Let X denote the random variable corresponding to the number of rounds until an instance of ACMPC terminates. From [BCG93], we have

$$\mathbf{E}[X] = O(D \log N).$$

Instead of running only one instance of ACMPC, we run $O(\log N)$ instances sequentially each for $2\mathbf{E}[X]$ rounds. The output corresponding to the first instance that terminates will be returned as the output of the gate. Using the Markov's inequality,

$$\Pr(X \geq 2\mathbf{E}[X]) \leq 1/2.$$

In each gate of C, each party also participates in a run of Gen-Rand which itself calls AVSS-Share. Similar to ACMPC, for each instance of AVSS-Share, we run $O(\log N)$ instances sequentially each for $2\mathbf{E}[X]$ rounds. The sharing corresponding to the first instance that terminates will be accepted by the parties.

Since $O(\log N)$ instances of ACMPC and AVSS-Share are executed in each gate, the computation of the gate correctly terminates after at most

$$2\mathbf{E}[X]\log N = O(D\log^2 N)$$

rounds with high probability. Since C has $m = \mathsf{poly}(n)$ gates, using union bound over all gates of C, our MPC algorithm correctly terminates with high probability. Finally, since C has depth d, the expected running time of our asynchronous MPC protocol is $O(Dd\log^2 N)$. In Section 2.4.1.2, we argued that the circuit computed by Circuit-Eval has depth $D = \mathsf{polylog}(n)$. Thus, the expected running time of our protocol is $O(d\,\mathrm{polylog}(n))$.

2.4.3 Remarks

As described in the introduction, the goal of MPC is to simulate a trusted third party in the computation of the circuit, and then send back the computation result to the parties. Let S denote the set of parties from whom input is received by the (simulated) trusted party. Recall that $|S| \geq n - t$.[1] Thus, for an arbitrary S, a description of S requires $\Omega(n)$ bits, and cannot be sent back to the parties using only a scalable amount of communication. Therefore, we relax the standard requirement that S be sent back to the parties. Instead, we require that at the end of the protocol each honest party learns the output of f; whether or not their own input was included in S; and the *size* of S.

Also note that although we have not explicitly included this in the input commitment step, it is very easy for the parties to compute the size of the computation set S. Once each input quorum Q_i has performed the third step of Wait-For-Inputs and has agreed on the flag $b_i = 1$, they can simply use an addition circuit to add these bits together, and then disperse the result. This is an MPC, all of whose inputs are held by honest parties, since each input flag b_i is jointly held by the entire quorum Q_i, and all the quorums are good. Thus, the computation can afford to wait for all n inputs and computes the correct sum.

In our both protocols, it may be the case that a party P participates more than one time in the quorums performing a single instance of the classic MPC. In such a case, we allow P to play the role of more than one different parties in CMPC and ACMPC, one for each quorum to which P belongs. This ensures that the fraction of corrupted parties in any instance of the classic MPC is always less than $1/3$ for the synchronous case and $1/4$ for the asynchronous case. We stress that CMPC and ACMPC both maintain privacy guarantees even in the face of gossiping coalitions of

[1] We allow $|S| > n - t$ because the adversary is not limited to delivering one message at a time; two or more messages may be received simultaneously.

constant size. Thus, each party will learn no information beyond the output and its own inputs after running these protocols.

2.5 Proof of Theorem 2.1.2

We first describe the UC framework in Section 2.5.1, and then give a sketch of our proof in Section 2.5.2. We prove the UC-security of Protocol 1 in sections 2.5.3 to 2.5.5. Finally, we calculate the resource costs of this protocol in Section 2.5.7.

2.5.1 The UC Framework

The UC framework is based on the *simulation paradigm* [Gol00], where the protocol is considered in two models: *ideal* and *real*. In the ideal model, the parties send their inputs to a trusted party who computes the function and sends the outputs to the parties. We refer to the algorithm run by the trusted party in the ideal model as the *functionality* of the protocol. In the real model, parties run the actual protocol that assumes no trusted party. We refer to a run of the protocol in one of these models as the *execution* of the protocol in that model.

A protocol $\mathcal{P}$ securely computes a functionality $F_{\mathcal{P}}$ if for every adversary $\mathcal{A}$ in the real model, there exists an adversary $\mathcal{S}$ in the ideal model, such that the result of a real execution of $\mathcal{P}$ with $\mathcal{A}$ is indistinguishable from the result of an ideal execution with $\mathcal{S}$. The adversary in the ideal model, $\mathcal{S}$, is called the *simulator*.

The simulation paradigm provides security only in the stand-alone model. To prove security under composition, the UC framework introduces an adversarial entity called the *environment*, denoted by $\mathcal{Z}$, who generates the inputs to all parties, reads all outputs, and interacts with the adversary in an arbitrary way throughout the computation. The environment also chooses inputs for the honest parties and gets their outputs when the protocol is finished.

A protocol is said to *UC-securely* compute an ideal functionality if for any adversary $\mathcal{A}$ that interacts with the protocol there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with a run of the protocol and $\mathcal{A}$, or with a run of the ideal model and $\mathcal{S}$.

Now, consider a protocol $\mathcal{P}$ that has calls to ℓ subprotocols $\mathcal{P}_1,...,\mathcal{P}_\ell$ which are already proved to be UC-secure. To facilitate the security proof of $\mathcal{P}$, we can make use of the *hybrid model*, where the subprotocols are assumed to be ideally computed by a trusted third-party. In other words, we replace each call to a subprotocol with a call to its corresponding functionality. This hybrid model is usually called the $(\mathcal{P}_1,...,\mathcal{P}_\ell)$-*hybrid* model. We say $\mathcal{P}$ is *UC-secure in the hybrid model* if $\mathcal{P}$ in

the hybrid model is indistinguishable by the adversary from $\mathcal{P}$ in the ideal model. The *modular composition theorem* [Can00] states that if $\mathcal{P}_1,\ldots,\mathcal{P}_\ell$ are all UC-secure, and $\mathcal{P}$ is UC-secure in the hybrid model, then $\mathcal{P}$ is UC-secure in the real model.

2.5.2 Proof Sketch

Before proceeding to the proof, we remark that the error probability in Theorem 2.1.2 comes entirely from the possibility that Build-Quorums or the threshold counting procedure may fail to output correct results. All other components of our protocol are deterministic and thus have no error probability. We also assume that, at the beginning of our MPC protocol, the parties have already agreed on n good quorums, and the threshold counting procedure is performed successfully.[1]

As in [Gol04], we refer to the security in the presence of a malicious adversary controlling t parties *t-security*. For every gate $u \in C$, let I_u denote the set of the corrupted parties in the quorum associated with u. Also, let I denote the set of all corrupted parties, where $|I| < t$.

Our goal is to prove the UC-security of Protocol 1. To do this, we must show two steps. Step 1) is to show that each of our subprotocols are UC-secure. Step 2) is to show that our protocol is UC-secure in the hybrid model. Once we show these two steps, then by the modular composition theorem, we conclude that our protocol is UC-secure in the real model.

In Lemma 2.5.7, we show Step 2, that the adversary can not distinguish the execution of the hybrid model from the ideal model.

We next describe our approach to Step 1, which is more challenging. For this step, we make use of a theorem that will help us show that our subprotocols are UC-secure. Kushilevitz *et al.* [KLR10] show Theorem 2.5.1. This theorem targets perfectly-secure protocols that are shown secure using a straight-line black-box simulator. A black-box simulator is a simulator that is given only oracle access to the adversary (see [Gol00] Section 4.5 for a detailed definition). Such a simulator is straight-line if it interacts with the adversary in the same way as real parties, meaning that it proceeds round by round without ever going back.

Theorem 2.5.1 ([KLR10])**.** Every protocol that is perfectly-secure in the stand-alone model and has a straight-line black-box simulator is UC-secure.

We first define the ideal functionalities shown in Table 2.2 that correspond to the subprotocols used in Protocol 1. We then prove that Protocol 1 is t-secure in the (F_{BA}, $F_{\text{VSS-Share}}$, $F_{\text{VSS-Reconst}}$,

[1] For simplicity, we assume the primitive Build-Quorums is run only once, and it does not run concurrently with other protocols.

Table 2.2: Ideal functionalities

Functionality	Implemented by
F_{BA}	Protocol BA
$F_{\text{VSS-Share}}$	Protocol VSS-Share
$F_{\text{VSS-Reconst}}$	Protocol VSS-Reconst
F_{CMPC}	Protocol CMPC
$F_{\text{Gen-Rand}}$	Protocol Gen-Rand
F_{Input}	Input Commitment stage of Protocol 1
$F_{\text{Circuit-Eval}}$	Protocol Circuit-Eval
F_{Output}	Output Reconstruction and Output Propagation stages of Protocol 1

F_{CMPC}, $F_{\text{Gen-Rand}}$, F_{Input}, $F_{\text{Circuit-Eval}}$, F_{Output})-hybrid model. Finally, we use Theorem 2.5.1 to infer the UC-security of Protocol 1.

In order to prove the t-security of Protocol 1 in the hybrid model, we first show that all of our subprotocols are UC-secure. Similar to the above approach, we first prove t-security of every subprotocol in its corresponding hybrid model using a straight-line black-box simulator, and then use Theorem 2.5.1 to infer its UC-security.

To prove the t-security of a protocol Π, we describe a simulator $\mathcal{S}_\Pi$ that simulates the real protocol execution by running a copy of Π in the ideal model. For each call to a secure subprotocol π, the simulator calls the corresponding ideal functionality F_π. A *view* of a corrupted party from execution of a protocol is defined as the set of all messages it receives during the execution of that protocol. At every stage of the simulation process, $\mathcal{S}_\Pi$ adds the messages received by every corrupted party in that stage to its view of the simulation. This is achieved by running a copy of Π for each corrupted party with its actual input as well as by running a copy of Π for each honest party with a dummy input.[1] The view of the adversary is then defined as the combined view of all corrupted parties.

2.5.3 Security of Input Commitment

Before proceeding to the proof of security for Input Commitment stage, we show the following auxiliary lemma.

Lemma 2.5.2. If a quorum Q sends to a quorum Q' a message M, it is eventually received by all honest parties in Q'.

[1] $\mathcal{S}_\Pi$ learns neither the actual inputs nor the actual outputs of the honest parties.

Protocol 5 F_{Input}

Goal. The functionality guarantees valid inputs are received by at least $n-t$ input quorums. Then, the functionality notifies all input quorums to proceed to the next stage of the protocol with either a valid input or a default input.

Functionality:

1. Wait to receive at least $n-t$ valid inputs from the set of all n parties. For every such input x_i, the functionality receives $\widehat{x_i} = x_i + r_i$ and r_i from party P_i where $i \in [n]$. Let S denote the set of parties whose inputs have been accepted (Note that if $P_i \in S$, then $\widehat{y_i}$ and r_i are valid).

2. If $P_i \notin S$, then define $\widehat{y_i} = 0$ and $r_i = 0$.

3. Broadcast $\langle \mathsf{Done} \rangle$ and $y_i \leftarrow x_i + r_i$ to Q_i and run $F_{\mathsf{VSS\text{-}Share}}$ to secret-share r_i in Q_i.

Proof. Recall that when Q sends M to Q', every honest party in Q sends M to all parties in Q'. A party in Q' considers itself to have received the message M from Q if it receives M from at least 7/8 of the parties in Q. Since n quorums have successfully been formed, more than 7/8 of the parties in each quorum are honest. In particular, this is true for Q. Thus, at least 7/8 of the members of Q send M to each member of Q'. Since the adversary must eventually deliver all the messages that have been sent, albeit with arbitrary delays, it follows that eventually each honest party in Q' receives M from at least 7/8 of the members of Q. □ □

We now proceed to the proof of the Input Commitment stage. The ideal functionality, F_{Input}, is given in Protocol 5. This functionality creates a set S containing the index of the parties whose inputs have been accepted (as defined in Step 1 of Protocol 5) by the protocol to be used for the computation. If a party's input is not in S, then the functionality sets this input to the default value. Next, the functionality sends each masked input $\widehat{x_i}$ to quorum Q_i and secret-shares the mask r_i in Q_i. In Lemma 2.5.3, we show the Input Commitment stage in Protocol 1 correctly implements this functionality. Thus, the parties in Q_i eventually either have received consistent VSS-shares of x_i and have agreed on $\widehat{x_i} = x_i + r_i$ as well as on i being in S or they have agreed that $i \notin S$ and have set these values to the predefined value and r_v and all its shares to 0. We say that a quorum has come to agreement on X if all honest parties in the quorum agree on X.

Lemma 2.5.3. The Input Commitment stage of Protocol 1 is UC-secure.

Proof. First, we show that corrupted parties cannot do anything but choose their input as they wish; thus, the Input Commitment stage correctly computes F_{Input}. This means that all honest parties receive the $\langle \mathsf{Done} \rangle$ message. Moreover, there exists a set S such that for every $i \in [n]$, the following statements hold:

1. All parties in Q_i eventually agree whether $i \in S$ or not.

2. At least $n-t$ input quorums agree that their corresponding party's index is in S.

3. All parties in Q_i agree that party $i \in S$ if and only if they collectively hold enough shares to reconstruct P_i's input. If all parties in Q_i agree that $i \in S$, then party P_i's input will be used in the computation. Otherwise, the default value will be used instead.

First, since there are $n-t$ honest parties, at least $n-t$ valid inputs are eventually sent to Thresh-Count. Based on Theorem 2.6.1, all parties will be notified when $n-t$ inputs are received.

Each party in Q_i has set its flag bit to either 1 or 0 depending on whether it has received a valid input share from P_i. Let $q = |Q_i|$. Upon receiving the ⟨Done⟩ message, the parties in Q_i run the third step of Wait-For-Inputs to decide whether at least $\frac{5q}{8}$ of them have set their flag bit to 1. If they have, they assume $i \in S$.

If $i \in S$, then at least $\frac{7q}{8}$ of the parties in Q_i have received input shares from P_i before they received the ⟨Done⟩ message. Of these, more than $\frac{3q}{4}$ parties in Q_i are honest and have set their flag bit to 1. Since CMPC in Line 3 starts even if as many as $q/8$ inputs are missing, the parties in Q_i will correctly decide that at least $\frac{5q}{8}$ flag bits among them are set to 1. Thus, the parties in Q_i all agree that $i \in S$. If $i \notin S$, then CMPC in Line 3 has determined that less than $\frac{5q}{8}$ flag bits are set to 1. Since Q_i contains less than $q/8$ corrupted parties, more than $q/2$ parties set their flags to 0 and the parties in Q_i all agree that $i \notin S$. As a result, at least $n-t$ input quorums agree that their corresponding inputs are in S, and hence $|S| \geq n-t$.

We prove the t-security of the Input Commitment stage in the ($F_{\text{VSS-Share}}$, $F_{\text{VSS-Reconst}}$,F_{CMPC})-hybrid model which is similar to the Input Commitment stage of Protocol 1 except that every call to its subprotocols is replaced with a call to their corresponding functionality. We define the corresponding simulator $\mathcal{S}_{\text{Input}}$ in Protocol 6.

Let V_1 denote the view of the adversary from the hybrid execution, and V_2 be its view from the simulation. The inputs to Thresh-Count and Line 3 of Protocol 4 are completely independent of the inputs of Protocol 1. Thus, V_1 contains only the masked inputs, $\widehat{x_i}$'s, and at most $1/8$ fraction of the shares for each random mask, r_i's. The masked inputs convey no information about the inputs. Moreover, a $1/8$ fraction of the shares are not enough to reconstruct the random number. Since V_2 contains all random elements, the adversary cannot distinguish V_1 from V_2. Since our simulator is straight-line and black-box, it follows from Theorem 2.5.1 that the Input Commitment stage is UC-secure. □ □

Protocol 6 $\mathcal{S}_{\text{Input}}$

For every $i \in [n]$, party P_i holds an input $x_i \in \mathbb{F}$. Associated with this input, we consider a quorum Q_i. Let I_i denote the set of corrupted parties in Q_i, and let I denote the set of all corrupted parties among $P_1,...,P_n$.

Inputs. $\{r_i\}_{i\in[n]}$, and $\{\widehat{x}_i\}_{i\in[n]}$ from parties in I (set of all corrupted parties).

Simulation:

1. For every $i \in [n]$, if $P_i \in I$, send $x_i + r_i$ to all parties in Q_i, and run $F_{\text{VSS-Share}}$ to secret-share r_i in Q_i.
2. If $P_i \notin I$,
   a) Choose r_i and x_i uniformly at random from $\mathbb{F}$ and $\widehat{x}_i \leftarrow x_i + r_i$.
   b) Send $\widehat{x}_i$ to all parties in Q_i.
   c) Run $F_{\text{VSS-Share}}$ to secret-share r_i in Q_i.
   d) For every party in I_i, add his share of r_i and $\widehat{x}_i$ to his view.
3. For every party in Q_i, run Wait-For-Inputs to wait for at least $n - t$ inputs.
   a) Run $F_{\text{Thresh-Count}}$ with flag b_i initially set to zero to count the number of received inputs.
   b) If x_i and r_i are valid and consistent (based on the broadcast protocol and the verification stage of VSS-Reconst respectively), raise an event to set $b_i \leftarrow 1$ in $F_{\text{Thresh-Count}}$.
   c) Upon receiving ⟨Done⟩ from the parent quorum, run CMPC using b_i as your input to set $d \leftarrow$ True if a 5/8-fraction of the parties in Q have their b_i's set to one. Otherwise, $d \leftarrow$ False.
   d) If d is set to False, then set $y \leftarrow$ Default and $r_i \leftarrow 0$.

Protocol 7 $F_{\text{Gen-Rand}}$

Goal. For a gate $u \in C$, generate a random value $r \in \mathbb{F}$ and VSS-share it among parties $P_1,...,P_N$ in the quorum associated with u.

Functionality:

1. Receive inputs $\rho_1,...,\rho_N \in \mathbb{F}$ from $P_1,...,P_N$ respectively. For every $i \in [N]$, if P_i does not send an input, then define $\rho_i = 0$.
2. Calculate $r = \sum_{i=1}^{N} \rho_i$ and invoke $F_{\text{VSS-Share}}$ to send a share r_i of r to P_i.

2.5.4 Security of Circuit Evaluation

We first prove the security of Gen-Rand. The ideal functionality $F_{\text{Gen-Rand}}$ is given in Protocol 7. At least $7n/8$ of the inputs $\rho_1,...,\rho_N$ are sent by honest parties and thus are chosen uniformly and independently at random from $\mathbb{F}$. Hence, $r = \sum_{i=1}^{N} \rho_i$ is also a uniform and independent random element of $\mathbb{F}$. This is because the sum of elements of $\mathbb{F}$ is uniformly random if at least one of them is uniformly random.

Lemma 2.5.4. The protocol Gen-Rand is UC-secure.

Proof. We prove the t-security of Gen-Rand in the $F_{\text{VSS-Share}}$-hybrid model which is similar to

Protocol 8 $\mathcal{S}_{\text{Gen-Rand}}$

Inputs. For a gate $u \in C$, the inputs $\{\rho_j\}_{P_j \in I_u}$ of the corrupted parties $P_1,...,P_N$ in the quorum associated with u.

Simulation:

1. For every $P_i \in (Q_u - I_u)$ (*i.e.*, for every honest party P_i), call $F_{\text{VSS-Share}}$ with dummy input 0. Let $s_1^i,...,s_N^i$ denote the outputs.
2. For every $P_j \in I_u$,
   a) Run $F_{\text{VSS-Share}}$ with input ρ_j. Let $\rho_1^j,...,\rho_N^j$ denote the outputs. For every $k \in [N]$, add ρ_j^k to the view of P_j.
   b) Compute $r_j = \sum_{k=1}^{N} \rho_j^k$ and add r_j to the view of P_j.

Protocol 9 $F_{\text{Circuit-Eval}}$

Goal. For each gate $u \in C$ with children $v,w \in C$, $3N$ parties $P_1,...,P_{3N}$ provide inputs to the functionality to allow it evaluate the functionality of u denoted by f_u.

Functionality:

1. For every $i \in [N]$, receive ρ_i from P_i, $\widehat{y}_v$ and $r_v^{(i)}$ from P_{i+N}, and $\widehat{y}_w$ and $r_w^{(i)}$ from P_{i+2N} respectively.
2. Run $F_{\text{Gen-Rand}}$ with inputs $\rho_1,...,\rho_N$ to generate $r_u^{(1)},...,r_u^{(N)}$.
3. Run F_{CMPC} to locally compute the following functionality:
   a) $r_u \leftarrow F_{\text{VSS-Reconst}}$ over $r_u^{(1)},...,r_u^{(N)}$.
   b) $r_v \leftarrow F_{\text{VSS-Reconst}}$ over $r_v^{(1)},...,r_v^{(N)}$.
   c) $r_w \leftarrow F_{\text{VSS-Reconst}}$ over $r_w^{(1)},...,r_w^{(N)}$.
   d) $y_1 \leftarrow \widehat{y}_v - r_v$
   e) $y_2 \leftarrow \widehat{y}_w - r_w$
   f) $\widehat{y}_u \leftarrow f_u(y_1,y_2) + r_u$

Protocol 3 except that every call to VSS-Share is replaced with a call to the ideal functionality $F_{\text{VSS-Share}}$. The corresponding simulator $\mathcal{S}_{\text{Gen-Rand}}$ is given in Protocol 8.

The views of the corrupted parties in the hybrid execution and the simulation are indistinguishable because the only difference between the two views is that $\mathcal{S}_{\text{Gen-Rand}}$ generates the shares from dummy input 0 instead of actual inputs. Since $F_{\text{VSS-Share}}$ generates uniform and independent random shares from any input, the two views are identically distributed. Since our simulator is straight-line and black-box, Gen-Rand is UC-secure. □ □

We now proceed to the security proof of Circuit-Eval. The ideal functionality $F_{\text{Circuit-Eval}}$ is given in Protocol 9.

Lemma 2.5.5. The protocol Circuit-Eval is UC-secure.

Protocol 10 $\mathcal{S}_{\text{Circuit-Eval}}$

For every gate $u \in C$ with children $v,w \in C$, consider three groups of parties Q_u,Q_v, and Q_w, each of whom have N parties. In each group, up to $N/8$ parties are corrupted.

Inputs. $\{\rho_i\}_{P_i \in I_u}$,$\{r_u^{(i)}\}_{P_i \in (I_v \cup I_w)}$, and $\widehat{y}_v$ and $\widehat{y}_w$ from parties in $I_v \cup I_w$.

Simulation:

1. Run $F_{\text{Gen-Rand}}$ with the following inputs: ρ_i for every $P_i \in I_u$ and a dummy input for every party in $Q_u - I_u$. Let $\{r_u^{(i)}\}_{P_i \in Q_u}$ denote the outputs. For every $P_i \in I_u$, add $r_u^{(i)}$ to the view of P_i.

2. Let $Q_\triangle = Q_u \cup Q_v \cup Q_w$ and $I_\triangle = I_u \cup I_v \cup I_w$. Run F_{CMPC} to compute the functionality defined in Line 3 of $F_{\text{Circuit-Eval}}$ with the following inputs: the input of every party in $I_\triangle$ as described in $F_{\text{Circuit-Eval}}$, and a dummy input for every party in $Q_\triangle - I_\triangle$. Let $\widehat{y}_u$ denote the output. For every party in $I_\triangle$, add $\widehat{y}_u$ to the view of the party.

Proof. We first show that for each gate $u \in C$, $F_{\text{Circuit-Eval}}$ correctly computes $\widehat{y}_u = y_u + r_u$. Based on F_{Input} and $F_{\text{Gen-Rand}}$, for each gate $u \in C$, the inputs of the honest parties in Q_u are enough to reconstruct r_u. If u is an input gate not included in the computation from the Input Commitment stage, then r_u and its shares are 0. Thus, all three values of r_u, r_v, and r_w can be correctly reconstructed by the functionality since $F_{\text{VSS-Reconst}}$ can tolerate up to a $1/4$ fraction of the inputs being invalid.

We prove $\widehat{y}_u = y_u + r_u$ by induction on the height of u, where y_u is the correct output of the gate u. The base case is correct because based on the correctness of F_{Input}, for each input gate v', we have $\widehat{y}_{v'} = y_{v'} + r_{v'}$ and $r_{v'}$ can correctly be reconstructed from the inputs received from honest parties in $Q_{v'}$. Suppose that for all gates u' whose height is less than the height of u, the functionality can compute $\widehat{y}_{u'} = y_{u'} + r_{u'}$ and $r_{u'}$. This induction hypothesis is valid for v and w.

We now describe the induction step. In the computation of u, the functionality runs F_{CMPC}. We now argue based on the definition of the function computed by F_{CMPC} that the output of F_{CMPC} is $\widehat{y}_u = r_u + y_u$. By the induction hypothesis, the functionality can reconstruct correct r_v and r_w and consequently it can correctly find y_v and y_w even if a $1/8$ fraction of the inputs are missing. It is because the majority of the parties in Q_v and Q_w hold correct values of $\widehat{y}_v$ and $\widehat{y}_w$. Thus, the functionality can correctly compute $f_u(y_v, y_w) + r_u$.

We now prove the t-security of Circuit-Eval in the $(F_{\text{Gen-Rand}}, F_{\text{CMPC}})$-hybrid model which is similar to Protocol 2 except that every call to CMPC and Gen-Rand is replaced with a call to F_{CMPC} and $F_{\text{Gen-Rand}}$ respectively. The corresponding simulator $\mathcal{S}_{\text{Circuit-Eval}}$ is given in Protocol 10.

We now show that the views of the corrupted parties in the hybrid execution and the simulation are indistinguishable. After the evaluation of u, the following information will be added to the view of every corrupted party $P_i \in I_\triangle$: $\widehat{y}_u$ and $\{r_u^{(j)}\}_{P_j \in I_u}$. Recall that $\widehat{y}_u$ is the output of F_{CMPC} during the computation of u which is equal to $y_u + r_u$, and r_u is a uniformly random element of $\mathbb{F}$ based on

$F_{\text{Gen-Rand}}$, independent of all other randomness in the algorithm.

First, if a corrupted party P_i is not in any of the quorums associated with u,v, and w, then no additional information will be added to its view during the computation of u; thus, its view will be identically distributed in the hybrid execution and the simulation.

Second, a corrupted party $P_i \in I_\triangle$ may add a share r_u as well as shares of the individual random elements whose sum is r_u to its view in the computation of $F_{\text{Gen-Rand}}$. Also, it adds $y_u + r_u$ to its view. However, P_i cannot learn any additional information about the shares of r_u (and thus about r_u) based on F_{CMPC} and $F_{\text{Gen-Rand}}$. In other words, the parties in $I_\triangle$ are unable to directly determine r_u, since the only relevant inputs are the shares of r_u, and they do not have enough of those since they have fewer than half of them.

These parties also do not have enough shares of shares of r_u to reconstruct it. However, they add to their view shares of each of the other shares of r_u multiple times: once during the input stage of F_{CMPC} in which u is involved, and once during the computation of the parent of u. Each time, they do not get enough shares of shares r_u to reconstruct any shares of r_u. But, can they combine the shares of shares from different runs for the same secret to gain some information? Since fresh and independent randomness was used by the dealers creating these shares on each run, the shares from each run are independent of the other runs, and so they do not collectively give any more information than each of the runs give separately. Since each run does not give the parties in $I_\triangle$ enough shares to reconstruct anything, it follows that they do not learn any information about r_u.

Second, parties in $I_\triangle$ add shares of shares for r_v and r_w to their views. However, with a similar argument as r_u, they cannot reconstruct r_v and r_w as well even if these parties participate in one or more of the instances of F_{CMPC} which involve v or w: the computation of v or w themselves or the computations of u as their parents.

Moreover, $\widehat{y}_u$ is also a random element in the field since r_u is uniformly random and $\widehat{y}_u = y_u + r_u$. Thus, $\widehat{y}_u$ holds no information about y_u, and the corrupted parties cannot learn any information about y_u except what is implicit in his input and the circuit output. This means that the corrupted parties cannot distinguish if they are participating in a run of the hybrid model or the simulation. Finally, since $\mathcal{S}_{\text{Circuit-Eval}}$ is straight-line and black-box, Circuit-Eval is UC-secure. □ □

2.5.5 Security of Output Stages

The ideal functionality for the Output Reconstruction and the Output Propagation stages of Protocol 1 are given in Protocol 11.

Lemma 2.5.6. The Output Reconstruction and Output Propagation stages of Protocol 1 are UC-

Protocol 11 F_{Output}

Goal. The functionality guarantees the output is reconstructed correctly and it is learned by all honest parties.

Functionality:

1. Run $F_{\text{VSS-Reconst}}$ to reconstruct the output.
2. Send the output to all the parties.

Protocol 12 $\mathcal{S}_{\text{Output}}$

Inputs. For the output gate z and the corresponding quorum Q_z, the inputs of the simulator are $\{r_z^{(i)}\}_{P_i \in I_z}$, and $\widehat{y}_z$ from parties in I_z.

Simulation:

1. Run $F_{\text{VSS-Reconst}}$ with inputs $\{r_z^{(i)}\}_{P_i \in I_z}$ and dummy inputs for honest parties. Add the output to the view of parties in I_z.
2. For every $i \in \{2,...,n\}$, parties in Q_i perform the following steps:
   a) Receive y from $Q_{\lfloor i/2 \rfloor}$ and add it to the view of every parties in $I_{\lfloor i/2 \rfloor}$.
   b) Send y to all parties in Q_{2i} and Q_{2i+1}.

secure.

Proof. We first show that the two stages correctly compute F_{Output}. Let z be the output gate of C. By Lemma 2.5.5, all parties in the output quorum Q_z eventually agree on $y_z + r_z$ and hold shares of r_z. In the Output Reconstruction stage, these parties run the VSS-Reconst. Since at least a 7/8 fraction of them are honest, they correctly reconstruct r_z. Since all honest parties in Q_z know $y_z + r_z$ and subtract from it the reconstructed r_z, they all eventually learn y_z. Thus, all parties in Q_z eventually learn y_z.

We now show by induction that all honest parties eventually learn y_z. Since Q_1 is assigned to the output gate, it provides a base case. For $i > 1$, consider the parties in Q_i, and for all $j < i$ assume the correct output is learned by all parties in Q_j. During the Output Propagation stage, the parties in Q_i receive putative values for the output from the parties at $Q_{\lfloor i/2 \rfloor}$. Since $Q_{\lfloor i/2 \rfloor}$ is good, and by induction hypothesis all honest parties in it have learned the correct output, it follows that all honest parties in quorum $Q_{\lfloor i/2 \rfloor}$ send the same message which is the correct output. By Lemma 2.5.2, all honest parties in Q_i eventually learn the correct output. By induction, all the parties learn the correct value.

We now prove the t-security of the output stages in the $F_{\text{VSS-Reconst}}$-hybrid model. The corresponding simulator $\mathcal{S}_{\text{Output}}$ is given in Protocol 12. The views of the corrupted parties in the hybrid execution and the simulation are indistinguishable since the only message that is added to the view

of the adversary is the output. Based on the security definition of MPC, the adversary is allowed to learn the output. □ □

2.5.6 Security of Protocol 1

We now show that our main protocol is UC-secure.

Lemma 2.5.7. Protocol 1 is UC-secure.

Proof. Canetti [Can95] proves the t-security of VSS-Share, VSS-Reconst, and CMPC using straight-line black-box simulators. So, based on Theorem 2.5.1, these protocols are UC-secure. Moreover, Lindell *et al.* [LLR06] show that any Byzantine agreement protocol in the standard model (such as the protocol of [CR93]) is UC-secure. Hence, the Byzantine agreement of [FM88] is also UC-secure.

Protocol 1 is t-secure since in lemmas 2.5.3, 2.5.5, and 2.5.6 we showed that all stages of the Protocol 1 are t-secure. Based on Theorem 2.5.1, since we have proved the t-security of Protocol 1 using a straight-line black-box simulator, the protocol is also UC-secure. □ □

2.5.7 Cost Analysis

We now analyze the resource costs of Protocol 1.

Lemma 2.5.8. During the Input stage, each quorum sends at most $O(\log n)$ messages.

Proof. For the input stage, each quorum is mapped to at most one of the input gates and hence one of the nodes in the count tree. Thus, from Theorem 2.6.1 it follows that the total number of messages sent by each quorum is $O(\log n)$. Since each quorum has $\log n$ parties, an additional polylog(n) messages are sent by each quorum during VSS-Share and VSS-Reconst to check whether the input is correctly secret-shared. □ □

Lemma 2.5.9. If all honest parties follow Protocol 1, then with high probability, each party sends at most $\tilde{O}(m/n + \sqrt{n})$ messages.

Proof. By Theorem 2.7.1, we need to send $\tilde{O}(\sqrt{n})$ messages per party to build the quorums. Subsequently, each party must send messages for each quorum in which it is a member. Recall that each party is in $\Theta(\log n)$ quorums.

By Lemma 2.5.8, each quorum sends $\tilde{O}(\log(n))$ messages during Input stage. Recall that each quorum is mapped to $\Theta(\frac{m+n}{n})$ nodes of C. A quorum runs Gen-Rand and the gate evaluation

step of Circuit-Eval once per node it is mapped to in C. Since each gate has in-degree two and out-degree at most two, a quorum runs CMPC at most three times for every node it is mapped to in C. Also, at most polylog(n) messages are sent per party per instance of CMPC, Gen-Rand, and gate evaluation. Finally, each quorum sends $O(\log n)$ messages in the dissemination of the output. Thus, each quorum sends polylog(n) messages per node it represents. It follows that each party sends $\tilde{O}(m/n + \sqrt{n})$ messages. □ □

Lemma 2.5.10. If all honest parties follow Protocol 1, with high probability, the total latency is $O(d\,\text{polylog}(n))$ where d is depth of the circuit the protocol computes.

Proof. Based on Theorem 2.7.1, the latency for creating quorums is polylog(n). Based on Theorem 2.6.1, the latency for the Thresh-Count algorithm is $O(\log n)$ which implies that the Input Commitment stage also has polylog(n) latency.

In the computation of the circuit, to evaluate the gate g in the upper level of the circuit, first its input gates in lower level of the circuit must be evaluated. This implies that the evaluation of the circuit is level by level and the latency for evaluating the circuit is $O(d)$ times the latency of CMPC over $\log n$ parties. □ □

2.6 Asynchronous Threshold Counting

In this section, we present an asynchronous Monte Carlo algorithm called Thresh-Count which solves the threshold counting problem and provides the following theorem proved in Section 2.6.4.

Our threshold counting algorithm runs in a setting with n honest parties in a fully-connected network with private and authenticated channels and asynchronous communication. In our asynchronous MPC protocol presented in Section 2.4, we run Thresh-Count among a set of quorums, where each quorum represents an honest party.

Theorem 2.6.1. The algorithm Thresh-Count solves the threshold counting problem with high probability, while ensuring:

1. Each party sends at most $O(\log n)$ messages of constant size,
2. Each party receives at most $O(\log n)$ messages,
3. Each party performs $O(\log n)$ computations,
4. Total latency is $O(\log n)$.

Recall that in the threshold counting problem there are n honest parties in an asynchronous communication network with private channels. Each party has an input flag which is initially 0. At least τ of the parties' bits will eventually be set to 1 based on an external event. When this happens, we say the threshold is reached. The goal is for each of the parties to terminate at some time after the threshold is reached.

Although in our application τ is linear in n, we address the more general case, where $\tau = O(n)$. Our algorithm depends on prior knowledge of τ. As specified in Theorem 2.6.1, each party running the algorithm sends and receives $O(\log n)$ messages of constant size and performs $O(\log n)$ computations; moreover the total latency is $O(\log n)$.

For ease of presentation, we first describe an algorithm which works when $\tau = \Theta(n)$, in particular, when τ is at least $n/2$. We then indicate why this fails when τ is smaller, and show how to modify it so that it works for all τ. The formal algorithm is shown as Protocol 13.

Consider a complete binary tree where each party sends its input to a unique leaf node when it is set to 1. Then, for every node v, each child of v sends v a message giving the number of inputs it has received so far and it sends a new message every time this number changes. The problem with this approach is that it is not load-balanced: each node at depth i has $n/2^i$ descendants in the tree, and therefore, in the worst case, sends and receives $n/2^i$ messages. Thus, a child of the root sends $n/2$ messages to the root and receives the same number of messages from its children.

To solve the load-balancing problem, we use a randomized approach which ensures with high probability that each leaf of the data structure receives at least $7 \log n$ messages and does not communicate with its parent until it has done so. Subsequent messages it receives are not forwarded to its parent but rather to other randomly chosen leaves to ensure a close to uniform distribution of the messages.

Our algorithm consists of up and down stages. For the up stage the parties are arranged in a predetermined tree data structure, which we call the *count tree.* The count tree consists of a root node with $O(\log n)$ children, each of which is itself the root of a complete binary tree; these subtrees have varying depths as depicted in Figure 2.4. In the up stage, parties in the trees count the number of 1-inputs, *i.e.* , the number of parties' inputs that are set to **1**. The root then eventually decides when the threshold is reached. In the down stage, the root notifies all the parties of this event via a complete binary tree of depth $\log n$. Note that the tree used in the down stage has the same root as the count tree.

Let $D = \lceil \log \frac{\tau}{14 \log n} \rceil$. Note that $D = O(\log n)$. The root of the count tree has degree D. Each of the D children of the root is itself the root of a complete binary subtree, which we will call a *collection subtree*. For $1 \leq j \leq D$, the jth collection subtree has depth $D + 1 - j$. Party 1 is assigned

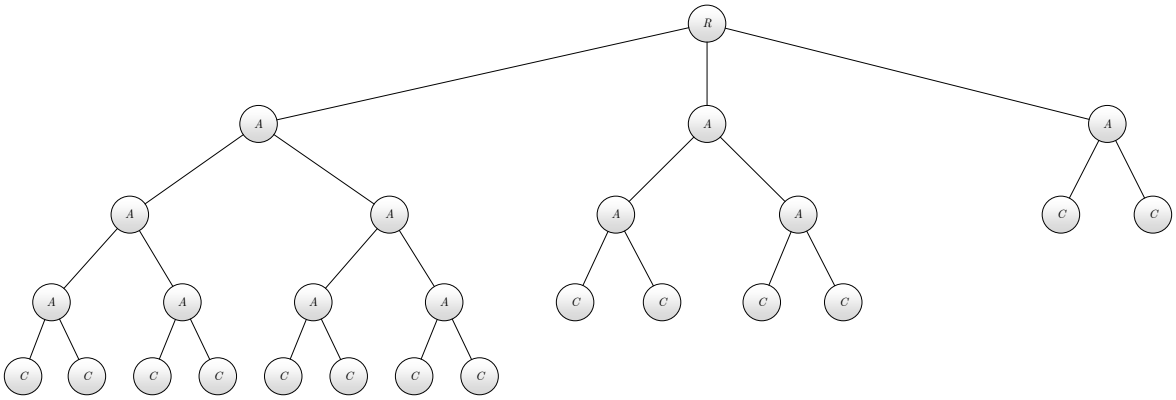

Figure 2.4: The count tree for $n = 2048$ and $\tau = 1232$. $D = \lceil \log \frac{1232}{14 \times 11} \rceil = 3$. The node marked R is the root, nodes marked A are adding nodes, and nodes marked C are collection nodes.

to the root and parties 2 to $D+1$, are assigned to its children, *i.e.* , the roots of the collection subtrees, with party $j+1$ being assigned to the jth child. The remaining nodes of the collection trees are assigned parties in order, starting with $D+2$, left to right and top to bottom. One can easily see that the entire data structure has fewer than n nodes, (in fact it has fewer than $\frac{\tau}{3 \log n}$ nodes) so some parties will not be assigned to any node.

The leaves of each collection subtree are *collection nodes*, while the internal nodes of each collection tree are *adding nodes*.

2.6.1 Up Stage

When a party's input is set to **1**, it sends a ⟨Flag⟩ message, which we will sometimes simply refer to as a flag, to a uniformly random collection node from the first collection subtree. Intuitively, we want the flags to be distributed as evenly as possible among the collection nodes. The parameters of the algorithm are set up so that with high probability each collection node receives at least $7 \log n$ ⟨Flag⟩ messages.

Each collection node in the j-th collection tree waits until it has received $7 \log n$ flags. It then sends its parent a ⟨Count⟩ message. For each additional flag received, up to $14 \log n$, it chooses a uniformly random collection node in the $(j+1)$-st collection subtree and forwards a flag to it. If $j = D$, then it forwards these $14 \log n$ flags directly to the root. Subsequent flags are ignored. Again, we use the randomness to ensure a close to even distribution of flags with high probability.

Each adding node waits until it has received a ⟨Count⟩ message from each of its children. Then, it sends a ⟨Count⟩ message to its parent. We note that, with high probability, each adding node

sends exactly one message during the algorithm. The parameters of the algorithm are arranged so that all the ⟨Count⟩ messages that are sent in the the jth collection subtree together account for $\tau/2^j$ of the 1-inputs. Thus, all the ⟨Count⟩ messages in all the collection subtrees together account for $\tau\left(1-\frac{1}{2^D}\right)$ of the 1-inputs. At least $\frac{\tau}{2^D}$ 1-inputs remain unaccounted for. These 1-inputs and up to $O(\log n)$ more are collected as flags at the root.

2.6.2 Down Stage

When party 1, at the root, has accounted for at least τ 1-inputs, it starts the down stage by sending the ⟨Done⟩ message to parties 2 and 3. For $j > 1$, when party j receives the ⟨Done⟩ message, it forwards this message to parties $2j$ and $2j+1$. Thus, eventually the ⟨Done⟩ message reaches all the parties, who then learn that the threshold has been met.

Note that all three types of messages sent in this protocol, ⟨Flag⟩, ⟨Count⟩ and ⟨Done⟩, are notifications only; they do not contain any numerical value. Since 2 bits are sufficient to distinguish three different kinds of messages, all the messages sent in this protocol are 2-bit strings. Note that we distinguish between flags and ⟨Count⟩ messages since the root receives both kinds. However it is the only node for which this is a problem. We could add another node, as the $(D+1)$st child of the root, (equivalently as a collection subtree of depth 0,) which waits for $14\log n$ messages, and sends a ⟨Count⟩ message to the root. In so doing, we could eliminate the need to explicitly distinguish ⟨Flag⟩ and ⟨Count⟩ message, since they would be automatically distinguished by the role of the receiving node. Thus, we could actually reduce all message lengths to a single bit.

2.6.3 Handling Sublinear Thresholds

Now, we consider the case where $\tau = o(n)$. It is easy to see that the worst load in terms of the number of received messages is when all n inputs are 1. In this case, a collection node in the first collection subtree receives, on average, $14(n/\tau)\log n$ flags. When $\tau = \Theta(n)$, this is still $O(\log n)$, but when $\tau = o(n)$ this is $\omega(\log n)$. Before we describe how to fix this, we note that the problem exists *only* in the leaves of the *first* collection subtree. Subsequent collection nodes receive only $O(\log n)$ flags, because each node only forwards up to $14\log n$ flags.

For the sake of having a definite cutoff and tractable constants, we will apply the following fix whenever $\tau < n/2$. Below each collection node in the first collection tree, we put in a *filter*, which is a complete binary tree of depth $\log n - 2 - D$ with $\frac{7n\log n}{2\tau}$ leaves. This is equivalent to extending the first collection tree to depth $\log n - 2$ so that it has $n/4$ leaves. The collection nodes will remain at depth D though. See Figure 2.5.

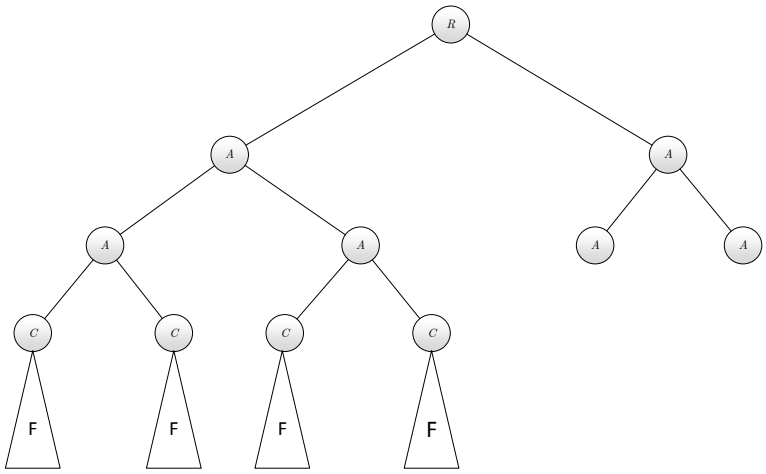

Figure 2.5: The count tree for $n = 2048$ and $\tau = 616$. $D = \lceil \log \frac{616}{14 \times 11} \rceil = 2$. The node marked R is the root, nodes marked A are adding nodes and nodes marked C are collection nodes. The filters, marked F, are complete binary trees of depth 7, with 128 leaves each, for a total of 512 filter leaves.

When a party's input is set to 1, it selects a random collection node in the first collection tree, but rather than sending a flag directly to it, it sends the flag to a random leaf of the collection node's filter. The nodes in the filter simply forward any flags they receive, up to $21 \log n$, to their parent in the filter. Subsequent flags are ignored. Clearly, this means that the collection node at the root of the filter cannot receive more than $42 \log n$ flags, which solves the load problem. Moreover, we have not simply transfered the problem to the leaves of the filter. Since there are so many more of them, each one actually receives fewer flags on average and the parameters are adjusted to make their maximum load $O(\log n)$ with high probability. As we will also see in the analysis, these filters do not filter out too many flags; when there are only τ 1-inputs among the parties, with high probability all the flags get through.

2.6.4 Proof of Theorem 2.6.1

In this section, we prove the correctness and resource costs of Protocol 13. The process of each party independently selecting a random collection node to notify after its input has been set to **1** can be modeled as a balls and bins problem and hence be approximated by the Poisson distribution.

Protocol 13 Thresh-Count

Goal. n is the number of parties, τ is the threshold, b is a flag bit initially set to zero which may be set to one by an external event throughout the protocol and $D = \lceil \log(\frac{\tau}{14 \log n}) \rceil$. The algorithm notifies all the parties upon receiving τ flag bits set to one.

1. **Setup.** No messages sent in this stage:

   a) Build the count tree and set party 1 as the root:
   For $1 \leq j \leq D$, party $j+1$ is a child of the root (and the root of the jth collection subtree with depth $D+1-j$). Starting with party $D+2$, the remainder of the nodes are assigned to parties, left to right and top to bottom. If $\tau < n/2$ the remaining parties are assigned to filters, left to right and top to bottom.

   b) Let $sum = 0$ for the root.

2. **Up Stage.**

   a) Upon $b = 1$, choose a uniformly random collection node v from collection subtree 1,

   - If $\tau > n/2$, send a ⟨Flag⟩ to v.
   - Otherwise, choose a uniformly random leaf in v's filter and send a ⟨Flag⟩ to it.

   b) Upon receiving a ⟨Flag⟩, if previously forwarded fewer than $21 \log n$ flags, forward the flag to parent. Otherwise, ignore it.

   c) Perform the following steps to collect nodes in the collection subtree j:

   - Upon receiving $7 \log n$ ⟨Flag⟩s, send parent a ⟨Count⟩ message.
   - Upon subsequently receiving a ⟨Flag⟩, if $j < D$, send it to a uniformly random collection node in collection subtree $j+1$. If $j = D$, then send it directly to the root. Do this for up to $14 \log n$ flags. Then, ignore all subsequent ⟨Flag⟩ messages.

   d) Upon receiving ⟨Count⟩ from both children, send ⟨Count⟩ to the parent.

   e) If $sum < \tau$,

   - Upon receiving a ⟨Count⟩ from party $j+1$, set $sum \leftarrow sum + \tau/2^j$.
   - Upon receiving a ⟨Flag⟩, $sum \leftarrow sum + 1$.

3. **Down Stage.** If $sum \geq \tau$,

   a) Party 1 (the root): Send ⟨Done⟩ to parties 2 and 3, and then terminate.

   b) Party j for $j > 1$: Upon receiving ⟨Done⟩ from party $\lfloor j/2 \rfloor$, forward it to parties $2j$ and $2j+1$ (if they exist), and then terminate.

2.6.4.1 Preliminaries

We first recall the following Chernoff bound for a Poisson random variable from Mitzenmacher and Upfal [MU05].

Theorem 2.6.2 (Theorem 5.4 of [MU05])**.** Let $Y \sim \text{Poisson}(\mu)$. Then,

1. for $x < \mu$, $\text{Prob}(Y \leq x) \leq e^{-\mu}(e\mu/x)^x$, and
2. for $x > \mu$, $\text{Prob}(Y \geq x) \leq e^{-\mu}(e\mu/x)^x$.

Lemma 2.6.3. Assume αk balls are thrown independently and uniformly at random into k bins. Let E_1 denote the event that the minimum load is less than $\alpha/2$, and let E_2 denote the event that the maximum load exceeds $3\alpha/2$. Then,

$$\text{Prob}(E_1) \leq ek\sqrt{\alpha k}\left(\frac{2}{e}\right)^{\alpha/2} \tag{2.1}$$

and

$$\text{Prob}(E_2) \leq ek\sqrt{\alpha k}\left(\frac{8e}{27}\right)^{\alpha/2}. \tag{2.2}$$

Proof. For $1 \leq i \leq k$, let X_i denote the number of balls in the ith bin, and let $Y_i \sim \text{Poisson}(\alpha)$ be an independent Poisson random variable with mean α. It is well known that the distribution of each X_i is close to that of Y_i, and moreover that the *joint* distribution of the X_i's is well approximated by the joint (*i.e.* , product) distribution of the Y_i's (see Chapter 5 in Mitzenmacher and Upfal [MU05]). Indeed, Corollary 5.11 from [MU05] states that for any event E that is monotone in the number of balls, if E occurs with probability at most p in the Poisson approximation, then E occurs with probability at most $2p$ in the exact case. Since maximum and minimum load are both clearly monotone increasing in the number of balls, applying this corollary we have:

$$\begin{aligned}
\text{Prob}(E_1) &= \text{Prob}\left(\exists i \text{ s.t. } X_i \leq \alpha/2\right) \\
&\leq 2\,\text{Prob}\left(\exists i \text{ s.t. } Y_i \leq \alpha/2\right) \\
&\leq 2\sum_{i=1}^{k} \text{Prob}\left(Y_i \leq \frac{\alpha}{2}\right) \\
&\leq 2k\left(\frac{2}{e}\right)^{\alpha/2},
\end{aligned}$$

where the last inequality follows from Theorem 2.6.2 with $\mu = \alpha$ and $x = \alpha/2$. Similarly,

$$\begin{aligned}
\text{Prob}(E_2) &= \text{Prob}\left(\exists i \text{ s.t. } X_i > 3\alpha/2\right) \\
&\leq 2\,\text{Prob}\left(\exists i \text{ s.t. } Y_i > 3\alpha/2\right) \\
&\leq 2\sum_{i=1}^{k} \text{Prob}\left(Y_i \geq \frac{3\alpha}{2}\right) \\
&\leq 2k\left(\frac{8\text{e}}{27}\right)^{\alpha/2},
\end{aligned}$$

where the last inequality follows from Theorem 2.6.2 with $\mu = \alpha$ and $x = 3\alpha/2$. □

2.6.4.2 Protocol Analysis

Let σ be the number of 1-inputs. We know that $\tau \leq \sigma \leq n$. Let $s = \sigma/\tau$. For simplicity of the analysis, we will assume that the first τ flags to be sent are *marked* while the remaining $\sigma - \tau$ are *unmarked*. As we track the progress of the flags through our data structure, we pay particular attention to the marked flags. Due to asynchrony, the marked flags need not be the first τ to arrive at their destinations.

Lemma 2.6.4. Suppose $\tau \geq n/2$. In the Thresh-Count algorithm, with probability at least $1 - \frac{1}{7n\log n}$, the first collection subtree satisfies all of the following:

1. Each collection node receives between $7s\log n$ and $21s\log n$ flags.
2. The ⟨Count⟩ messages generated in this tree, when they reach the root, account for $\tau/2$ 1-inputs.
3. At least $\tau/2$ and at most τ flags are forwarded to the second collection tree.

Proof. The process of sending σ ⟨Flag⟩ messages to the collection nodes in the first collection tree can be modeled as a balls and bins problem as in Lemma 2.6.3 with $\alpha = 14s\log n$ and $k = \tau/14\log n$. E_1 and E_2 are, respectively, the events that some collection node fails to receive $7s\log n$ flags and that some collection node receives more than $21s\log n$ flags. By applying the lemma, we get

$$\begin{aligned}
\text{Prob}(E_1) &\leq \frac{2\tau}{14\log n}\left(\frac{2}{\text{e}}\right)^{7s\log n} \\
&\leq \frac{2n}{14\log n}2^{-0.4426\times 7s\log n} \\
&\leq \frac{1}{7n^{2s}\log n}
\end{aligned}$$

and

$$\begin{aligned}\text{Prob}(E_2) &\le \frac{2\tau}{14\log n}\left(\frac{8\mathrm{e}}{27}\right)^{7s\log n}\\ &\le \frac{2n}{14\log n}2^{-0.3121\times 7s\log n}\\ &\le \frac{1}{7n^{1.1s}\log n}\end{aligned}$$

Thus, the probability that (a) fails is at most $\frac{1+n^{0.9s}}{7n^{2s}\log n}$.

To see (b), we note that there are $\tau/(14\log n)$ collection nodes in the first collection subtree, each of whom generates a ⟨Count⟩ message when it has received $7\log n$ flags. The flags correspond to distinct 1-inputs, and hence together they account for $\tau/2$ 1-inputs. Thus, (b) fails only if some node fails to receive at least $7\log n$ flags, which is already accounted for in the failure of (a).

To prove (c), we need to track the progress of the marked flags. Let E_1' and E_2' denote respectively, the events that some node fails to receive at least $7\log n$ marked flags and that some node receives more than $21\log n$ marked flags. Then, since there are τ marked flags, applying Lemma 2.6.3 with $\alpha = 14\log n$ and $k = \tau/14\log n$ we see that

$$\begin{aligned}\text{Prob}(E_1') &\le \frac{2\tau}{14\log n}\left(\frac{2}{\mathrm{e}}\right)^{7\log n}\\ &\le \frac{2n}{14\log n}2^{-0.4426\times 7\log n}\\ &\le \frac{1}{7n^2\log n}\end{aligned}$$

and

$$\begin{aligned}\text{Prob}(E_2') &\le \frac{2\tau}{14\log n}\left(\frac{8\mathrm{e}}{27}\right)^{7\log n}\\ &\le \frac{2n}{14\log n}2^{-0.3121\times 7\log n}\\ &\le \frac{1}{7n^{1.1}\log n}.\end{aligned}$$

Within each collection node, by transferring the marks from some marked flags to some unmarked flags, we may assume that the marked flags are the first to arrive. We can do this transfer because it does not change the distribution of marked and unmarked flags between the nodes, nor does it change the total number of marked flags across all collection nodes. The advantage of this change is that in following the algorithm, each node will first use all its marked flags before using unmarked flags.

In particular, as long as E_1' and E_2' do not occur, each node will use $7\log n$ flags to generate a ⟨Count⟩ message, after which it will be left with between 0 and $14\log n$ marked flags. Since it forwards up to $14\log n$ flags to the next collection subtree, it follows that it will forward *all* of its marked flags and possibly some unmarked flags to the next subtree. Since there are τ marked flags across all the collection nodes, and the ⟨Count⟩ messages account for $\tau/2$ of them, it follows that the remaining $\tau/2$ marked flags are forwarded. Hence, at least $\tau/2$ flags are forwarded. Moreover, since there are $\tau/(14\log n)$ nodes and each forwards at up to $14\log n$ flags, at most τ flags are forwarded, which establishes (c).

Now, let $E = E_1 \cup E_2 \cup E_1' \cup E_2'$ be the union of all the bad events we've encountered. For large enough n,

$$\begin{aligned}\text{Prob}(E) &\leq \frac{1}{7n^{2s}\log n} + \frac{1}{7n^{1.1s}\log n} + \frac{1}{7n^2\log n} \\ &\quad + \frac{1}{7n^{1.1}\log n} \\ &\leq \frac{1}{7n\log n}\end{aligned}$$

Thus, with probability at least $1 - \frac{1}{7n\log n}$, (a), (b), and (c) are all true, as desired. □ □

We will also need to prove a similar lemma when $\tau < n/2$. Note that when $\tau \geq n/2$, we have $\sigma \leq 2\tau$, or $s = \sigma/\tau \leq 2$. When $\tau < n/2$, σ may be much bigger than τ. Let $M = \min\{\sigma/\tau, 2\}$.

Lemma 2.6.5. Suppose $\tau < n/2$. In the Thresh-Count algorithm, with probability at least $1 - \frac{1}{7n\log n}$, the first collection subtree satisfies all of the following:

1. Each collection node receives between $7\log n$ and $21M\log n$ flags.
2. Each filter node receives at most $21M\log n$ flags.
3. The ⟨Count⟩ messages generated in this tree, when they reach the root, account for $\tau/2$ 1-inputs.
4. At least $\tau/2$ and at most τ flags are forwarded to the second collection tree.

Proof. When $\tau < n/2$, the flags are not sent directly to the collection nodes, but rather to leaf nodes of the filters below the collection nodes. We will say that a filter receives a flag if the flag is received by any of its leaf nodes.

We first note that each party's process of selecting a random collection node, and then selecting a random leaf in its filter, is equivalent to simply selecting a uniformly random leaf node from among

all the leaf nodes for all the filters. We've already remarked that adding the filters is equivalent to extending the first collection subtree to depth $\log n - 2$ while keeping the collection layer the same. Thus, there are $n/4$ filter leaf nodes to choose from. Using the Poisson approximation and an argument similar to the one in Lemma 2.6.3, it is easy to see that when $\sigma \leq n$ parties each independently send a flag to a uniformly random filter leaf node out of $n/4$ choices, the probability of the event E_0, that there is a leaf node that receives more than $21 \log n$ flags is less than $n^{-\log\log n}$.

Once the flags have been sent to the leaf nodes of the filters, they are forwarded up the filter from nodes to their parents, all the way to the collection node, with the only caveat that nodes do not forward more than $21 \log n$ flags. Since each node has two children, it follows that each node in the filter receives at most $42 \log n$ flags, and the same is true of the collection nodes. At the same time, viewing the process as first selecting a collection node, and then a filter leaf node below it, we see as in Lemma 2.6.4 that the probability of the event E_2, that there is a filter that receives more than $21s \log n$ flags is at most $\frac{1}{7n^{1.1s}\log n}$. Since no node in the filter can get more flags than the filter as a whole, it follows that the filter nodes and the collection nodes all receive no more than $21M \log n = \min\{21s \log n, 42 \log n\}$ flags. This shows (b) and the upper bound in (a).

To show that the collection nodes each receives at least $7 \log n$ flags with high probability, and that together the collection nodes receive at least τ flags, we will once again track the marked flags. As we have remarked previously, although the marked flags are the first τ to be sent, by asynchrony, they need not be the first τ to arrive at the filters. Thus, it need not be the case that all these marked flags are forwarded through to the collection nodes. Nevertheless, we will argue that for every marked flag that fails to be forwarded, at least one unmarked flag was forwarded instead. To see this, note that as in Lemma 2.6.4, all the filters receive between $7 \log n$ and $21 \log n$ *marked* flags, except with probability $\frac{1+n^{0.9}}{7n^2 \log n}$. Thus, each node in a filter can have at most $21 \log n$ marked flags arrive at it.

Now, suppose a filter node fails to forward one or more marked flags. It can only do this if it has previously forwarded $21 \log n$ flags, and since it can receive at most $21 \log n$ marked flags, it follows that it has already forwarded at least as many *unmarked* flags as it is choosing to ignore marked ones. Once again, by transferring marks from the marked flags that are dropped to the unmarked flags that have been sent in their place, we can ensure that except with probability $\frac{1+n^{0.9}}{7n^2 \log n}$, between $7 \log n$ and $21 \log n$ marked flags get through each filter to the corresponding collection node, and at least τ marked flags get through all the filters together, to the collection layer of the first collection subtree. This shows the lower bound in (a) and sets us up to show (c) and (d).

For (c), we will once again pretend, by transferring marks that at each node the marked flags are the first to arrive and be used. As before, we do this without altering the distribution of marked

and unmarked flags between collection nodes. Note that each newly marked flag at the collection node corresponds to a distinct 1-input, so the $7\log n$ of them used by each of $\tau/(14\log n)$ collection nodes to generate a ⟨Count⟩ message accounts for $\tau/2$ 1-inputs at the root. This leaves between 0 and $14\log n$ marked flags at each collection node which add up to $\tau/2$ of them across all the collection nodes. Since each collection node forwards up to $14\log n$ flags, all the marked flags are forwarded, so that at least $\tau/2$ flags are forwarded to the next collection subtree. Since each of $\tau/(14\log n)$ collection nodes forwards up to $14\log n$ flags, at most τ flags are forwarded to the next collection tree, proving (d).

Finally, adding up the probabilities of all the bad events we've encountered, we see that for large enough n, $\frac{1+n^{0.9}}{7n^2\log n} + \frac{1}{7n^{1.1}\log n} + n^{-\log\log n} < \frac{1}{7n\log n}$. It follows that with probability at least $1 - \frac{1}{7n\log n}$, (a), (b), (c), and (d) are all true, as desired. □ □

We are now ready to study what happens further up in the data structure. We will say that the algorithm succeeds up to level j if for all $i \le j$ the following are true:

1. All the collection nodes in the ith collection subtree receive between $7\log n$ and $42\log n$ flags.
2. The ⟨Count⟩ messages generated in the ith subtree account for $\tau/2^i$ 1-inputs at the root.
3. Between $\tau/2^i$ and $\tau/2^{i-1}$ flags are forwarded from the ith collection subtree to the $(i+1)$st collection subtree

Lemma 2.6.6. Let $j \le D$. In the Thresh-Count algorithm, with probability at least $1 - \frac{j}{7n\log n}$, the algorithm succeeds up to level j.

Proof. We proceed by induction on j. We have already established the base case $j = 1$ in Lemmas 2.6.4 and 2.6.5. Now suppose $j \ge 2$, and for an induction hypothesis we assume that the algorithm succeeds to level $j-1$ with probability at least $1 - \frac{j-1}{7n\log n}$. Let us condition on this event. This means that between $\tau/2^{j-1}$ and $\tau/2^{j-2}$ flags are forwarded to the jth collection subtree, which has $\frac{\tau}{2^{j-1}14\log n}$ collection nodes.

Thus, we can apply Lemma 2.6.3 with α between $14\log n$ and $28\log n$. The proof that conditioned on the algorithm having succeeded up to level $j-1$, it succeeds to level j, except with probability $\frac{1}{7n\log n}$, is identical to the proof of Lemma 2.6.4. By Bayes' law and the induction hypothesis, the unconditional probability that the algorithm succeeds to level j

$$\left(1 - \frac{j-1}{7n\log n}\right)\left(1 - \frac{1}{7n\log n}\right) \ge 1 - \frac{j}{7n\log n},$$

as desired. □ □

Corollary 2.6.7. With probability at least $1 - \frac{1}{7n}$, the root node successfully accounts for at least τ 1-inputs.

Proof. The last collection subtree is the one corresponding to $j = D$, and by Lemma 2.6.6, with probability at least $1 - \frac{D}{7 \log n}$ the root has accounted for $\sum_{j=1}^{D} \tau/2^j = \tau(1 - 2^{-D})$ 1-inputs, and moreover, between $\tau/2^D$ and $\tau/2^{D-1}$ flags have been forwarded directly to the root, by the collection nodes in the last collection subtree. Since no randomness is involved, the root eventually receives all of these flags. Thus, conditioned on the algorithm succeeding up to level D, the root eventually accounts for at least τ 1-inputs. Since $D < \log \tau < \log n$, the success probability is at least $1 - \frac{1}{7n}$. □ □

We now prove the Theorem 2.6.1. Lemmas 2.6.4 to 2.6.6 and Corollary 2.6.7 show that with probability at least $1 - \frac{1}{7n}$, the root accounts for at least τ 1-inputs while ensuring the following:

1. Filter nodes receive no more than $42 \log n$ messages and send no more than $21 \log n$ messages.

2. Collection nodes receive no more than $42 \log n$ messages and send no more than $14 \log n + 1$ messages. (The extra 1 is for the ⟨Count⟩ message.)

3. The root receives no more than $\tau/2^{D-1} = 28 \log n$ ⟨Flag⟩ messages.

Additionally, the adding nodes each receive two ⟨Count⟩ messages and send one ⟨Count⟩ message, and the root receives $D \leq \log n$ ⟨Count⟩ messages, one from each of the collection subtrees. Individual parties send at most one message each, when their input is set to **1**. We have already remarked that the messages used in this algorithm can be encoded using two bits. Thus, in the Up stage of the algorithm each party sends and receives $O(\log n)$ messages of constant size. In the Down stage, ⟨Done⟩ messages are sent via a canonical complete binary tree, so each party except the root receives exactly one ⟨Done⟩ message, and each party that is not a leaf in the tree sends (at most) two ⟨Done⟩ messages. Since all messages that are sent are eventually received, eventually all the parties receive the ⟨Done⟩ message and terminate. Since the depths of the data structure used in the Up stage and the binary tree used in the Down stage are both $\log n$, the longest chain of messages is of length $2 \log n$, and hence the total latency is $O(\log n)$. Finally, since the computations done by each node during the algorithm amount to counting the number of messages it receives and generating up to $14 \log n$ random numbers, each node performs $O(\log n)$ computations. □

2.6.5 Using Quorums as Nodes in the Count Tree

So far in this section, we have assumed that all of the nodes in the count tree follow the protocol honestly. However, this is not the case in our MPC model, where some of the parties can play maliciously. To fix this, we assign a quorum to each node in the tree and let the quorums perform the roles of the parties. In our MPC protocol described in Section 2.4, we introduce Protocol 4 that allows us run the threshold counting algorithm in a malicious setting.

Lemma 2.5.2 shows that a quorum Q can securely send a message M to another quorum Q'. However, there is some subtlety involved in using this fact. Every party in a quorum communicates with its parent when it has received at least half as many inputs as the parents' threshold. However, due to asynchrony, multiple messages may arrive simultaneously; when the threshold is set, not all parties in the quorum may be in the same state. Some may already have more inputs than the threshold, while others may still be waiting, because messages from their children have been delayed. Lemma 2.5.2 tells us that if all parties in the quorum send the *same* message to the parent quorum, then the parent quorum can resolve that message. Thus, in order to ensure that all parties in the quorum send the same message to the parent quorum, we have required that even if a party's received inputs exceed his threshold, it should only inform the parent of having met the threshold, not of having exceeded it. The remaining inputs are held to be sent later.

2.7 Asynchronous Quorum Formation

In this section, we describe the quorum building algorithm of King *et al.* [KSSV06b, KLST11], and then adapt it to the asynchronous communication model by proving the following theorem:

Theorem 2.7.1. Consider n parties connected to each other pairwise in an asynchronous network, where up to $t < (\frac{1}{4} - \epsilon)n$ of them are corrupted, for some small constant $\epsilon > 0$. If all honest parties follow the protocol Build-Quorums, then with high probability,

1. the parties agree on n quorums,
2. each party sends at most $\tilde{O}(\sqrt{n})$ field elements,
3. each party performs $\tilde{O}(\sqrt{n})$ computations, and
4. the protocol latency is $O(\text{polylog}(n))$.

One may alternatively use the asynchronous Byzantine agreement protocol of Braud-Santoni *et al.* [BGH13] to build a set of n quorums. This protocol requires each party on average to send

Protocol 14 Build-Quorums

Goal. Generate n quorums.

1. All parties run SRS-Agreement.
2. All parties run SRS-to-Quorum.

polylog(n) field elements, and perform polylog(n) computations. However, it is not load-balanced: some parties may send a linear number of field elements. Using this result our MPC protocol needs only logarithmic bits and computations.

We start the description of our protocol by defining the *semi-random-string agreement problem*, where the goal is to agree on a single string of length $O(\log n)$ with a constant fraction for random bits, where for any positive constant ϵ, a $1/2+\epsilon$ fraction of the parties are honest. King *et al.* [KLST11] present an asynchronous algorithm as an additional result that we call SRS-to-Quorum. The SRS-to-Quorum algorithm can go from a solution for *semi-random-string agreement* problem to the solution for the *quorum building* problem. Thus, their techniques can be extended to the asynchronous model assuming a scalable asynchronous solution for the semi-random-string agreement problem. We describe Build-Quorums algorithm based on SRS-to-Quorum and an algorithm, that solves semi-random-string agreement problem in the asynchronous model with pairwise channels that we call SRS-Agreement.

King *et al.* [KSSV06b] present a synchronous algorithm that a set of parties, up to 1/3 of which are controlled by an adversary, can reach almost-everywhere[1] agreement with probability $1-o(1)$. Their main technique is to divide the parties into groups of polylogarithmic size; each party is assigned to multiple groups. In parallel, each group uses bin election algorithm [Fei99] to *elect* a small number of parties from within their group to move on. This step is recursively repeated on the set of elected parties until size of the remaining parties in this set becomes polylogarithmic. At this point, the remaining parties can solve the semi-random-string agreement problem (similarly, they can run a Byzantine agreement protocol to agree on a bit). Provided the fraction of corrupted parties in the set of remaining parties is less than $1/3$ with high probability, these parties succeed in agreeing on a semi-random string. Then, these parties communicate the result value to the rest of the parties.

Bringing parties to agreement on a semi-random string is trickier in the asynchronous model. The major difficulty is that the bin election algorithm cannot be used in asynchronous model since

[1] King *et al.* [KSSV06b] relax the requirement that all honest parties reach agreement at the end of the protocol, instead requiring that a $1-o(1)$ fraction of honest parties reach agreement. They refer to this relaxation as almost-everywhere agreement.

Protocol 15 Simple-Elect-Subcommittee

Goal. $\Omega(\ln^8 n)$ parties agree on a subcommittee of size $\Omega(\ln^3 n)$. The protocol is performed by parties $P_1,...,P_k \in W$ with $k = \Omega(\ln^8 n)$.

1. Party P_i generate a vector of $c \ln^3 n$ random numbers chosen uniformly and independently at random from 1 to k where each random number maps to one party.
2. Run CMPC to compute the component-wise sum modulo k of all the vectors. Arbitrarily, add enough additional numbers from 1 to k to the sum vector to ensure it has $c \ln^3 n$ unique numbers.
3. Let W_B be the set of winning parties which are those associated with the components of the sum vector.
4. Return W_B as the elected subcommittee.

the adversary can prevent a fraction of the honest parties from being heard, and then prevent them to be part of the election. We present a similar algorithm to [KSSV06b] that solves this issue in asynchronous model with private channels. The main result of this section is as follows.

Theorem 2.7.2. Suppose there are n parties, for any fix positive ϵ constant fraction $b < 1/4 - \epsilon$ of which are corrupted. There is a polylogarithmic (in n) bounded degree network and a protocol such that:

1. With high probability, a $1 - O(1/\ln n)$ fraction of the honest parties agree on the same value (bit or string).
2. Every honest party sends and processes only a polylogarithmic (in n) number of bits.
3. The number of rounds required is polylogarithmic in n.

The important novelty of our method compare to King *et al.* [KSSV06b] is that instead of bin election algorithm, we use CMPC to decide on the parties who move on to the next level. The simple version of our election method is presented as Simple-Elect-Subcommittee in Protocol 15 that has the properties described in Lemma 2.7.3. The complete protocol and its proof of correctness are given in Section 2.7.5

Lemma 2.7.3. Let W be a committee of $\Omega(\ln^8 n)$ parties, where the fraction, f_W, of honest parties is greater than 3/4. Then, there exists some constant c, such that with high probability, the Elect-Subcommittee protocol elects a subset W_B of W such that $|W_B| = c \ln^3 n$ and the fraction of honest parties in W_B is greater than $(1 - 1/\ln n) f_W$. The Elect-Subcommittee protocol uses a polylogarithmic number of bits and polylogarithmic number of rounds in a fully connected network.

Proof. The proof follows from a straightforward application of union and Chernoff bounds. Let X be the number of honest parties in W_B. By the correctness of the CMPC algorithm, each party in W_B is randomly chosen from W. Let Y_i be an indicator random variable, that equals to 1 if the i-th member of W_B is honest. Then, $E[Y_i] = f_W$ and $E[X] = f_W c_1 \ln^3 n$. Using Chernoff bounds, we have $Pr[X < (1-1/\ln n) f_W c_1 \ln^3 n] = Pr[X < (1-1/\ln n)E[X]] \leq e^{-\frac{E[X]/\ln^2 n}{2}} < 1/n^c$. Since $f_W > 1/2$, setting $c_1 = 4c$, establishes the first part of Lemma 2.7.3. □ □

We establish a polylogarithmic bound on the number of bits used in Elect-Subcommittee protocol since the bit cost of Elect-Subcommittee is polynomial in the number of parties participating in the algorithm.

2.7.1 The Election Graph

Our algorithms make use of an election graph to determine which parties will participate in which elections. This graph was described in [KSSV06a, KSSV06b] and is repeated here.

Before describing the election graph, we first present a result similar to that used in [CL95]. Let X be a set of parties. For a collection $\mathcal{F}$ of subsets of X, a parameter δ, and a subset X' of X, let $\mathcal{F}(X',\delta)$ be the sub-collection of all $F' \in \mathcal{F}$ for which

$$\frac{|F' \bigcap X'|}{|F'|} > \frac{|X'|}{|X|} + \delta.$$

In other words, $\mathcal{F}(X',\delta)$ is the set of all subsets of $\mathcal{F}$ whose overlap with X' is larger than the "expected" size by more than a δ fraction. Let $\Gamma(r)$ denote the neighbors of node r in a graph.

Lemma 2.7.4. Let l,r,n be positive integers such that l and r are all no more than n and $r/l \geq ln^{1-z}n$. Then, there is a bipartite graph $G(L,R)$ such that $|L| = l$ and $|R| = r$ and

1. Each node in R has degree $\ln^z n$.
2. Each node in L has degree $O((r/l)\ln^z n)$.
3. Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then, for any subset L' of L, $|\mathcal{F}(L',1/\ln n)| < \max(l,r)/\ln^{z-2} n$.

The proof of Lemma 2.7.4 follows from a counting argument using the probabilistic method and is omitted. The following corollaries follows immediately by repeated application of the above lemma.

Corollary 2.7.5. Let ℓ^* be the smallest integer such that $n/\ln^{\ell^*} n \leq \ln^{10} n$. There is a family of bipartite graphs $G(L_i, R_i), i = 0, 1, \ldots, \ell^*$, and constants c_1 and c_2 such that $|L_i| = n/\ln^i n$, $|R_i| = n/\ln^{i+1} n$, and

1. Each node in R_i has degree $\ln^{c_1} n$.
2. Each node in L_i has degree $O(\ln^{c_2} n)$.
3. Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then, for any subset L'_i of L_i, $|\mathcal{F}(L'_i, 1/\ln n)| < |R_i|/\ln^6 n$.
4. Let $\mathcal{F}'$ be the collection of sets $\Gamma(l)$ for each $l \in L$. Then, for any subset R'_i of R_i, $|\mathcal{F}'(R'_i, 1/\ln n)| < |L_i|/\ln^6 n$.

Corollary 2.7.6. Let ℓ^* be the smallest integer such that $n/\ln^{\ell^*} n \leq \ln^{10} n$. There is a family of bipartite graphs $G(L_i, R_i), i = 0, 1, \ldots, \ell^*$, such that $|L_i| = n/\ln^i n$, $|R_i| = n/\ln^{i+1} n$, and

1. Each node in R_i has degree $\ln^5 n$.
2. Each node in L_i has degree $O(\ln^4 n)$.
3. Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then, for any subset L'_i of L_i, $|\mathcal{F}(L'_i, 1/\ln n)| < |L_i|/\ln^3 n$.

Lemma 2.7.4 and its corollaries show there exists a family of bipartite graphs with strong expansion properties which allow the formation of subsets of parties where all but a small fraction contain a majority that are honest.

We are now ready to describe the election graph. Throughout, we refer to nodes of the election graph as e-nodes to distinguish them from nodes of the static network. Let ℓ^* be the minimum integer ℓ such that $n/\ln^\ell n \leq \ln^{10} n$; note that $\ell^* = O(\ln n/\ln\ln n)$. The topmost layer ℓ^* has a single e-node which is adjacent to every e-node in layer $\ell^* - 1$. For the remaining layers $\ell = 0, 1, \ldots, \ell^* - 1$, there are $n/\ln^{\ell+1} n$ e-nodes. There is an edge between the ith e-node, A, in layer ℓ and the jth e-node, B, in layer $\ell + 1$ if and only if there is an edge between the ith node in $L_{\ell+1}$ and the jth node in $R_{\ell+1}$ from Corollary 2.7.6. In such a case, we say that B is the parent of A, and A is the child of B. Note that e-nodes have many parents.

Each e-node will contain a set of parties known as a *c*ommittee. All e-nodes, except for the one on the top layer and those in layer 0, will contain $c \ln^3 n$ parties. Initially, we assign the n parties to e-nodes on layer 0 using the bipartite graph $G(L_0, R_0)$ described in Corollary 2.7.6. The i^{th} party is a member of the committee contained in the j^{th} e-node of layer 0 if and only if there is an edge in

G between the i^{th} node of L_0 and the j^{th} node of R_0. Note every e-node on layer 0 has $\ln^5 n$ parties in it.

The e-nodes on higher layers have committees assigned to them during the course of the protocol. Let A be an e-node on layer $\ell > 0$, let $B_1, \ldots, B_s$ be the children of A on layer $\ell - 1$, and suppose that we have already assigned committees to e-nodes on layers lower than ℓ. If $\ell < \ell^*$, we assign a committee to A by running Elect-Subcommittee on the parties assigned to $B_1, \ldots, B_s$, and assigning the winning subcommittee to A. (Note that we can run each of these elections in parallel.) If A is at layer ℓ^*, the parties in A, $B_1, \ldots, B_s$, run byzantine agreement for Byzantine agreement.

2.7.2 Static Network with Polylog-Bounded Degree

We now repeat the description of the bounded degree static network [KSSV06b] and show how it can be used to hold elections specified by the election graph. For each e-node A, we form a collection of parties which we call it s-node: $s(A)$. Intuitively, the s-node $s(A)$ serves as a central communication point for an election occurring at e-node A. Our goal is to bound the fraction of s-nodes controlled by the adversary by a decreasing function in n, namely $1/\ln^{10} n$, for each layer. As the number of s-nodes grows much smaller with each layer, we need to make each s-node more robust. To do this, the number of parties contained in the s-node increases with the layer. Specifically, the s-nodes for layer i are sets of $\ln^{i+12} n$ parties. We determine these s-nodes using the bipartite graph from Lemma 2.7.4, where L is a collection of n nodes, one for each party, R is the set of s-nodes for layer i and the degree of each node in R is set to $\ln^{i+12} n$. The neighbors of each node in R constitute a set of parties in an s-node on layer i.

We use the term *link* to denote a direct connection in the static network. The communications for an election A will all be routed through $s(A)$: a message from a party x to $s(A)$ on layer i will pass from the party to a layer 0 s-node, whose parties will forward the message to a layer 1 s-node and so on, the goal being to reliably transmit the message via increasingly larger s-nodes up to $s(A)$. Similarly, communications to an individual party x from $s(A)$ will be transmitted down to a layer 0 s-node whose parties will transmit the message to x. We describe the connections in the static network.

Connections in the static network. Consider the following:

- Let A be an e-node on layer 0 in the election graph. Every party in A has a link to every party in $s(A)$.

- Let A and B be e-nodes in the election graph at levels i and $i-1$ respectively such that A is a parent of B. Thus, $s(A)$ has $\ln^{i+12} n$ parties in it and $s(B)$ has $\ln^{i+11} n$ parties in it. Let G be a bipartite graph as in Lemma 2.7.4 where L is the set of parties in $s(A)$, R is the set of parties in $s(B)$ and the degree of R is set to $\ln^{c_1} n$ and the degree of L is set to $O(\ln^{c_2} n)$. If there is an edge between two nodes in L and R respectively, then the corresponding party in $s(A)$ has a link to the corresponding party in $s(B)$. We will sometimes say that $s(A)$ is adjacent to $s(B)$ in the static network.

The following lemma follows easily from the application of Lemma 2.7.4 and its corollaries. Item (1) follows from Lemma 3.1; items (2) and (4) from Corollary 3.2; and item (3) from Corollary 3.1. Although item (2) only makes a guarantee about layer 0 e-nodes, we will see eventually that with high probability, the fraction of corrupted e-nodes on every layer is small.

Lemma 2.7.7. With high probability, the election graph and the static network have the following properties:

1. (Bad s-nodes) Any s-node whose fraction of corrupt parties exceeds $b + 1/\ln n$ will be called *b*ad. Else, we will call the s-node *g*ood. No more than a $1/\ln^{10} n$ fraction of s-nodes on any given layer are bad.

2. (Bad e-nodes) Any e-node whose fraction of corrupt parties exceeds $b + 1/\ln n$ will be called *b*ad. Else we call the e-node *g*ood. No more than a $1/\ln^2 n$ fraction of e-nodes on layer 0 are bad.

3. (Bad s-node to s-node connection) For any pair of e-nodes A and B joined in the election graph, the parties in s-nodes $s(A)$ and $s(B)$ are linked such that the following holds. For any subset W_A of parties in $s(A)$, at most a $1/\ln^6 n$ fraction of parties in $s(B)$ have more than a $|W_A|/|s(A)| + 1/\ln n$ fraction of their links to $s(A)$ with parties in W_A.

4. (Bad e-node to e-node connection) Let $|I|$ represent the total number of e-nodes on layer i in the election graph. For any set W of e-nodes on any layer i, at most a $1/\ln^2 n$ fraction of e-nodes on layer $i+1$ have more than $|W|/|I| + 1/\ln n$ fraction of their neighbors in W.

The degree of the static network is polylogarithmic.

2.7.3 Communication Protocols

A *permissible path* is a path of the form $P = x, s(A_0), s(A_1), ..., .s(A_i)$ where x is a party in A_0, i is the current layer of elections being held, each A_j is an e-node on layer j, and there is an edge in the election graph between A_j and A_{j+1} for $j = 0, ..., i$. Each party y in an s-node $s(A)$ on each layer j keeps a *List* of permissible paths that determine which parties' messages it will forward. The List (for $y \in s(A)$) represents y's view of which parties are elected (to the subcommittee) at A that are still participating in elections on higher layers. If y's List indicates that x is such a party, then the List will also have the entire path for x, which stretches from x to the elections on layer i in which x is currently participating in. We have the following definitions.

- We say a *s-node knows a message* [resp., *knows a permissible path*, or resp., *knows a* List *of permissible paths*] if $1 - b - 2/\ln n$ parties in the s-node are honest and receive the same message [resp., have the same path on their Lists, or resp., all have the same List.]
- A permissible path P is good if every s-node on the path knows P. Else the path is bad. We will show our construction of the static network ensures at most a $1/\ln n$ fraction of the permissible paths are bad.

We now describe three primitive communication subroutines: Sendhop, Send, and MessagePass. The subroutine Sendhop describes how s-nodes (with direct links) communicate with each other, Send describes how a party communicates with an s-node, and MessagePass describes how two parties communicate with each other.

Sendhop(s, r, m, P)**.** A message m can be passed from s (the sender) to r (the receiver) from a level i to a level $i-1$ or from a level i to a level $i+1$, where s and r are s-nodes on these layers or one of s, r is a 0-layer s-node and the other is a party. If a party x sends a message to a layer 0 s-node $s(A)$ it sends the message to every party in $s(A)$ (note by construction it will have a direct link with every party in $s(A)$). Similarly if a message is sent from a layer 0 s-node $s(A)$ to a party x, every party in $s(A)$ sends the message to x.

When an s-node $s(A)$ sends a message to s-node $s(B)$, every party in $s(A)$ sends the message to those parties of $s(B)$ to which it has a direct link. When each party in $s(B)$ receives such a set of messages, it waits until it receives the same messages from the majority to determine the message. If there is no majority value, the party ignores the message. Along with sending the message the parties also send information which specifies along which path P the message is being sent. Each time a message is received by a party of an s-node $s(B)$ on layer $j \leq i$, it checks that:

1. The message came from the s-node previous to it in the path P; if not the message is dropped.

2. The path P (or its reverse) is on its List of permissible paths. If not, the message is dropped.

3. Only messages that conform to the protocol in size and number are forwarded up and down the permissible paths. If more or longer messages are received from a party, messages from that party are dropped.

Send(s,r,m,P)**.** Of the first two parameters, one must be a party ("x") and one must be an s-node ("$s(A)$"). The path P contains the first parameter s as its start and the second parameter r as its endpoint. Send(s,r,m,P) sends the message m from s to r along the path P via repeated application of Sendhop.

MessagePass$(x \in A, y \in B, m, P_x, P_y)$**.** Both A and B are adjacent e-nodes. Hence, $s(A)$ and $s(B)$ are adjacent in the static network. A message from party x in e-node A sends message m to party y in e-node B by first calling Send$(x,s(A),m,P_x)$. Then, $s(A)$ sends m to $s(B)$ by calling Sendhop$(s(A),s(B),m,P)$, where P is the path consisting of two s-nodes $s(A),s(B)$. Finally, the message is transmitted from $s(B)$ to y by calling Send$(s(B),y,m,P_y^r)$, where P_y^r is the reversal of path P_y.

2.7.4 SRS-Agreement Protocol

Before describing the SRS-Agreement protocol, we first adapt the Elect-Subcommittee protocol for the static network. Let A be an e-node with children $B_1,\ldots,B_s$, and let X be the set of all parties from $B_1,\ldots,B_s$. For each $i \in [s]$ and $x \in B_i$, let P_x denote a good path of s-nodes from x to $s(B_i)$ concatenated with $s(A)$. At the start of the election for A, we assume that each node in P_x knows P_x and $s(A)$ knows $\{P_x \mid x \in X\}$.

We now describe the implementation of the Elect-Subcommittee algorithm. Every party $x \in X$ generate a vector of random numbers chosen uniformly and independently at random where each random number maps to one party. The parties use the CMPC protocol to determine the winners (recall that the number of parties in e-nodes is always polylogarithmic, so this can be done sending only polylogarithmic messages). The list of winners is sent up to $s(A)$, where each party in $s(A)$ takes a majority to determine the winners. Then, $s(A)$ sends down the list of winners along all the permissible paths to each party $x \in X$. Parties on the path (*i.e.* , in s-nodes along the path) update their Lists of permissible paths to remove those party-paths who lost as well as those party-pairs who won too many elections (we will quantify this shortly), and make $\ln^4 n$ copies of each of the

Protocol 16 Elect-Subcommittee

Goal. Adapted version of Simple-Elect-Subcommittee for static networks.

1. For each $x \in X$: *// This stage done in parallel*
2. Party x randomly selects one of $k/(c_1 \ln^3 n)$ random numbers chosen uniformly and independently at random from zero to k where each random number maps to one party.
3. Parties in X run CMPC to compute the component-wise sum modulo k of all the vector. Arbitrarily, add enough additional numbers to the vector to ensure it has $c \ln^3 n$ unique numbers.
4. Let M be the set of winning parties, which are those associated with some component of the vector sum.
5. Each $y \in X$ sends M to $s(A)$ by calling Send$(y, s(A), M, P_y)$.
6. Parties in $s(A)$ determine M by waiting until they receive the majority of same messages. These become the elected parties.
7. For each party $x \in X$ that is elected, the parties in $s(A)$ use Send$(s(A), x, m, P_x^r)$ to tell x, along with each s-node in P_x, that x was elected.
8. Each party in each s-node revises its list of permissible paths to:

   Retain only the winners. Eliminate parties who have won more than 8 elections. Make $\ln^4 n$ copies of remaining permissive paths, concatenating each with a different s-node neighbor on layer $i+1$.
9. $s(A)$ sends its list to every adjacent s-node $s(B)$ on layer $i+1$ using Sendhop$(s(A), s(B), m, P)$, where P is the path consisting only of $s(A)$, $s(B)$.

Protocol 17 Scalable-SRS-Agreement

Goal. Parties agree on a semi-random string.

1. For $l = 1$ to l^*:
2. For each e-node A in layer l, let $B_1, ..., B_s$ be the children of A in layer $l-1$ of the election graph, and
3. If $l < l^*$, run Elect-Subcommittee on the parties in nodes $B_1, ..., B_s$. Assign winning parties to node A.
4. Else parties in nodes $B_1, ..., B_s$ solve semi-random-string agreement problem.
5. Let A^* be the e-node on layer l^*, every party x assigned to A^* communicates the result of Step 4 to $s(A^*)$ using Send$(x, s(A^*), m, P_x)$.
6. Every party in $s(A^*)$ waits for the majority of same message to determine the result of Step 4.

winners' paths and concatenate a different layer $i+1$ s-node parent onto each one. We present a detailed description of Elect-Subcommittee in the following.

The condition in Step 5 that requires parties who have won more than 8 elections to be eliminated is a technical condition that insures the protocol is load-balanced and parties in an s-node do not communicate more than a polylogarithmic number of bits. We now describe the SRS-Agreement protocol.

Since every party is a member of $s(A^*)$, steps 5 and 6 will insure the final result of the protocol is communicated to every party.

2.7.5 Proof of Build-Quorums

To establish the correctness of the protocol presented in Section 2.7.4, we first state some claims regarding the primitive communication protocols. Their proofs follow by straightforward probabilistic arguments and are omitted in the interest of space. Recall the fraction of corrupted parties is b, where $b < 1/4 - \epsilon$ for any fix positive ϵ.

Claim 2.7.8. Let $s(A)$ and $s(B)$ be s-nodes on consecutive layers. Assume the following conditions hold:

1. Both $s(A)$ and $s(B)$ are good.
2. $s(A)$ is on a permissible path known by $s(B)$.
3. There exists a set W of parties from $s(A)$ such that for every message m, all parties in W are honest and agree on a message m. Further W consists of at least a $1 - b - 2/\ln n$ fraction of the parties in $s(A)$.

Then, there is a set W' of parties from $s(B)$ such that for every message m, every party in W' is honest and agrees on the message m after Sendhop$(s(A), s(B), m, P)$ is called. (Here, P is the path $s(A), s(B)$.) Further, W' consists of all but a $1/\ln^6 n$ fraction of the honest parties in $s(B)$.

Proof. Every party in W is honest and sends the same massage to its connected parties in $s(B)$. The parties in $s(B)$ can afford to wait for the majority of same messages, since $s(A)$ is good and W consists of at least $1 - b - 2/\ln n$ fraction of parties which is more than $1/2$ and for majority we need to receive a fraction of $1/2$ same messages from the parties in $s(A)$. Thus, all honest party but a $1/\ln^6 n$ fraction of parties in $s(B)$ will eventually receive the message based on corollary 2.7.5. □ □

Claim 2.7.9. Let x be an honest party. Assume P_x is a good path. Then, after Send$(x, s(A), m, P_x)$ is executed, there is a fixed set W of honest parties which contains all but a $1/\ln^6 n$ fraction of the honest parties in $s(A)$ and every party $z \in W$ agrees on m.

An election at e-node A is *legitimate* if the following two conditions hold simultaneously for more than a $3/4$ fraction of parties x participating in the election at A: (1) party x is honest; (2) The path P_x is good.

Lemma 2.7.10. For a legitimate election at node A, let X be a set of honest parties with good permissible paths. (Note $|X| > 3\ln^8 n/4$.) Let W be the set of honest parties in $s(A)$ that know X. Then, after the execution of Elect-subcommittee, the parties in W know the winners of the election in A, as do the s-nodes that belong to good paths P_x.

Proof. From Claim 2.7.9, we have that every message m sent by MessagePass($y \in B_i, z \in B_j, m, P_y, P_z$) from $y \in X$ to $z \in X$ is received by some fixed set W of honest parties in $s(B_i)$, such that W contains at least $1 - b - 2/\ln n$ fraction of the parties in $s(B_i)$. By Claim 2.7.8, every message sent by y is received by z. Since X contains more than 3/4 of the total parties participating in the election, (after running CMPC) all the parties in X will all agree on the same set of for random parties. Thus, after the parties in X send these values to $s(A)$, $s(A)$ will know the winners. When $s(A)$ sends these winners to X, by repeated application of Claim 2.7.8, we have every $x \in X$ and every s-node in P_x will know these winners. □ □

We have shown that in a legitimate election at node A, $s(A)$ knows the list of winners. We next consider when paths are dropped from the permissible path Lists.

2.7.5.1 Permissible Paths Removal

Let y be a party in some s-node on layer i. A permissible path P_x is removed from a party y's List on layer i if y receives a message from an s-node above it in P_x, indicating either x has won more than 8 elections or x lost in the election held at the last node of P_x. Here, we consider when P_x is removed for the former reason, *i.e.* , we give an upper bound on the fraction of parties that are reported to have won too many elections on layer i.

First we consider the effect of legitimate elections. The following lemma, a version of which appears in [KSSV06a, KSSV06b], shows that on a given layer a very small fraction of honest parties win more than 8 times in legitimate elections.

Lemma 2.7.11. With high probability, the parties that win more than 8 elections, counting multiplicities, account for no more than a $16/\ln^3 n$ fraction of the honest parties that are winners of legitimate elections.

Next, we bound the effect of elections that are not legitimate. We first consider the case where $s(A)$ is good, yet the fraction of honest parties participating in A with good paths is less than 3/4. For the remainder of the proof we shall treat such an e-node A as a bad e-node.

Claim 2.7.12. Suppose less than a $1/7$ fraction of the honest parties of a good $s(A)$ agree on a message m. Then, after Sendhop$(p(A),p(B),m,P)$ is executed, all but a $1/\ln^6 n$ fraction of the honest parties in $s(B)$ will ignore m.

Proof. Even if the corrupted parties agree on m, since $b < 1/4$, the total fraction of parties in $s(A)$ sending the message m is less than $11/28$. Thus, at most a $1/\ln^6 n$ fraction of the parties in $s(B)$ will receive m from a majority of parties in $s(A)$. □ □

Hence, a good $s(A)$ can only communicate with seven different sets of winners to the s-nodes below it. Since each honest party will send $\ln^3 n$ winners, the total number of winners sent is at most $7\ln^3 n$. Therefore, a bad e-node can cause at most $7\ln^3 n$ parties to have their permissible paths removed.

Next, we consider the effect of a bad s-node. We will assume one bad s-node $s(A)$ on layer i can cause the removal of all the permissible paths for every party participating in the election at A. Since $\ln^8 n$ parties participate in an election, and fewer than a $1/\ln^{10} n$ fraction of the s-nodes are bad on a layer, the fraction of honest winners affected is less than $1/\ln^2 n$. Thus, we can bound the fraction of the honest winners on any layer i that have their permissible paths removed by $1/\ln^2 n + 1/\ln^3 n + 7\beta_i$; where β_i represents the fraction of bad e-nodes on layer i. Thus, we have the following lemma.

Lemma 2.7.13. Assume the fraction of bad e-nodes on layer i is bounded by $c/\ln^2 n$, for some constant c. Then, the fraction of honest winners that have their permissible paths removed on layer i is bounded by $8c/\ln^2 n$.

2.7.5.2 Proof of Theorems 2.7.2

We now complete the proof of Theorem 2.7.2 which follows from the following lemma.

Lemma 2.7.14. On layer i, with high probability, at least a $1 - 4/\ln^2 n$ fraction of s-nodes $s(A_j)$ have the following properties:

1. $s(A_j)$ is good.
2. At least a $1 - b - 4i/\ln n$ fraction of the parties in node A_j are honest and have good paths to $s(A_j)$ (note this implies $s(A_j)$ knows this path). That is, A_j is a good e-node.

Proof. We prove the lemma by induction. On all layers and particularly layer 0, only a $1/\ln^{10} n$ fraction of the s-nodes are bad. If $s(A)$ is good, then every party in A has a good path to $s(A)$. Further by construction all but a $1/\ln^2 n$ fraction of the e-nodes on layer 0 consist of at least a $1-b-1/\ln n$ fraction of honest parties. So the lemma is true on layer 0.

Assume the lemma is true for layer i. Then, a $1-4/\ln^2 n$ fraction of e-nodes are good, more specifically these e-nodes have at least a $1-b-4i/\ln n$ fraction of honest parties that have a good path to their corresponding s-node. Since the election is legitimate by Lemmas 2.7.3 and 2.7.10, with high probability, after Elect-Subcommittee at least a $1-b-4i/\ln n-1/\ln n$ fraction of the parties elected are honest and have a good path to any good parent of their s-node. Thus, at least a $1-b-(4i+1)/\ln n$ fraction of the parties elected at layer i are honest and have good paths to good parent s-nodes on layer $i+1$. By Lemma 2.7.13 this fraction is reduced by at most $32/\ln^2 n$. Thus, at least a $1-b-(4i+2)/\ln n$ fraction of the parties elected at layer i are honest and have good paths to good parent s-nodes on layer $i+1$. Since the fraction of bad s-nodes on layer $i+1$ is at most $1/\ln^{10} n$, by Corollary 2.7.6 at least a $1-1/\ln^2 n-1/\ln^{10} n$ fraction of the e-nodes (and their corresponding s-nodes) are good on layer $i+1$, and have at least a $1-b-(4i+2)/\ln n-1/\ln n$ fraction of honest parties that have good paths to their corresponding s-nodes. □ □

By Lemma 2.7.14, with high probability the layer ℓ^* e-node is good. Thus, the parties in this e-node succeed in solving the semi-random-string agreement problem (Step 4 of algorithm SRS-Agreement). Since all the parties are in the s-node (though they may appear multiple times) corresponding to A on ℓ^*, by Claim 2.7.9 all but a $O(1/\ln n)$ fraction of the honest parties learn the final result. To prove the number of bits sent by each party is polylogarithmic we note each party is in a polylogarithmic number of e-nodes and s-nodes on each layer i, and participates in at most a polylogarithmic number of election on layer i. Since the number of layers is $O(\ln n)$ Theorem 2.7.2 follows. Finally, the correctness of Theorem 2.7.1 follows from Theorem 2.7.2 and the correctness of SRS-to-Quorum protocol.

2.8 Conclusion

We described a Monte Carlo algorithm to perform asynchronous MPC in an scalable manner. Our protocols are scalable in the sense that they require each party to send $\tilde{O}(m/n+\sqrt{n})$ messages and perform $\tilde{O}(m/n+\sqrt{n})$ computations. They tolerate a static adversary that controls up to a $1/8-\epsilon$ fraction of the parties, for ϵ any positive constant. We showed that our protocol is secure in the universal composability framework. We also described efficient algorithms for two important

building blocks of our protocol: threshold counting and quorum building. These algorithms can be used separately in other distributed protocols.

The following problems remain open. Can we prove lower bounds for the communication and computation costs for Monte Carlo MPC? Can we implement and adapt our algorithm to make it practical for a MPC problem such as the beet auction problem described in [BCD+09]. Finally, can we prove upper and lower bounds for resource costs to solve MPC in the case where the adversary is *adaptive*, able to take over parties at any point during the algorithm?

Chapter 3

Scalable Mechanisms for Rational Secret Sharing

Secret sharing is one of the most fundamental problems in security, and is an important primitive in many cryptographic protocols. In t-out-of-n secret sharing, we are interested in designing a protocol to distribute shares of a secret to each n players and to reconstruct the secret ensuring that: (1) if any group of t players follow the protocol, they will all learn the secret; and (2) knowledge of less than t of the shares reveals nothing about the secret. In rational secret sharing, all players are rational. Moreover, we want our protocol to be a *Nash equilibrium* in the sense that no player can improve its utility by deviating from the protocol, given that all other players are following the protocol.

Recently, there has been interest in solving *rational secret sharing* [KN08, GK06, HT04, ADGH06, LT06]. Unfortunately, all previous solutions to this problem require each agent to send $O(n)$ messages in expectation, and so do not scale to large networks.

In this chapter, we address this issue by designing scalable mechanisms for rational secret sharing. Our main result is a protocol for rational n-out-of-n secret sharing that (1) requires each agent to send only an expected $O(\log n)$ bits; and (2) has $O(\log n)$ expected latency. We also design a scalable mechanism for t-out-of-n rational secret sharing when the parties are computationally bounded.

3.1 The Problem

Shares of a secret are to be dealt to n rational players, who will later reconstruct the secret from the shares. The players are *learning-preferring,* in the sense that each player prefers every outcome in which he learns the secret to any outcome in which he does not learn the secret. However, a player

may prefer the situation where he learns the secret and other players do not to the situation where all players learn the secret.

The secret is an arbitrary element of a large (fixed) finite field $\mathbb{F}_q$. At the beginning of the game, a dealer provides the shares to the players. The dealer has no further role in the game. The players must then communicate with each other in order to recover the secret.

Communication between the players is *point-to-point* and through secure private channels. In other words, if player A sends a message to player B, then a third player C is not privy to the message that was sent, or indeed even to the fact of a message having been sent. Communication is *synchronous* in that there is an upper-bound known on the maximum amount of time required to send a message from one player to another. However, we assume *non-simultaneous* communication, and thus allow for the possibility of *rushing*, where a player may receive messages from other players in a round before sending out his own messages.

Our goal is to provide protocols for the dealer and rational players such that the players following the protocol can reconstruct the secret. Moreover, we want a protocol that is *scalable* in the sense that the amount of communication and the latency of the protocol should be a slow growing function of the number of players.

3.2 Related Work

The problem of secret sharing with rational players was introduced by Halpern and Teague in [HT04]. They showed that, rational secret sharing is possible using randomized mechanisms with constant expected running time and it is impossible using a mechanism that has a fixed running time. They also show that there is no practical mechanism for 2-out-of-2 rational secret sharing even with an infinite game tree. A mechanism is practical if it is a Nash equilibrium that survives iterated deletion of weakly-dominated strategies. Since then, there has been significant work on the problem of rational secret sharing, including results of Gordon and Katz [GK06], Abraham et al. [ADGH06], Lysyanskaya and Triandopoulos [LT06], Asharov and Lindell [AL09], Kol and Naor [KN08] et al. [ZTW11] and Fuchsbauer et al. [FKN10].

Most of this related work except for [KN08, AL09, FKN10, ZTW11], assume the existence of simultaneous communication, either by broadcast or private channels. Several of the protocols proposed [GK06, ADGH06, LT06, ZTW11] require cryptographic assumptions and achieve an equilibrium under the assumption that the players are computationally bounded. In contrast, our results do not assume simultaneous communication and our n-out-of-n algorithm does not make any cryptographic assumptions.

Fuchsbauer et al. [FKN10] proposed protocols for 2-out-of-2 and t-out-of-n rational secret sharing. Similarly to our model, they assume the players only have point-to-point channels and they do not require broadcast channel. Unlike us, the proposed protocols in [FKN10] are based on cryptographic assumptions and the authors introduce the notion of a computational strict Nash equilibrium, that is a Nash equilibrium when all players are polynomial time bounded. They show that their protocols are both computational strict Nash. A caveat is that the protocols are susceptible to backwards induction.

Zhang, Tartary, and Wang in [ZTW11], propose a protocol for rational t-out-of-n secret sharing scheme based on the Chinese reminder theorem. Their protocol works in a communication model similar to our model. Their protocol requires a synchronous but not simultaneous broadcast channel and synchronous but not simultaneous point-to-point private channels. They assume some computational hardness related to the discrete logarithm problem and RSA. The proposed protocol is a $(t-1)$-resilient computational strict Nash equilibrium that is stable with respect to trembles. Their protocol runs in time polynomial in k and has shares of size $O(k)$ bits where k is the protocol security parameter.

The work of Kol and Naor [KN08] is closest to our own work and our protocols make use of several clever ideas from their result. Kol and Naor show that in the non-simultaneous broadcast model (*i.e.* , when rushing is possible), there is no Nash equilibrium that ensures all agents learn the secret, at least for the case of two players. They thus consider and solve the problem of designing an ϵ-Nash equilibrium for the problem in this communication model. Furthermore, the equilibrium they achieve is *everlasting*.

The impossibility of a Nash equilibrium for two players carries over to the setting with secure private channels, since there is no difference between private channels and broadcast channels when there are only two players. However, one might hope that the algorithm of [KN08] could be efficiently simulated over secure private channels to give an everlasting ϵ-Nash equilibrium. Unfortunately, simulation of broadcast over private channels is expensive. To the best of our knowledge, a single broadcast requires each player to send $\Theta(n)$ messages.

In [DMRS11], Dani et al. overcame this difficulty, by providing a *scalable* protocol for rational secret sharing, in which each player only sends $O(1)$ bits per round and the expected number of rounds is constant (although each round takes $O(\log n)$ time). Moreover, following the protocol is an ϵ-Nash equilibrium. Unfortunately, a certain bad event with small but constant probability caused some players, when they recognized it, to deviate from the protocol so that the equilibrium from [DMRS11] is not everlasting.

3.3 Our Results

The main result of this chapter is presented as Theorem 3.3.1.

Theorem 3.3.1. Let $n \geq 3$. There exists a protocol for rational n-out-of-n secret sharing with the following properties.

- The protocol is a Nash equilibrium in which all players learn the secret.
- In expectation, the protocol requires each player to send $O(\log n)$ bits, and has latency $O(\log n)$.
- The protocol is not robust to coalition on any size.

This chapter is the full version of the extended abstract in [DMRS11]. Additionally, it improves on the work in [DMRS11] in two ways. First, the new proposed algorithm for n-out-of-n secret sharing is correct with probability one and there is no probability of error. Second, we show that our new protocol for n-out-of-n is a Nash equilibrium, not just an ϵ-Nash equilibrium, as long as $n \geq 3$.

***t*-out-of-*n* secret sharing** We also consider the problem of t-out-of-n rational secret sharing for the case where $t < n$. Scalable rational secret sharing for the t-out-of-n case may also be of interest for applications like the Vanish peer-to-peer system [GKLL09]. Vanish uses secret sharing in a peer-to-peer system with churn in order to provide data that vanishes over time. In this setting we prove the following.

Theorem 3.3.2. Let $n \geq 3$ and $t \leq n$. Assuming the players are computationally bounded to polynomial-time algorithms, there exists a protocol for rational t-out-of-n secret sharing with the following properties.

- The protocol is an everlasting ϵ-Nash equilibrium in which all active players learn the secret, where ϵ can be any arbitrary positive value.
- In expectation, the protocol requires each player to send $O(\log n)$ bits, and has latency $O(\log n)$.
- The protocol is not robust to coalition of any size.

This is an improvement to the $\Theta(n)$-out-of-n result we proved in [DMRS11], in the sense that in this new result, t can be any number smaller than n. However, in our new t-out-of-n protocol, we require a cryptographic digital signature scheme with security parameter $\kappa = \max_j \frac{U_j^+ + U_j}{2U_j + \epsilon}$. See Section 3.6.2 for definitions of U_j^+ and U_j.

3.4 Our Approach

The difficulty in designing a Nash equilibrium in a communication model where rushing is possible, is that the last player to send out his share has no incentive to actually do so. He already has the shares of all the other players and can recover the secret alone. To get around this, it is common (see [HT04, ADGH06, GK06, LT06, KN08]) for the protocol to have a number of fake rounds designed to catch cheaters. The uncertainty in knowing which is the "definitive" round, during which the true secret will be revealed causes players to cooperate.

In the work of [KN08] this uncertainty is created by dealing one player only enough data to play until the round preceding the definitive one. Thus, there is a single "short" player and $n-1$ "long" players. None of the players know whether they are short or long. The long players must broadcast their information every round, since they cannot predict the definitive round in advance. The short player knows the definitive round in advance, but has no information about the secret in that round. In the definitive round the short player is the last to speak so that he (and all the other players) receives the shares of all the long players and can recover the secret. The short player's failure to broadcast a message is what cues the other players to the end of the game, and they too can recover the secret.

Moreover, having learned the secret, the short player cannot pretend that he actually had a share for that round as the messages sent by all the players are verified by a tag and hash scheme (see, e.g., [WC81, RBO89, KN08]). In fact, it is the small but positive chance of cracking the tag and hash scheme that results in the protocol from [KN08] being an ϵ-Nash equilibrium rather than a Nash equilibrium. In this chapter for the n-out-of-n case, we overcome this problem by ensuring there are at least two short players when $n \geq 3$. Thus, our protocol is a Nash equilibrium for this case since if a single short player breaks the tag and hash scheme, he cannot prevent any other player from learning the secret. See Section 3.8 for complete analysis.

Unfortunately, the same trick of having more than one short player to achieve a Nash-equilibrium cannot be easily applied to the t-out-of-n case. Since the dealer is not aware of the identity of the active players, he cannot choose short players in a way to make sure there are at least two short players active in the game. Since there is a chance that there is only one active short player, a short player will now have some incentive to try to forge the tag and hash scheme.

Due to the using of short and long players, both of our protocols are susceptible to coalitions of any size. Consider a long player who is the first one that learns the secret. He can join together with a short player to learn that this round is definitive. Thus, both players in this coalition will learn the secret early, and the remaining players will be fooled into thinking that the previous round was the

definitive round.

We introduce two novel techniques to ensure scalable communication. The first technique is to arrange players at the leaves and nodes of a complete binary tree, and require that the players only communicate with their neighbors in the tree. The assignment of players to the leaves is independently random in every round, and their assignment to internal nodes is related to their assignment to leaves according to a predefined scheme. Every round of the game, information travels up to the root where it is decoded and then travels back down again to the leaves. We also use short and long players to determine the definitive round. The short players are the players who are parents of the leaves in the definitive round and all other players are long players. So, every player is a child of a short player in the definitive round. In the definitive round, the short players will fail to send messages to all the leaf nodes. Thus, every player, upon not receiving a message, will learn that this is the definitive round and so will learn the secret.

The second main idea is that we make use of an iterated secret sharing scheme over this tree in order to divide up shares of secrets among the players. This scheme is similar to that used in recent work by King and Saia [KS10] on the problem of scalable Byzantine agreement, and suggests a deeper connection between the two problems.

One drawback of our both protocols is vulnerability to coalitions of any size. The player who is at the root in the definitive round can team up with a short player and learn the secret early without telling it to others.

As in previous work [WC81, RBO89, KN08] we use a information-theoretic message authentication codes (also referred as tag-and-hash scheme) to ensure that players cannot forge messages in the protocol in our n-out-of-n protocol.

3.5 chapter Organization

The rest of this chapter is laid out as follows. In Section 3.6, we give notation and preliminaries. In Section 3.7, we describe our algorithm for scalable n-out-of-n secret sharing. In Section 3.8, we analyze this algorithm; the main result of this section is a proof of Theorem 3.3.1. In Section 3.9, we give our algorithm and analysis for scalable t-out-of-n secret sharing; the main result of this section is a proof of Theorem 3.3.2. Finally in Section 3.10, we conclude and give directions for future work.

3.6 Notation and Preliminaries

The secret to be shared is an arbitrary element of a set $\mathcal{S}$. There are n players with distinct player ids in $[n] = \{1,2,\ldots n\}$. During the course of the algorithm, we will want to do arithmetic manipulations with player ids and shares of the secret, including adding in, or multiplying by random elements to preserve secrecy. In order to be able to do these sorts of manipulations, we embed the sets $\mathcal{S}$ and $[n]$ into a finite field $\mathbb{F}$ of size $q > \max\{n, |\mathcal{S}|\}$. The latter embedding will be the canonical one; the former may be arbitrary, but is assumed to be known to all parties.

The messages sent by players in the algorithm will be elements of $\mathbb{F}$. The length of any such message is $\log q = \Omega(\log n)$. Since our goal is to provide a scalable algorithm we cannot afford the message lengths to be much bigger than that. We will choose $\mathbb{F}$ to be a prime field of size $q = O(n)$. We remark that although generally $\mathcal{S}$ is of constant size, we can tolerate $|\mathcal{S}| = O(n)$.

3.6.1 Utility Functions

We will denote the utility function of player j by u_j. As mentioned before, we assume that the players are learning preferring, *i.e.* , each player prefers any outcome in which he learns the secret to every outcome in which he does not learn the secret. More formally, for outcome $\mathbf{o}$ of the game, let $R(\mathbf{o})$ denote the set of players who learn the secret. If $\mathbf{o}$ and $\mathbf{o}'$ are outcomes of the game such that $j \in R(\mathbf{o}) \setminus R(\mathbf{o}')$, then $u_j(\mathbf{o}) > u_j(\mathbf{o}')$. As in [KN08], we denote

$$\begin{aligned} U_j^+ &= \max\{u_j(\mathbf{o}) \mid j \in R(\mathbf{o})\} \\ U_j &= \min\{u_j(\mathbf{o}) \mid j \in R(\mathbf{o})\} \\ U_j^- &= \max\{u_j(\mathbf{o}) \mid j \notin R(\mathbf{o})\}. \end{aligned}$$

Thus U_j^+ is the utility to player j of the best possible outcome for j, U_j is the utility to j of the worst possible outcome in which j still learns the secret, and U_j^- is the best possible utility to j when he does not learn the secret. By the learning-preferring assumption, we have for all j,

$$U_j^+ \geq U_j > U_j^-.$$

We will denote by $\mathcal{U}$, the quantity

$$\mathcal{U} := \max_{j \in [n]} \frac{U_j^+ - U_j^-}{U_j - U_j^-}.$$

Note that $\mathcal{U} \geq 1$. We assume that $\mathcal{U}$ is constant, *i.e.* , that it does not depend on n.[1]

[1] Technically, we can achieve scalable (polylog) communication even if we allow $\mathcal{U}$ to be as big as polylog(n).

Similar to previous work on rational secret sharing [KN08], we also assume that the utilities are such that the players have an incentive to play the game rather than just guess the secret at random. Moreover, as in previous works, we assume that the prior distribution of the secret is uniform for all players. Thus, if player j decides to guess the secret, with probability $1/|\mathcal{S}|$ he can get at most the maximum utility of U_j^+ and with probability of $1-1/|\mathcal{S}|$ can get at most the maximum utility of U_j^-. Thus, player j's utility is at most $\frac{U_j^+-(1-|\mathcal{S}|)U_j^-}{|\mathcal{S}|}$ if he guesses instead of running the protocol. On the other hand, if the protocol runs correctly, he has utility at least U_j. Hence, we require $\frac{U_j^+-(1-|\mathcal{S}|)U_j^-}{|\mathcal{S}|} < U_j$ for all $1 \le j \le n$. To satisfy these inequalities we need: $|\mathcal{S}| > \max_{j\in[n]} \frac{U_j^+-U_j^-}{U_j-U_j^-}$. Thus we require,

$$\mathcal{U} < |\mathcal{S}|. \tag{3.1}$$

3.6.2 Game Theoretic Concepts

In this section, we review a few game-theoretic background concepts. We adapt the standard concepts here from the previous work (e.g. see [HT04, ADGH06]). We describe a game with a tree, where each node represents the local states of players based on their moves in previous rounds. The local state of player j describes player j's initial information and messages sent and received by player j. At each round, all the players move to a new state by deciding which messages to send. This decision can happen after receiving other players' messages. Each message takes exactly one round to arrive.

A strategy or protocol for player j is a function from player j's information set (*i.e.* , player j's local state) to actions. A strategy can be a randomized function. A joint strategy $\vec{\sigma}=(\sigma_1,\ldots,\sigma_n)$ is a tuple of strategies, one for each player. We let $\vec{\sigma}=(\vec{\sigma}_{-j},\sigma_j)$, where $\vec{\sigma}_{-j}$ denotes a tuple consisting of each player's strategy in $\vec{\sigma}$ other than player j's. $Ur_j(\vec{\sigma})$ is player j's expected utility if $\vec{\sigma}$ is played. Thus, the utility function depends on the path in the tree that starts from the root.

Recall that a joint strategies for a game is called a *Nash equilibrium* if no player has an incentive to unilaterally deviate from the equilibrium strategy, when all others are following it. Formally, $\vec{\sigma}$ is a Nash equilibrium if,

$$\forall j,\forall\sigma'_j, U_j(\vec{\sigma}_{-j},\sigma_j) \ge U_j(\vec{\sigma}_{-j},\sigma'_j).$$

In the same way, $\vec{\sigma}$ is an ϵ-Nash equilibrium if,

$$\forall j,\forall\sigma'_j, U_j(\vec{\sigma}_{-j},\sigma_j) + \epsilon \ge U_j(\vec{\sigma}_{-j},\sigma'_j).$$

In games of incomplete information which have multiple rounds, there is the further question of whether the players are forced to commit to their strategies before the start of the game or whether

they have the option to change strategies in the middle of the game, after some rounds have been played and they may learn some new information. Kol and Naor [KN08] defined a equilibrium to be *everlasting* if after any history that is consistent with all players following the equilibrium strategy, it is still true (despite whatever new information the players may have learned over that history) that a player choosing to deviate unilaterally cannot gain in expectation, *i.e.* , following the prescribed strategy remains in equilibrium. For some equilibrium concepts such as ϵ-Nash, everlastingness is a stronger concept than the same equilibrium where the strategies are committed to up front [KN08].

However, when the underlying equilibrium is Nash, everlastingness does not make a difference. That is, an everlasting Nash equilibrium is the same as a Nash equilibrium. Nash equilibrium cannot fail to be an everlasting equilibrium. Consider a Nash equilibrium joint strategy $\vec{\sigma}$ and a path related to this joint strategy in the game tree. If this joint strategy is not an everlasting equilibrium, then along the path, there is some node consistent with $\vec{\sigma}$= $(\vec{\sigma}_{-j},\sigma_j)$ at which some player, say player j, has an incentive to deviate and follow strategy σ'_j. Thus, the strategy σ'_j would dominate strategy σ_j, contradicting the assumption that the original $\vec{\sigma}$ is Nash.

Linger Avoiding Assumption: In this chapter, we make the standard *linger avoiding assumption*. This means that our mechanisms do not require players to continue after they learn the secret. Without this assumption, there are trivial but fragile mechanisms that are Nash equilibria. See [KN08] for details.

3.6.3 Information-Theoretic Message Authentication Codes

A Message Authentication Code (MAC) scheme consists of three algorithms Key-Generation, Mac and Verification. The key-generation algorithm generates a key as its name suggests. The Mac algorithm takes a key and a message from a field as input and outputs a tag. The Verification algorithm takes a key, a message and a tag as input, and verifies the tag. In our algorithm we use the following information theoretic MAC scheme in field $\mathbb{F}$ with q elements. The Key-Generation algorithm outputs $c \in \mathbb{F}$ and $b \in \mathbb{F}^* = \mathbb{F} \setminus \{0\}$ independently, uniformly at random. (b,c) is the verification vector, to be given to the recipient of the message y. The Mac algorithm takes an $\mathbb{F}$-element message y and generates the tag, $a = c - b \cdot y$, which will be given to the sender of message y to send it along with the message. The recipient of the message y can run the Verification algorithm to verify the tag by testing if $c - b \cdot y$ is equal to a.

Next, we discuss the properties of the above information theoretic MAC scheme. This scheme makes it hard for a sender to successfully fool the intended recipient of a message by sending a faked message. At the same time, it does not give the recipient of the message any information about

the message prior to receiving it. Such schemes have been used before (see *e.g.* [WC81, RBO89, KN08]); we include the following proposition for completeness. See Lemma 1 of [RBO89] for the proof of the following proposition.

Proposition 3.6.1. The MAC scheme described above has the following properties:

- The verification vector contains no information about the message, *i.e.* , the probability of correctly guessing the message given the verification vector is the same as the unconditional probability of guessing the message.
- The probability that a faked message will satisfy the verification function is $\frac{1}{q-1}$.

3.7 Algorithm for n-out-of-n Secret Sharing

We now describe our scalable mechanism for n-out-of-n secret sharing. First, in Section 3.7.1 and 3.7.2 we describe the communication tree and how to label it that is used by the dealer and players. An informal description of the mechanism follows in Section 3.7.3. The formal descriptions of the dealer's and players' protocols appear respectively as Algorithms 19 and 20.

3.7.1 The Communication Tree

Communication between the players in our protocol will be restricted to sending messages to their neighbors in a *communication tree*. The communication tree is a complete binary tree with n leaves. Recall that a complete binary tree is a binary tree in which all the internal nodes have exactly two descendants, all the leaves are at the two deepest levels, and the leaves on the deepest level are as far left as possible.

The communication tree is used to build *iterated shares* of a value. The dealer sends these iterated shares to the players, so they can reconstruct the value later. The iterated shares players receive are constructed by starting with the value to be reconstructed at the root and recursively constructing 2-out-of-2 Shamir shares down the tree, all the way down to the leaves. The shares at the leaves are the iterated shares the players receive. See Figure 3.1 and Algorithm 18 for details of how the iterated shares are constructed. At reconstruction time, shares are sent up the tree. At each internal node, a pair of shares received from the two children is reconstructed into a degree 1 polynomial which is used to obtain the value to be sent further up the tree. At the root, the original symbol is reconstructed and transmitted down the tree. Note that the advantage of this scheme over simply using n-out-of-n Shamir shares is that the size of the messages does not increase as the

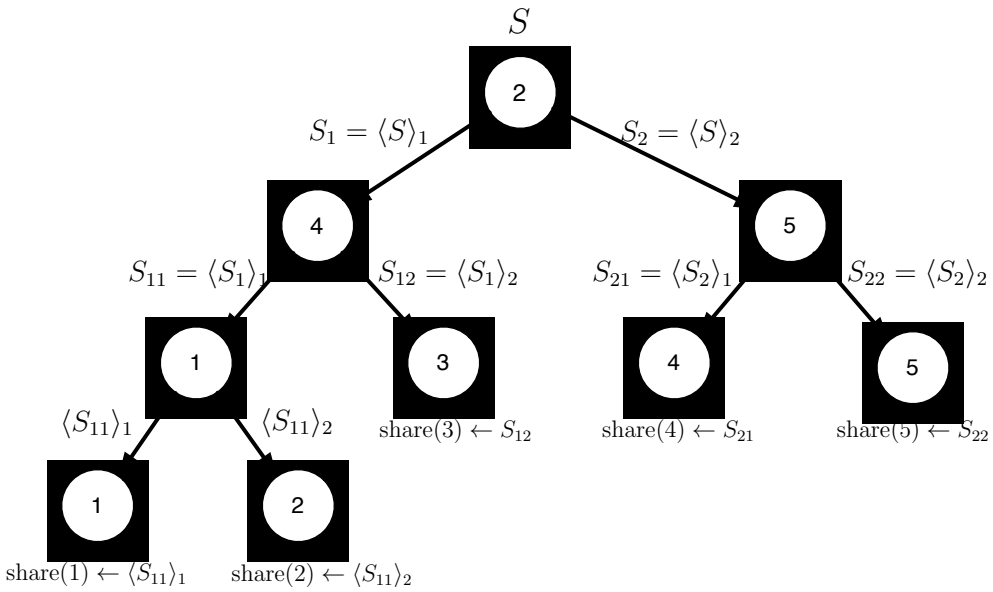

Figure 3.1: Construction of the iterated shares. We define $\langle S \rangle_1$ as the first share and $\langle S \rangle_2$ as the second share generated from the 2-out-of-2 Shamir's secret sharing scheme.

messages are transmitted up the tree. Lemma 3.7.1 describes that shares resulted from Algorithm 18 is an n-out-of-n secret sharing scheme.

Lemma 3.7.1. Let $\sigma \in \mathbb{F}$ be a value that is encoded into n iterated shares, $\sigma_1,\ldots,\sigma_n$, by Algorithm 18. Then, σ can be decoded from all n of the shares, but knowledge of fewer than n of the shares reveals no information about σ.

Proof. Starting with the shares $\sigma_1,\ldots,\sigma_n$ at the leaves of the tree, we can recursively reconstruct the values bottom-up using 2-out-of-2 Shamir's secret sharing reconstruction algorithm. Since this is exactly the reverse of the process used to create these shares, the value at the root will be σ.

To see why fewer than n shares give us no information about σ, observe that the values at the two children of the root were created by 2-out-of-2 secret sharing algorithm. Both of these values together determine the secret, but a single one of them does not reveal any information about the secret. Thus the values at the children of the root *individually* contain no information about the value of the root, and in order to decode the value at the root, we need both the values at its children. But now, this reasoning applies recursively to all the internal nodes, relative to their children. Suppose there is a leaf of the tree at which the share is missing. Then, the share of its parent cannot be decoded because it is equally likely to be any element of the $\mathbb{F}$. This propagates up to its grandparent, and then its great-grandparent and so on all the way to the root, so that the

Protocol 18 *RecursiveShares* (node w , $\mathbb{F}$-element y):

Parameters: V is a n-leaf complete binary tree global data structure; for node w, $V(w)$ denotes the location for the data associated with w.
Usage: Initially called with the root node and the value for which shares are to be created, this function populates V with intermediate values. The values at the leaves are the shares for the players at the corresponding leaves of the communication tree.

1. $V(w) \leftarrow y$.
2. If w has children $L(w)$ and $R(w)$:
   a) share y using 2-out-of-2 secret sharing. Let the shares be y_L, y_R
   b) *RecursiveShares*$(L(w), y_L)$.
   c) *RecursiveShares*$(R(w), y_R)$.

root cannot be decoded. Thus, if even one of the shares is missing, the remaining shares provide no information about the value of σ. □

3.7.2 Labeling the Communication Tree: Short and Long Players

Recall that the short players are the players who are parents of the leaves in the communication tree in the definitive round and all other players are long players. To make the definition easier, we devise a specific labeling for the communication tree. We have exactly n labels and thus, the labels may be repeated in the tree. The leaves will be labeled 1 to n from left to right. Next every internal node which is a parent of two leaves is labeled with the odd label from among its two children. Finally, the remaining internal nodes are labeled in order with even numbers, proceeding top to bottom and left to right, starting with 2 at the root. If n is odd, then each even number appears at some internal node. If n is even, we will place the last even number, n at the root, along with 2, so the root will have two labels. Figure **??** illustrates the labeling scheme for five and six players. The tree thus labeled has the following properties:

1. Every even label occurs at some internal node. If n is odd, there will be an odd label that occurs only at a leaf and not at any internal node. This will not matter.
2. No even labeled internal node has an odd labeled node above it.
3. Every path from root to leaf has exactly one odd label (the same odd label may occur once or twice on the path).

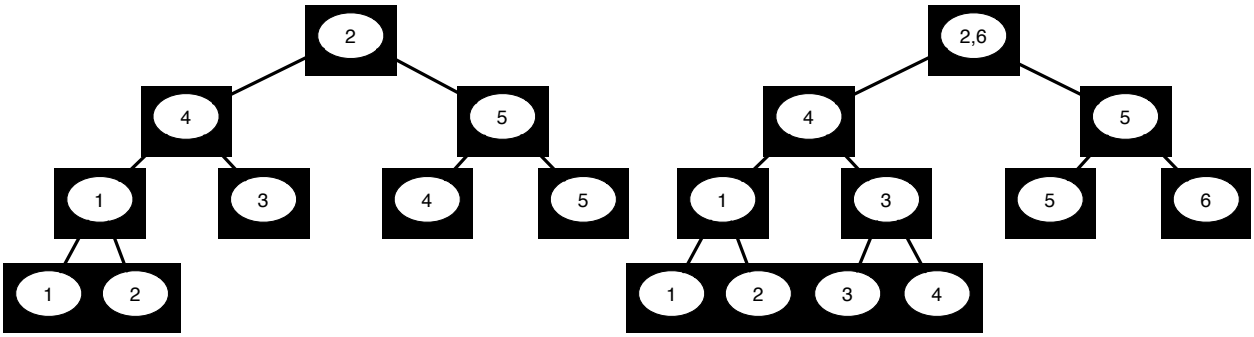

Figure 3.2: Communication trees for 5 players (left) and 6 players (right)

We generate a permutation of players and assign players to n labels based on his permutation position. We use this permutation to assign players to the nodes of the tree. A player is assigned to a node if they have the same labels. Thus, each player can be assigned to multiple nodes of the tree and two players may assign to the root node of the tree. Based on this labeling, the short players are ones in odd positions in the permutation in the definitive round and all the other players are long players [1].

As discussed in Sections 3.7.4 and 3.8, it is critical in our analysis that the players themselves do not know which are short or long until the definitive round. Further, each player must have as a priori equal chance to be short or long. Thus, the permutation and the definitive round must be random and unknown to the players. Therefore, in our protocol, the dealer first generates random trees based on random permutations and chooses a definitive round randomly. Then, he assigns the long and short players accordingly to the choice of the tree in the definitive round.

3.7.3 Our Algorithm

The dealer is active only once at the beginning of the game, and during this phase of the game the players' inputs are prepared.

The dealer independently samples two random variables X and Y from a geometric distribution with parameter β (to be determined later). X will be the definitive iteration, or the round of the game in which the true secret is revealed. Y will be the amount of padding on the long players' input. We have two kinds of players: short players will receive enough input to last for X rounds of the game while long players will receive enough input to last for $X+Y$ rounds of the game.

[1] Note that the above numbered properties for odd and even labeled nodes in the tree are also correct for short and long players in the definitive round.

Protocol 19 Dealer's Protocol

Parameters: $\mathbb{F}$ is a field of size $q > n$ (to represent messages in the algorithm) n players with distinct identifiers $1,2,\ldots n \in \mathbb{F}$. $\beta \in (0,1)$ is geometric distribution parameter. Complete binary tree with n leaves, labeled as described in Section 3.7.1 known to everyone.

1. Choose X,Y, independently from a geometric distribution with parameter β and let $L = X + Y$. Round X is the definitive one. Short players will receive full input for $X-1$ rounds and partial input for round X. Long players will receive full input for $L-1$ rounds and partial input for round L. For convenience we will create all the inputs for L rounds, and truncate them appropriately before sending them to the players.

2. for round $r = 1$ set $m_1 = 0$

3. For each round r between 1 and L:

   a) If $r < L$, then choose a random permutation $\pi_r \in S_n$. Else, randomly choose a permutation π_L which is random subject to the constraint that all the long players (determined by π_X) are assigned to odd labels under π_L.

   b) *Preparing labeling information:*
   For all players j

      i. Set player j's label to $\pi_r(j)$. player j will be assigned to all nodes marked $\pi_r(j)$ in the tree.

      ii. Let $\nu_r(j)$ be the set of all neighbors of nodes labeled $\pi_r(j)$ in the tree. $\pi_r(j)$ has a constant number of neighbors.

      iii. Let $\pi_r^{-1}(\nu_r(j))$ be the tuple of identities of all the players assigned to a node in set $\nu_r(j)$. Thus, it is the tuple of identities of all neighbors of j.

      iv. Generate the labeling information tuple, $(\pi_r(j),\pi_r^{-1}(\nu_r(j)))$.

      v. Mask the labeling information tuple,
      $P_r^j = (\pi_r(j),\pi_r^{-1}(\nu_r(j))) + m_r$.

   c) *Preparing shares of the secret:*

      i. If $r = X$, then $s_r \leftarrow$ true secret.
      Else, $s_r \leftarrow$ random element of $\mathcal{S}$

      ii. Create shares of s_r by calling *RecursiveShares* (root,s_r).

   d) *Preparing mask shares for the next round:*

      i. Choose a random mask $m_{r+1} \in \mathbb{F}^c$.

      ii. Create shares of m_{r+1} by calling *RecursiveShares* (root,m_{r+1}).

   e) Create tags and verification functions for all the messages to be sent in round r

   f) For each player j, j's (full) input I_r^j for round r consists of P_r^j, shares of m_{r+1} and s_r corresponding to node $\pi_r(j)$, tags to authenticate all messages to be sent by j and verification vectors for all the messages to be received by j. Partial input $\tilde{I}_r^j$ consists of all of the above except the authentication tags for sending messages to your children (in the down-stage).

4. Identify the short players as those players j who are at odd numbered nodes in the definitive

Protocol 20 Protocol for Player j

Parameters: S=0; M=0
If at any time you receive spurious messages (messages not expected under the protocol), ignore them.
On round r:

Up-Stage:

1. $m_r = M$
2. Subtract m_r from P_r^j to find out your position in the tree and the identities of your neighbors for round r.
3. (as a player at a leaf) Send your shares of s_r and m_{r+1} along with their tags to your parent in the tree.
4. (as a player at an internal node)
   a) Receive (intermediate) shares of s_r and m_{r+1} and tags from left and right chidren. Use the appropriate verification vectors to check that correct messages have been sent. If a fault is detected (missing or incorrect message) output S and quit.
   b) For each of s_r and m_{r+1}: interpolate them from their shares. This is your share.
   c) If you are not at the root, send the above reconstructed shares of s_r and m_{r+1} to your parent(s) along with the appropriate tags. If you *are* at the root, these shares are the actual values of s_r and m_{r+1}.

Down-Stage:

1. (as a player at the root) Set $S = s_r$ and $M = m_{r+1}$ and send these values along with authentification tags to your left and right children.
2. (as a player at a non-root node)
   a) Receive s_r and m_{r+1} and tags from your parent and use verificaton vectors to check them. If fault detected, output S and quit.
   b) Set $S = s_r$ and $M = m_{r+1}$.
3. (as a player at a non-root internal node) Send s_r and m_{r+1} to your children along with the appropriate tags. If you are a short player and have no authentication tags, output s_r and quit.

$r \leftarrow r + 1$

In order not to reveal which players are the short players, the players will be reassigned to new random positions in the tree in each round. This is accomplished by choosing a random permutation of the players and label them for each round. Again recall that the short players are the ones who are at odd labeled nodes in the definitive round.

Since the players must be at different nodes in the tree each round, their input must contain this information. At the same time, the positions of the players for all the rounds cannot be revealed up front, since this may give away information about who the short players are. A naive idea to solve this problem is, in each round, to distribute shares of the permutation for the next round. Then, during each round, the players could reconstruct the permutation from the shares and use it to reposition for the next round. Unfortunately, there is a problem with this approach. To represent permutations of n symbols, we need a field of size at least $n!$. To transmit elements of such a field, players would need to send messages of length $\log(n!) \sim n \log n$. This is unacceptable if we desire scalability.

To get around this problem, we note that it is not really necessary for players to know the entire permutation. Each player only needs to know its own position and the identities of its neighbors. We only need a field of size order n to encode this, and so, symbols of this field may be transmitted with messages of length $O(\log n)$. Since it is dificult via share reconstruction to transmit different messages to the leaves of the tree, we simply provide each player with a list of positional data for the entire game. But in order that players do not know their positional data for a round before actually getting to that round, this data is masked by adding in a random element of the field. Positional data for the first round is sent unmasked. The players also receive iterated shares of the masks for the next round. Thus, in each round, players reconstruct a mask, and use it to unmask the positional data and reposition themselves for the next round.

For each round, the full input consists of the following:

- iterated shares of a purported secret (the true secret in the definitive round);
- masked versions of positional data for the current round (position and identities of neighbors in the tree);
- shares of the mask for the *next round* of positional data;
- tags for all the messages to be sent; and
- verification vectors for all the messages to be received.

As mentioned earlier, round X is the definitive round, when the encoded symbol is the true secret. Short players receive full input for every round prior to this round. For round X they only

receive partial input. Long players receive input for $X + Y$ rounds. However, they, too receive only partial input for their last block of input. Otherwise, a player would be able to distinguish whether or not he is a short player by looking at his last block of input. Here, partial input consists of all of the pieces of data from the full input, *except* the tags to send the decoded message to your children in the down stage of the round.

Since in the definitive round the short players (with odd labels) are in the level above the leaves, and all the long players are at internal nodes higher than that in the tree, the long players have learned the secret before the short players, although since they have input for more rounds of the game, they do not know (*i.e.* , cannot guess) that it is the definitive round, and that the secret they have learned is in fact the true secret. Thus, they send the secret down the tree, and eventually it gets to the short players. Thus, the short players learn the secret as well. Since they have no more input they know that the game is over and the secret is the true one. However, since they do not have any more authentication data, they cannot gain by remaining in the game and trying to fool the others into thinking that the secret has not yet been reconstructed. Finally, when the long players do not receive a message at the end of the definitive iteration, they too realize the game has ended and output the correct secret.

3.7.4 Discussion

A careful reader may ask why it is necessary to permute the players. First, we note that if we never permute the players, then the short and long players know who they are at the beginning of the game. In the game, there is an event with small probability that the short players have an array of size exactly one less than the long players (e.g. short players have array of size one and long players have array of size two). If there are only two rounds remaining in the list of long player, he learns the one to last (the current) round is the definitive round since Y is at least 1. It is crucial for our analysis that long players do not know if the current round is the definitive round. Since, in this case, the long player at the root node can learn the secret without revealing it to any other players.

What if the players are only permuted in the definitive round? Then, when the players see that a permutation occurs, they know they are in the definitive round. Thus, again the long player(s) at the root node will be able to learn the secret without allowing other players to learn it. Hence, we need more than one random permutation during the algorithm. For simplicity of algorithmic presentation and analysis, we permute the players at every round.

A careful reader might wonder why we do not first construct a new scalable protocol for rational broadcast and then use it in the existing protocols for RSS. They are two main reasons. First of all, we note that composing two mechanisms that are Nash equilibria does not necessarily give a new

mechanism that is a Nash equilibrium. In general, composability of mechanisms and generating a new one with specific properties is not an easy problem. Second, it seems difficult to come up a mechanism that can broadcast in a scalable manner, because of the linger avoiding assumption we make. In particular, it seems that all players must participate in the broadcast and learn the result of the broadcast at the same time. We achieve scalability by never performing broadcast.

3.8 Analysis of Algorithm for All Players Present

In this section, we show that the secret sharing scheme we have described is in fact a scalable n-out-of-n secret sharing scheme, and that it is a Nash equilibrium in which all the players learn the secret.

Recall that in Lemma 3.7.1 we show our rescursive scheme for encoding a value into n iterated shares (Algorithm 18) is an n-out-of-n scheme.

We will focus our attention on showing that it is a Nash equilibrium for all the players to follow our protocol. Consider player j and suppose that all other players are committed to following the protocol. The next lemma gives a necessary criterion for j to have an incentive to cheat.

Lemma 3.8.1. If all other players are following the protocol, player j prefers to also follow the protocol, unless his probability of successfully cheating is at least $\frac{U_j - U_j^-}{U_j^+ - U_j^-}$.

Proof. Suppose j is considering deviating from the protocol. We will consider the deviation to be successful if either j learns the secret right away, with or without being caught, or he does not get caught and is therefore still in a position to learn it later. The deviation will have failed if it is detected, causing the game to end without j learning the secret. Let p_j be the probability that the deviation succeeds. The maximum utility that j can get is U_j^+. With probability $1 - p_j$, the game ends without j learning the secret, in which case the maximum payoff possible is U_j^-. Thus, a player's expected utility from cheating while everyone else follows the protocol is at most $p_j U_j^+ + (1 - p_j)U_j^-$.

On the other hand, if everyone else follows the protocol, then following the protocol guarantees a utility of at least U_j. Thus, the protocol will be a Nash equilibrium if $U_j > p_j U_j^+ + (1 - p_j)U_j^-$. By rearranging the terms, we have a Nash equilibrium if

$$p_j < \frac{U_j - U_j^-}{U_j^+ - U_j^-}.$$

□

When and how might a player cheat? We note that since players are not required to commit to their strategy before starting the game, and since the progression of the game reveals information, a player may as well defer his decision to cheat in a future round until that future round. Thus, at any given time, the decision facing the player is whether to cheat in the current round. In order to weigh the benefits of such a decision, the player needs an estimate of whether the current round is likely to be the definitive one.

As remarked earlier, the purpose of having short and long players is to create uncertainty about when the definitive round of the game is, until it is too late to gain from this information.

The players know that X is chosen from a geometric distribution with parameter β. Thus, *a priori* the probability that X takes on any particular value is at most β, the most likely being $X = 1$, whose probability is exactly β. As the game progresses, players receive partial information about the value of X; as soon as they receive their inputs they can eliminate all values of X larger than their input length, if the game did not end on the first round, they learn that $X \neq 1$ and so on. Clearly, when a player reaches his last block of input, he knows that the current round is definitive. The next lemma shows that until that stage, a player's estimate that the current round is definitive remains small.

Lemma 3.8.2. Let j be a player who initially received input for $k > 1$ rounds of the game, and let $1 \leq r < k$ be the current round. Then, j's estimate of the probability that the current round is definitive, conditioned on all the information he has learned, is at most 2β.

Proof. Let L_j denote the random variable which is the initial input length of player j. Then, we know that

$$L_j = \begin{cases} X & \text{if } j \text{ is a short player,} \\ X + Y & \text{if } j \text{ is a long player.} \end{cases} \tag{3.2}$$

Also, let $\mathcal{L}_j$ denote the event that j is a long player and $\mathcal{L}_j^{\mathrm{C}}$ the event that j is a short player. By hypothesis, the current round is $r \geq 1$, and player j received an initial input of length $k > r$. What information does player j know in round r?

- Since his initial input was of length k he knows that $L_j = k$ and $X \leq k$ and moreover, that $X = k$ if and only if $\mathcal{L}_j^{\mathrm{C}}$.
- Since the game has entered round r he knows that $X \geq r$.
- He knows his position $\pi_r(j)$ and the identities of his neighbors in round r.
- He also has learned s_r, m_{r+1} and using the latter to unmask his positional data, he knows $\pi_{r+1}(j)$ and the identities of his neighbors in round $r + 1$. Technically, he learns these just

prior to his turn in the downstage in round r, but this is fine, as we will argue later that no player ever has any reason to cheat during the upstage of a round.

We note that knowing s_r does not benefit player j in any way as far as estimating the probability that $X = r$ goes, since s_r is equally likely to be any element of $\mathcal{S}$, independently of X. Similarly, knowing the identities of his neighbors does not affect his estimate, since all other players are equally likely to be his neighbors independently of X.

On the other hand, knowing $\pi_r(j)$ and $\pi_{r+i}(j)$ does affect the estimate. By construction:

- In the definitive iteration, short players have odd labels, and long players have even labels; and
- Each player has an odd label in his last round of input.

Thus, if $\pi_r(j)$ is odd, then player j knows that the current round is not definitive, since if $X = r$, then $k > r$ implies that j is a long player and should have an even label. In particular, conditioned on everything he knows, $\Pr(X = r) = 0$. Since $2\beta > 0$ the lemma is proved in this case.

For the remainder of the proof, we will assume that $\pi_r(j)$ is even and denote this event $\mathcal{E}_r$. Consider the case $\pi_{r+1}(j)$. If $k = r+1$, then we know that $\pi_{r+1}(j)$ is odd by construction and knowing this contains no additional information over knowing $L_j = k$. On the other hand, when $k > r+1$, if $\pi_{r+1}(j)$ is odd, then player j knows that X cannot be $r+1$ and this affects the probability that $X = r$.

Let $b \in \{0,1\}$ be the observed parity of $\pi_{r+1}(j)$ and let $\mathcal{E}^b_{r+1}$ denote the event that the parity of $\pi_{r+1}(j)$ is b. Note that if $k = r+1$ we must have $b = 1$.

Let $P_{j,r}$ be player j's estimate that the current round, r, is definitive, conditioned on everything he knows. Then,

$$\begin{aligned} P_{j,r} & \\ &= \Pr(X = r \mid L_j = k \wedge r \le X \le k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1}) \\ &= \frac{\Pr(X = r \wedge L_j = k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1})}{\Pr(L_j = k \wedge t \le X \le k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1})} \\ &\le \frac{\Pr(X = r \wedge L_j = k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1})}{\Pr(L_j = k \wedge X \in \{r,k\} \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1})}, \end{aligned}$$

where the inequality follows from the fact that the event $X \in \{r,k\}$ is a subset of the event $r \le X \le k$.

If the current round *is* definitive *i.e.*, $X = r$, then j is not a short player, and $L_j = X + Y$. So the event $X = r \wedge L_j = k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1}$ is the same as the event $\mathcal{L}_j \wedge X = r \wedge Y = k - r \wedge \mathcal{E}^b_{r+1}$. Note that $\mathcal{E}_r$ is implied by $\mathcal{L}_j \wedge X = r$ and can therefore be dropped.

For the denominator, the event $L_j = k \wedge X \in \{r,k\} \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1}$ can be split into the union of disjoint events $\mathcal{L}^{\mathrm{C}}_j \wedge X = k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1}$ and $\mathcal{L}_j \wedge X = r \wedge Y = k-r \wedge \mathcal{E}^b_{r+1}$. The latter summand is the same as the numerator, and loses the $\mathcal{E}_r$ term for the same reason. Making these substitutions in the above expression, we get

$$P_{j,r} \leq A/B,$$

where

$$A = \Pr(\mathcal{L}_j \wedge X = r \wedge Y = k-r \wedge \mathcal{E}^b_{r+1})$$

and

$$\begin{aligned} B = {} & \Pr(\mathcal{L}^{\mathrm{C}}_j \wedge X = k \wedge \mathcal{E}_r \wedge \mathcal{E}^b_{r+1}) \\ & + \Pr(\mathcal{L}_j \wedge X = t \wedge Y = k-t \wedge \mathcal{E}^b_{r+1}). \end{aligned}$$

The random variables X, Y and the indicator that j is a long player are independent. Thus the numerator of the above expression becomes

$$\begin{aligned} & \Pr(\mathcal{L}_j \wedge X = r \wedge Y = k-r \wedge \mathcal{E}^b_{r+1}) \\ & \quad = \Pr(\mathcal{E}^b_{r+1} | \mathcal{L}_j \wedge X = r \wedge Y = k-r) \times \\ & \qquad \Pr(\mathcal{L}_j) \Pr(X = r) \Pr(Y = k-r) \\ & \quad = \Pr(\mathcal{E}^b_{r+1} | \mathcal{L}_j \wedge X = r \wedge Y = k-r) \times \\ & \qquad \frac{\lfloor n/2 \rfloor}{n} (1-\beta)^{r-1} \beta (1-\beta)^{k-r-1} \beta \\ & \quad = \Pr(\mathcal{E}^b_{r+1} | \mathcal{L}_j \wedge X = r \wedge Y = k-r) \times \\ & \qquad \frac{\lfloor n/2 \rfloor}{n} (1-\beta)^{k-2} \beta^2. \end{aligned}$$

Similarly we tackle the first term in the denominator. Since X and all the π_i are independent and $k \neq r$, then π_r and π_X are independent conditioned on $X = k$. It follows that the events $\mathcal{L}^{\mathrm{C}}_j = \pi_X(j)$

is odd; $\mathcal{E}_r = \pi_r(j)$ is even; and $X = r$ are independent. Thus, we have the following.

$$\begin{aligned}
&\Pr(\mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r \wedge \mathcal{E}_{r+1}^b) \\
&\quad = \Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r) \times \\
&\qquad \Pr(\mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r) \\
&\quad = \Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r) \times \\
&\qquad \Pr(\mathcal{L}_j^{\mathrm{C}}) \Pr(\mathcal{E}_r) \Pr(X = k) \\
&\quad = \Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r) \times \\
&\qquad \frac{\lceil n/2 \rceil \lfloor n/2 \rfloor}{n^2} (1-\beta)^{k-1} \beta \\
&\quad \geq \Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r) \times \\
&\qquad \frac{\lfloor n/2 \rfloor}{2n} (1-\beta)^{k-1} \beta.
\end{aligned}$$

If $k > r+1$ then π_X and π_{r+1} are independent conditioned on X being either r or k, and hence, the events $\mathcal{E}_{r+1}^b$ and $\mathcal{L}_j \wedge X = r \wedge Y = k - r$ are independent, as are the events $\mathcal{E}_{r+1}^b$ and $\mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r$. It follows that $\Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j \wedge X = r \wedge Y = k - r)$ and $\Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r)$ both equal $\Pr(\mathcal{E}_{r+1}^b)$. On the other hand if $k = r+1$ then $b = 1$ and $\mathcal{E}_{r+1}^b$ is implied in both cases Thus, $\Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j \wedge X = r \wedge Y = k - r)$ and $\Pr(\mathcal{E}_{r+1}^b | \mathcal{L}_j^{\mathrm{C}} \wedge X = k \wedge \mathcal{E}_r)$ are both 1. Either way, they are equal, and since their common value occurs in the numerator as well as in both terms in the denominator, it simply cancels out.

Putting everything together we see that

$$\begin{aligned}
P_{j,r} &\leq \frac{\frac{\lfloor n/2 \rfloor}{n} (1-\beta)^{k-2} \beta^2}{\frac{\lfloor n/2 \rfloor}{2n} (1-\beta)^{k-1} \beta + \frac{\lfloor n/2 \rfloor}{n} (1-\beta)^{k-2} \beta^2} \\
&= \frac{\beta}{\frac{1}{2}(1-\beta) + \beta} \\
&\leq 2\beta.
\end{aligned}$$

□

We remark that although in the above proof we have bounded player j's estimate during the down-stage, a nearly identical proof shows the same bound for the up-stage (when $\pi_{r+1}(j)$ is unknown). We are now ready to prove the main theorem.

of Theorem 3.3.1. We will begin by showing that the protocol is a Nash equlibrium in which all the players learn the secret.

Suppose all the players follow the protocol. In every round, during the up-stage, players send their shares up toward the root where they are decoded. During the down-stage, the reconstructed secret and mask are sent back toward the leaves. If the odd players in the round do not drop out at the end of the round, then the game continues into the next round. In the definitive round, the real secret is reconstructed at the root and all the even labeled players, who are long players learn it first. Once it gets to the short players with odd labels, they drop out of the game since they have no tags to send any more messages. This signals the end of the game to the long players who then realize that the current reconstructed secret is the true one. Thus if all the players follow the protocol, everyone learns the secret.

Suppose all players other than j are following the protocol. We want to show that player j prefers following the protocol over deviating.

At the beginning of the game, each player has set their current guess for the secret to 0. If no cheating occurred before the current round r then during round $r-1$ all the players set their current guess to s_{r-1}. Thus, at the beginning of round r, all players have the same guess for the secret. (Round $r-1$ has been eliminated as the definitive one, but s_{r-1} still has probability $1/|\mathcal{S}|$ of being the true secret). Moreover, since by Lemma 3.7.1 partial information about the shares reveals no information about the symbol they encode, throughout the up-stage player j has no better guess than s_{r-1} for the secret. Since $\mathcal{U} < |\mathcal{S}|$, it is strictly better for j not to leave the game in the up-stage. If j sends incorrect messages in the up-stage, then even if he is not caught(which results in not learning the secret), this deviation will cause an incorrect value to be decoded instead of s_r. This results in j not learning the secret if r happened to be the definitive round. Thus, j has no incentive to deviate in the up-stage.

Now, what about the down-stage? If j is on his last round of input, then he is a short player, and knows that the current round is definitive. At the same time, this means that he is an odd-labeled player and by the construction of the communication tree he cannot prevent anyone from learning the secret. Moreover, even if he successfully fakes a tag in order to convince the unique long player below him that the game has not ended, that player will detect in the next round that all other players have left the game and will therefore still output the correct secret. Thus this deviation does not change the outcome of the game, namely that all players learn the secret. It follows that j does not gain anything by this deviation (although he also does not lose anything by it). Effectively, if j is a short player on his last round of input, it is too late for him to improve his payoff by deviating.

If j is not on his last round of the input and is at an odd labeled node, then, as remarked in the proof of Lemma 3.8.2, he *knows* the current round is not definitive. So, cheating would be equivalent to randomly guessing the secret that is correct with probability only $1/|S|$. This is worse

than following the protocol by Equation (3.1).

Now, suppose that the current round, r, is not j's last round of the input and j is at an even labeled node. Note that any spurious messages sent by player j to players that are not expecting them, will be ignored. Also, any action involving not sending a message that is expected will be detected immediately, only by the involved players at first, but the knowledge will quickly propagate to all the players, before the end of the up-stage of the next round. Since detection of deviation causes other players to quit immediately, effectively such actions amount to player j leaving the game. Thus, the possible deviations we need to analyze for player j are:

- Leave the game with or without sending fake messages first, and;
- Send fake messages to one or both children and hope to stay in the game by not being caught.

Let $P_{j,r}$ be j's estimate of the probability that the current round is definitive. Then, by Lemma 3.8.2, $P_{j,r} \leq 2\beta$.

If player j leaves the game and outputs the value s_r then the probability that he has output the right value is $P_{j,r} + (1 - P_{j,r})/|\mathcal{S}|$, and since he has left the game, he has no later opportunity to improve that probability. By Lemma 3.8.1, in order to discourage this deviation it is sufficient if

$$P_{j,r} + \frac{(1 - P_{j,r})}{|\mathcal{S}|} \leq \frac{U_j - U_j^-}{U_j^+ - U_j^-}. \tag{3.3}$$

Now consider the other kind of deviation. Suppose instead of sending s_r to his descendents, player j sends a fake value to one or both of them. Let α be the probability that he is not caught. By Proposition 3.6.1 we know that $\alpha = \frac{1}{q-1}$ if he sends a fake message to only one of his children, and, since the two verification functions are chosen independently by the dealer, $\alpha = \frac{1}{(q-1)^2}$ if he fakes both messages. If he is not caught, and if the current round is definitive, then player j has learned the true secret and has prevented some of his descendants from learning it. If the current round was not definitive and his deviation was not detected, the game continues and since the values s_i are all independent, it does not affect the next round.[1] This means that player j can either revert to following the protocol and guarantee learning the secret along with everyone else, or he may find further opportunities to cheat.

On the other hand, if the faked message is detected, which happens with probability $1 - \alpha$, then the game ends right away and player j outputs s_r. In this case, there is still a $P_{j,r}$ chance that the current round was definitive and an additional $(1 - P_{j,r})/|\mathcal{S}|$ chance that the value s_r was correct

[1] This is why player j does not try to fake the mask m_{r+1} - a successfully transmitted incorrect m_{r+1} will wreak havoc in the next round, since some players will be talking to the wrong players.

despite the current round not having been definitive. So the probability that the deviation succeeds is

$$\alpha + (1-\alpha)\left(P_{j,r} + \frac{(1-P_{j,r})}{|\mathcal{S}|}\right).$$

As this quantity is bigger when $\alpha = \frac{1}{q-1}$, faking only one message dominates faking both messages. Thus, again by Lemma 3.8.1, to discourage this deviation, it is sufficient if

$$\frac{1}{q-1} + \frac{q-2}{q-1}\left(P_{j,r} + \frac{(1-P_{j,r})}{|\mathcal{S}|}\right) \leq \frac{U_j - U_j^-}{U_j^+ - U_j^-}. \tag{3.4}$$

Moreover, note that (3.4) implies (3.3). Thus for a Nash equilibrium, it is sufficient to show (3.4).

For the remainder of the proof, we are going to assume that n is sufficiently large, specifically that $n \geq \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$. We will discuss the modifications required when $3 \leq n < \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$ in Section 3.8.1.1.

We have so far not specified β. We do this now. Let

$$\beta = \frac{|\mathcal{S}| - \mathcal{U}}{4\mathcal{U}|\mathcal{S}|}.$$

Then, $1/\beta = O(1)$ so that the expected number of rounds in the game is constant.

To show (3.4), recall that $q > n$, so that $q - 1 \geq n$. We have

$$\begin{aligned}
&\frac{1}{q-1} + \frac{q-2}{q-1}\left(P_{j,r} + \frac{(1-P_{j,r})}{|\mathcal{S}|}\right) \\
&\quad< \frac{1}{n} + \left(P_{j,r} + \frac{(1-P_{j,r})}{|\mathcal{S}|}\right) \\
&\quad= \frac{1}{n} + \frac{|\mathcal{S}|-1}{|\mathcal{S}|}P_{j,r} + \frac{1}{|\mathcal{S}|} \\
&\quad\leq \frac{|\mathcal{S}|-\mathcal{U}}{2\mathcal{U}|\mathcal{S}|} + 2\beta + \frac{1}{|\mathcal{S}|} \\
&\quad\leq \frac{|\mathcal{S}|-\mathcal{U}}{2\mathcal{U}|\mathcal{S}|} + 2\frac{|\mathcal{S}|-\mathcal{U}}{4\mathcal{U}|\mathcal{S}|} + \frac{1}{|\mathcal{S}|} \\
&\quad= \frac{|\mathcal{S}|-\mathcal{U}}{\mathcal{U}|\mathcal{S}|} + \frac{1}{|\mathcal{S}|} \\
&\quad= \frac{1}{\mathcal{U}}
\end{aligned}$$

as desired.

Finally, we analyze the resource costs of our protocol. The communication tree has $2n-1$ nodes. In each round, each player is mapped to one leaf and one internal node. Players only communicate with their neighbors in the tree. So on each round, during the up-stage each player sends up to three messages: one to his parent when he is a leaf; one to his parent when he is a non-root internal node;

and if he is a child of the root and n is even, he has to send an additional message because there are two players at the root.

During the down stage, each player sends two messages, to his two children. Thus, each player sends at most five messages per round. Each message consists of four elements of $\mathbb{F}$ (shares of s_r, m_{r+1} and two tags) each of which is represented as $O(\log n)$, since $n < q \leq 2n$. Thus, each player send $O(\log n)$ bits per round. Since the expected number of rounds is $1/\beta$, which is constant, each player sends only $O(\log n)$ bits during the course of the game. Finally, since the tree has depth $O(\log n)$, the number of rounds is constant (in expectation). Note that the communication is synchronous, which follows that the expected latency is $O(\log n)$. □

3.8.1 Some Remarks

3.8.1.1 The Case of a Small Number of Players

When the number of players is a constant greater than 2, then scalability is not an issue, and one might hope to simply use the algorithm of Kol and Naor [KN08] by simulating non-simultaneous broadcast channels with secure private channels. Unfortunately their algorithm only provides an ϵ-Nash equilibrium, since the unique short player has a small chance of successfully pretending the game has not ended.

In our algorithm, $\lceil n/2 \rceil$ players are short players. In particular, even for $n = 3$, there are at least two short players, and none of the short players can increase their expected payoff by cheating alone. Thus, we obtain a Nash equilibrium, provided that we can prove inequality (3.4). The above proof used the fact that n was at least $\frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}$, so we need a separate argument.

However, scalability is immediate when n is constant. Thus, we have more leeway to choose a larger field to work with.[1] So, we can work in a prime field of size q where

$$\max\{|\mathcal{S}|, \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}\} < q \leq 2\max\{|\mathcal{S}|, \frac{2\mathcal{U}|\mathcal{S}|}{|\mathcal{S}|-\mathcal{U}}\}$$

and the proof of (3.4) goes through as before, giving us a Nash equilibrium.

3.8.1.2 Nash Equilibria vs. Strict Nash Equilibria

An n-tuple of strategies is called a strict Nash equilibrium if when all other players are following the prescribed strategy, a player unilaterally deviating achieves a strictly worse expected payoff than he would by following the equilibrium strategy.

[1] The upper bound of $O(n)$ on the field size came from the desire to keep $4\log q$, which is the size of an individual message, small.

Our algorithm fails to be a strict Nash equilibrium, for the following reasons:

- Any player may, at any time, send spurious messages that are not part of the protocol, to players that are not his neighbors in the tree. Such messages will be ignored by their recipients, who are following the protocol.
- At the end of the definitive round, a short player may try to fake a tag and send a message to the long player below him. This may go undetected with some small probability, but as noted in the proof, even in this case, it cannot fool that long player into outputting the wrong secret.

Our proof shows that our algorithm *does* have the property that any player deviating from the protocol in one of the above ways does not increase his payoff, and moreover *does not affect any other player's payoff either.* In other words, if a player deviating unilaterally from our protocol, does so in a manner that changes some other player's payoff, then he strictly reduces his own expected payoff. In this weaker sense, the equilibrium is strict.

3.8.1.3 A Note on Backwards Induction

The *backwards induction* problem arises when a multi-round protocol has a last round number that is known to all players. In particular, if it is globally known that the last round of the protocol is ℓ, then on the ℓ-th round, there is no longer any fear or reprisal to persuade a player to follow the protocol. But then if no player follows the protocol in the ℓ-th round, then in the $(\ell-1)$-th round, there is no reason for any player to follow the protocol. This same logic continues backwards to the very first round.

As in [KN08], we protect against backwards induction by having both long and short players. As the above analysis shows, we can ensure that the probability of making a correct guess as to when the protocol ends is too small to enable profitable cheating for any player. Thus, even when a player gets to the second to the last element in all his lists, he can not be very sure that the protocol will end in the next round. All players are aware of these probabilities at the beginning of the protocol, and thus each player knows that no other player will be able to accurately guess when the protocol ends.

The backwards induction problem may occur with protocols that make cryptographic assumptions. If there is a round, ℓ, in which enough time has passed, so that even a computationally bounded player can break the cryptography, then it is globally known that the protocol will end at round ℓ. Thus, even though ℓ may be far off in the future, by backwards induction, there is no incentive for a player to follow the protocol even at the beginning. We note that in out t-out-of-n

protocol, there is no such round since the keys are refreshed every round. Moreover, the information about the key for each round is shared until that round and nobody knows it beforehand.

3.9 Algorithm for t-out-of-n Secret Sharing

In this section, we discuss t-out-of-n secret sharing where $t < n$. Here, we want any subset of t or more of the players to be able to reconstruct the secret. However, fewer than t players should *not* be able to reconstruct the secret on their own.

We will assume, as in previous work, that the set of active players is known to all the active players, before the start of the players' protocol. However, this set is not known to the dealer at the time of share creation.

Our algorithm will be a hybrid of the Kol and Noar [KN08] scheme for t out of n players and our scheme described above for n out of n players. As in [KN08], we now have a single short player and the secret shares now include an indicator bit to enable any subset of t players to reconstruct the secret, even if that subset does not contain the short player. The existence of the short player is necessary to avoid backwards induction. See the work of Kol and Naor [KN08] for details.

Since the algorithm is a variant of the n-out-of-n scheme, for conciseness, we briefly describe the protocol focusing only on the places where the two algorithms differ. We first describe sharing and reconstruction (Sections 3.9.1 and 3.9.2) without authentication, and then show how to modify to ensure authentication in Section 3.9.3.

3.9.1 Creating the Shares

We assume each player has an id that is known to all other players. Moreover, each player initially has a label from 2 to $n+1$ based on their sorted ids. These labels may change during the course of the protocol. In particular, in each round, exactly one player will have its label change to 1, and the player player previously labeled 1 will revert to its initial label. At the start, the dealer selects a random player to be labeled 1 in the first round and sends its id to all players.

To ensure reconstructability, in addition to the secret, the shares will need to encode some more information. Specifically, they will also encode an indicator bit to identify the definitive round, and they will also encode the label of the player that will be labeled 1 for the following round. For a given round r, we let s_r be the potential secret, b_r be the indicator bit, and ℓ_r be the label of the player who will have label 1 in round $r+1$.

The dealer samples values X and Y independently from a geometric distribution with parameter β and lets $L = X + Y$. The game will have L rounds, and round X will be the definitive one.

The player labeled 1 in the definitive round is *the short player*.

For each round $1 \leq r \leq L$, the dealer proceeds as follows. The value ℓ_r is set to a number selected uniformly at random between 2 and $n+1$. Then, if $r = X$, s_r is set to be the true secret, $b_r \leftarrow 1$. Otherwise if $r \neq X$, s_r is set to a random value in the set $\mathcal{S}$ and $b_r \leftarrow 0$.

The dealer prepares the players' inputs as follows. For rounds, $1 \leq r \leq L$, the dealer uses Shamir's (t,n)-secret sharing to generate n different shares of the tuple $\langle r, s_r, b_r, \ell_r \rangle$ and also the hash and tags for each message as described in Section 3.9.3. For rounds $1 \leq r \leq X$, the dealer sends out shares to all the players. For rounds $X+1 \leq r \leq L$, the dealer sends out shares to all players *except* the short player.

3.9.2 Reconstruction phase

For the reconstruction phase, we will make use of the following property of the t-out-of-n Shamir's secret sharing: For every set of shares $s_1, \ldots, s_{t'}$ of cardinality $t' \geq t$, there exist coefficients (Lagrange coefficients) $\lambda_1, \ldots, \lambda_{t'}$, which depend only on the identities of the active players, such that $s = \sum_i^{t'} \lambda_i s_i$. See [Sha79, Knu97] for details.

Let T be the group of t players who are active, and let p_i be the i-th player in T in the sorted order based on their labels. Every active player knows the id and label of all players in T, and the id of the player labeled 1 for the first round. In each round, players generate a complete binary tree with t leaves and assign player p_i to the i-th leaf node, going from left to right. For all $1 < i \leq t$, player p_i is also assigned to an internal node of the tree from root node, going from top to bottom and from left to right.

Each player i at a leaf node multiplies his shares of $\langle r, s_r, b_r, \ell_r \rangle$ by λ_i and sends the result to his parent in the tree. Players at internal nodes of the tree add the received shares together and send the result up. The root node learns the tuple $\langle r, s_r, b_r, \ell_r \rangle$, and sends it down in the tree. Subsequently, each player sends the result down until everybody learns it. If a player does not receive the correct messages from his children or parent, or if he receives a message with indicator 1, he outputs the last secret that was associated with an indicator 0.

3.9.3 MAC Scheme for t-out-of-n

In the t-out-of-n setting, the set of active players, and hence, the position of the players in the reconstruction tree are unknown to the dealer. Thus, unlike the n-out-of-n case, the dealer cannot

generate a tag for the sender and a unique matching verification vector for the potential receiver of the message. Instead, the dealer uses the fact that the same message is propagated in the tree for the down stage. Thus, he uses digital signature to sign messages before sharing them.

Moreover, as in the n-out-of-n case, no player has an incentive to cheat in the up stage of a round, since that results in nobody recovering the secret, including, in particular, the cheater. Thus there is no need to have a verification scheme for the messages in the up stage.

We now formally describe this scheme. For each round r of the protocol, the dealer generates a pair of keys for a digital signature scheme, a secret key SK_r and a public key PK_r. The public key PK_1 is sent to all the players as the first part of their input share. For round $r \geq 1$, after constructing the tuple $\langle r, s_r, b_r, \ell_r \rangle$, the dealer appends it with the private key PK_{r+1}. He then signs the resulting message $\langle r, s_r, b_r, \ell_r, PK_{r+1} \rangle$ using his private key SK_r, and then creates Shamir shares of the resulting *signed* message. These shares are then sent to the players. During the reconstruction phase, the player at the root node recovers the *signed* tuple $\langle r, s_r, b_r, \ell_r, PK_{r+1} \rangle$ and uses the dealer's public key PK_r to verify that it was indeed generated by the dealer, and not forged. If it passes the verification, he forwards the *signed* tuple to his children. Otherwise, he quits the game and outputs the last secret that was associated with an indicator zero. This process of checking and forwarding is repeated by all internal nodes in the tree until all the players have learned the tuple $\langle r, s_r, b_r, \ell_r, PK_{r+1} \rangle$ for round r. Thus, each key pair is used to sign only one message. Based on the security of digital signatures, the probability that a player who does not know the dealer's secret key, can successfully forge a message and make it look like it came from the dealer is negligible. Thus this scheme gives us cryptographic security for our algorithm. We note that, unlike in the n-out-of-n case, we do *not* achieve information theoretic security.

3.9.4 Analysis

In this section, we will prove Theorem 3.3.2.

Lemma 3.9.1. Let j be any player who initially received input for $k > 1$ rounds of the game, and let r be the current round $1 \leq r < k$ such that $r \neq X + 1$. Then, j's estimate of the probability that round r is definitive, conditioned on all the information he has learned, is at most 2β.

Proof. The proof is similar to the proof of the n-out-of-n case. Let L_j denote the random variable which is the initial input length of player j. Then, we know that

$$L_j = \begin{cases} X & \text{if } j \text{ is a short player,} \\ X + Y & \text{if } j \text{ is a long player.} \end{cases} \tag{3.5}$$

Also, let $\mathcal{L}_j$ denote the event that j is a long player and $\mathcal{L}_j^{\mathrm{C}}$ the event that j is a short player. By hypothesis, the current round is $r \geq 1$, and player j received an initial input of length $k > r$. What information does player j know in round r? He knows the following.

- Since his initial input was of length k he knows that $L_j = k$ and $X \leq k$ and moreover, that $X = k$ if and only if $\mathcal{L}_j^{\mathrm{C}}$.
- Since the game has entered round r he knows that $X \geq r - 1$.
- He knows his position and the identities of his neighbors in the tree.
- He also has learned s_{r-1}, s_r, b_{r-1}, b_r. Technically, he learns these just prior to his turn in the downstage in round r, but this is fine, as we argued before, no player ever has any reason to cheat during the upstage of a round.

We note that knowing s_{r-1}, s_r, does not benefit player j in any way as far as estimating the probability that $X = r$ goes, since both s_{r-1}, s_r, are equally likely to be any element of $\mathcal{S}$, independently of X. Similarly, knowing the identities of his neighbors does not affect his estimate, since positions of the players are independent of X.

On the other hand, knowing his label does not affect player j's chance of successful cheating. By construction in the definitive round, the short player has label 1, and long players have other labels larger than 1. However, in the definitive round, the short player already reached its last input, and he knows that this round is the definitive round (without the help of knowing its label).

Moreover, each player thinks that he can have label 1 in his last round of inputs and long players never play until the last round to figure out it is not the case for them. Thus, if the label is 1 but $k > r$, then player j knows that the current round is not definitive, since if $X = r$, then $k > r$ implies that j is a long player and should not have label 1 in definitive round. In particular, conditioned on everything he knows, $\Pr(X = r) = 0$. Since $2\beta > 0$ the lemma is proved in this case.

Similarly, knowing that the values of b_{r-1}, and b_r are both zero (since $r \neq X + 1$) also does not affect the estimate since that just indicates that the game has not reached the definitive round yet. The remainder of the proof is exactly like the proof for n-out-of-n case (see Lemma 3.8.2) and we do not repeat it here. □

Now we can complete the proof for Theorem 3.3.2, which we recall here.

Theorem 3.9.1. Let $n \geq 3$ and $t \leq n$. Assuming the players are computationally bounded to polynomial-time algorithms, there exists a protocol for rational t-out-of-n secret sharing with the following properties.

- The protocol is an everlasting ϵ-Nash equilibrium in which all active players learn the secret, where ϵ can be any arbitrary positive value.
- In expectation, the protocol requires each player to send $O(\log n)$ bits, and has latency $O(\log n)$.
- The protocol is not robust to coalition of any size.

Proof. In this proof, we use case analysis for different values of r ($r < X$, $r = X$ and $r = X + 1$) and different kinds of players (short and long).

- When $r < X$, both short and long players have no incentive to deviate the protocol based on Lemma 3.9.1.
- When $r = X$ and player j is the short player, then in round r player j is in his last round of his input. Hence, player j knows he is in definitive round. However, he is not willing to cheat in the game since he wants to find the secret of this round and he will be the last one who learns it.
- When $r = X$ and player j is a long player, he has no incentive to cheat in this round based on Lemma 3.9.1.
- When $r = X + 1$ and player j is a short player, he does not have enough information to participate in the this round, and the only way he can cheat is to successfully forge the messages for the game.
- When $r = X + 1$ and player j is a long player who learns the last round was definitive based on b_r sooner than others since he is in the root, the only way he can cheat in this round is to successfully forge the messages for the game.

Therefore, the only way a player can cheat is to successfully forge a message by breaking the digital signature scheme. Player j gains the largest utility if he cheats in the definitive round after he learns the secret. The probability that a player can successfully break the digital signature scheme is at most $\frac{1}{2^\kappa}$ where κ is the security parameter for digital signature. Thus, if player j decides to cheat and forge a message at the definitive round, with probability $1/2^\kappa$ he can get at most the maximum utility of U_j^+ and with probability of $1 - 1/2^\kappa$ can get at most the maximum utility of $\mathcal{U}_j$. The player who decides to cheat before the definitive round gets the utility of $Uminus_j$ which is less than U_j. Thus, player j's utility is at most $\frac{U_j^+ + (1-2^\kappa)U_j}{2^\kappa}$. On the other hand, if the protocol runs correctly, he

has utility at least U_j. Therefore, our protocol is an ϵ-Nash where ϵ is

$$\frac{U_j^+ + (1-2^\kappa)U_j}{2^\kappa} - U_j = \frac{U_j^+ + (1-2^{\kappa+1})U_j}{2^\kappa}.$$

Solving this equation for κ, we have $\kappa = \max_j \frac{U_j^+ + U_j}{2U_j + \epsilon}$. Finally, similar to [KN08], since player j gains the largest utility if he cheats in the last round, our protocol is an everlasting ϵ-Nash equilibrium. □

3.10 Conclusion

We have presented *scalable* mechanisms for rational secret sharing problems. Our algorithms are scalable in the sense that the number of bits sent by each player is $O(\log n)$ and the latency is at most logarithmic in the number of players. For n-out-of-n rational secret sharing, we give a scalable algorithm that is a Nash equilibrium to solve this problem. For t-out-of-n rational secret sharing where, we give a scalable algorithm that is a ϵ-Nash equilibrium.

Chapter 4

Interactive Communication with Unknown Noise Rate

How can two parties run a protocol over a noisy channel? Interactive communication seeks to solve this problem while minimizing the total number of bits sent. Recently, Haeupler [Hae14] gave an algorithm for this problem that is conjectured to be optimal. However, as in previous work [Sch92, BN13, BK12, BR11, Bra12b, GMS11, GHS14, GH13], his algorithm critically relies on the assumption that the algorithm knows the noise rate in advance, *i.e.* , the algorithm knows in advance the number of bits that will be flipped by the adversary.

In this chapter, we remove this assumption. To do so, we add a new assumption of privacy. In particular, in our model, an adversary can flip an unknown number of bits, at arbitrary times, but he never learns the value of any bits sent over the channel. This assumption is necessary: with a public channel and unknown noise rate, the adversary can run a man-in-the-middle attack to mislead either party (see Theorem 4.9.1, Section 4.9).

Problem Overview We assume that Alice and Bob are connected by a noisy binary channel. Our goal is to build an algorithm that takes as input some distributed protocol π that works over a noise-free channel and outputs a distributed protocol π' that works over the noisy channel.

We assume an adversary chooses π, and which bits to flip in the noisy channel. The adversary knows our algorithm for transforming π to π'. However, he neither knows the private random bits of Alice and Bob, nor the bits sent over the channel, except when it is possible to infer these from knowledge of π and our algorithm.

We let T be the number of bits flipped by the adversary, and L be the length of π. As in previous work, we assume that Alice and Bob know L.

Our Results Our main result is summarized in the following theorem.

Theorem 4.0.1. Algorithm 3 tolerates an unknown number of adversarial errors, T, succeeds with high probability in the transcript length[1], L, and if successful, sends in expectation $L + O\left(\sqrt{L(T+1)\log L} + T\right)$ bits.

The number of bits sent by our algorithm is within logarithmic factors of optimal, assuming a conjecture from [Hae14] (see Theorem 4.9.3).

Results in this chapter first appeared in conference proceedings [DHM+15].

4.1 Related Work

For L bits to be transmitted from Alice to Bob, Shannon [Sha48] proposes an error correcting code of size $O(L)$ that yields correct communication over a *n*oisy channel with probability $1 - e^{-\Omega(L)}$. At first glance, this may appear to solve our problem. But consider an *i*nteractive protocol with communication complexity L, where Alice sends one bit, then Bob sends back one bit, and so forth where the value of each bit sent *d*epends on the previous bits received. Two problems arise. First, using block codewords is not efficient; to achieve a small error probability, "dummy" bits may be added to each bit prior to encoding, but this results in a superlinear blowup in overhead. Second, due to the interactivity, an error that occurs in the past can ruin all computation that comes after it. Thus, error correcting codes fall short when dealing with interactive protocols.

The seminal work of Schulman [Sch93, Sch92] overcame these obstacles by describing a deterministic method for simulating interactive protocols on noisy channels with only a constant-factor increase in the total communication complexity. This work spurred vigorous interest in the area (see [Bra12a] for an excellent survey).

Schulman's scheme tolerates an adversarial noise rate of $1/240$. It critically depends on the notion of a *t*ree code for which an exponential-time construction was originally provided. This exponential construction time motivated work on more efficient constructions [Bra12b, Pec06, MS14]. There were also efforts to create alternative codes [GMS11, ORS09]. Recently, elegant computationally-efficient schemes that tolerate a constant adversarial noise rate have been demonstrated [BK12, GH13]. Additionally, a large number of powerful results have improved the tolerable adversarial noise rate [BN13, BR11, GHS14, FGOS15, BE14].

The closest prior work to ours is that of Haeupler [Hae14]. His work assumes a fixed and known adversarial noise rate ϵ, the fraction of bits flipped by the adversary. Communication efficiency

[1] Specifically with probability at least $1 - \frac{1}{L \log L}$

is measured by *communication rate* which is L divided by the total number of bits sent. Haeupler [Hae14] describes an algorithm that achieves a communication rate of $1 - O(\sqrt{\epsilon \log\log(1/\epsilon)})$, which he conjectures to be optimal. We compare our work to his in Section 4.9.

Feinerman, Haeupler and Korman [FHK14] recently studied the interesting related problem of spreading a single-bit rumor in a noisy network. In their framework, in each synchronous round, each agent can deliver a single bit to a random anonymous agent. This bit is flipped independently at random with probability $1/2 - \epsilon$ for some fixed $\epsilon > 0$. Their algorithm ensures with high probability that in $O(\log n/\epsilon^2)$ rounds and with $O(n \log n/\epsilon^2))$ messages, all nodes learn the correct rumor. They also present a majority-consensus algorithm with the same resource costs, and prove these resource costs are optimal for both problems.

4.2 Formal Model

Our algorithm takes as input a protocol π which is a sequence of L bits, each of which is transmitted either from Alice to Bob or from Bob to Alice. As in previous work, we also assume that Alice and Bob both know L. We let Alice be the party who sends the first bit in π.

Channel Steps We assume communication over the channel is synchronous and individual computation is instantaneous. We define a *channel step* as the amount of time that it takes to send one bit over the channel.

Silence on the Channel When neither Alice nor Bob sends in a channel step, we say that the channel is silent. In any contiguous sequence of silent channel steps, the bit received on the channel in the first step is set by the adversary for free. By default, the bit received in subsequent steps of the sequence remains the same, unless the adversary pays for one bit flip in order to change it. In short, the adversary pays a cost of one bit flip each time it wants to change the value of the bit received in any contiguous sequence of silent steps.

4.3 Overview of Our Result

Challenges Can we adapt prior results by guessing the noise rate? Underestimation threatens correctness if the actual number of bit flips exceeds the algorithm's tolerance. Conversely, overestimation leads to sending more bits than necessary. Thus, we need a protocol that adapts to the adversary's actions.

One idea is to adapt the amount of communication redundancy based on the number of errors detected thus far. However, this presents a new challenge because the parties may have different views of the number of errors. They will need to synchronize their adaptions over the noisy channel. This is a key technical challenge to achieving our result.

Another technical challenge is termination. The length of the simulated protocol is necessarily unknown, so the parties will likely not terminate at the same time. After one party has terminated, it is a challenge for the other party to detect this fact based on bits received over the noisy channel.

A high-level overview of how we address these challenges is given in Section 4.5.4.

4.4 Chapter Organization

The rest of this chapter is organized as follows. In Section 4.5, we describe a simple algorithm for interactive communication that works when $T = O(L/\log L)$. We analyze this algorithm in Section 4.6. In Section 4.7, we describe an algorithm for interactive communication that works for any finite T; we prove this algorithm correction in Section 4.8. Section 4.9 gives some relevant remarks, including justifying private channels and comparing our algorithm with past work. Finally, we conclude and give directions for future work in Section 4.10.

4.5 Bounded T - Algorithm

In this section, we describe an algorithm that enables interactive communication problem when $T = O(L/\log L)$.

4.5.1 Overview, Notation and Definitions

Our algorithm is presented as Algorithm 21. The overall idea of the algorithm is simple: the parties run the original protocol π for a certain number of steps as if there was no noise. Then, Alice determines whether an error has occurred by checking a fingerprint from Bob. Based on the result of this verification, the computation of π either moves forward or is rewound to be performed again.

4.5.2 Helper Functions

Before giving details of the algorithm, we first describe some helper functions and notation (see Figure 4.1).

L	The length of the protocol to be simulated.
π	The L-bit protocol to be simulated, augmented by random bits to length $\left(1+\left\lceil \frac{L}{R_0} \right\rceil\right) R_0$.
$\pi[\mathcal{T},\ell]$	The result of the computation of the next ℓ bits of π after history $\mathcal{T}$.
R_0	Initial round size in the algorithm. This is the smallest power of 2 that is greater than $\sqrt{LF}$. So $\sqrt{LF} \leq R_0 \leq 2\sqrt{LF}$
F	The length of the fingerprint.
$\mathcal{T}_a$	Alice's tentative transcript.
$\mathcal{T}_b$	Bob's tentative transcript.
$\mathcal{T}_a^*$	Alice's verified transcript.
$\mathcal{T}_b^*$	Bob's verified transcript.
$\mathcal{T}[0:\ell]$	The first ℓ bits of $\mathcal{T}$. If $\lvert\mathcal{T}\rvert < L$ this is *null*

Figure 4.1: Glossary of Notation

Fingerprinting To verify communication, we make use of the following well-known theorem.

Theorem 4.5.1. [Naor and Naor [NN93]] For any positive integer $\mathcal{L}$ and any probability p, there exists a hash function $\mathcal{F}$ that given a uniformly random bit string S as the seed, maps any string of length at most $\mathcal{L}$ bits to a bit string hash value H, such that the collision probability of any two strings is at most p, and the length of S and H are $|S| = \Theta(\log(\mathcal{L}/p))$ and $|H| = \Theta(\log(1/p))$ bits.

We define two functions based on this theorem, h and MatchesFP. In this section, we will write h_L to denote that the probability of error p is polynomial in L. In particular, we can set $p = 1/L^2$, with fingerprints of size $O(\log L)$. The function $\mathrm{h}_L(T)$ takes a transcript T and returns a tuple (s,f), where s is uniformly random bit string and f is the output of the hash function $\mathcal{F}$ in the theorem above when given inputs s and T. We refer to this tuple as the *fingerprint* of T.

The function MatchesFP((s,f),T) takes a fingerprint (s,f) and a transcript T. It returns true if and only if the output of $\mathcal{F}$ when given bit string s and transcript T is equal to the value f. In both of these functions, the total length of the fingerprint is given by the value F, which will be defined later.

Algebraic Manipulation Detection Codes Our result makes critical use of Algebraic Manipulation (AMD) Codes from [CDF+08]. These codes provide three functions: amdEnc, IsCodeword and amdDec. The function amdEnc(m) creates an encoding of a message m. The function IsCodeword(m') returns true if and only if a received message m' is equal to amdEnc(m) for some sent message m. The function amdDec(m') takes a received value m', where IsCodeword(m'),

and returns the value m such that amdEnc$(m) = m'$. Intuitively, AMD Codes enable detection of bit corruptions on encoded words, with high probability.

We make use of the following theorem about AMD codes. This is a slight rewording of a theorem from [CDF+08].

Theorem 4.5.2. [CDF+08] For any $\delta > 0$, there exists functions amdEnc, IsCodeword and amdDec, such that, for any bit string m of length x:

- amdEnc(m) is a string of length $x + C\log(1/\delta)$, for some constant C;
- IsCodeword(amdEnc(m)) and amdDec(amdEnc(m)) $= m$;
- For any bit string $s \neq 0$ of length x, Pr(IsCodeword(amdEnc$(m) \oplus s$)) $\leq \delta$

In this section, we set $\delta = 1/L^2$ and add $O(\log L)$ additional bits to the message word. Also in this section, we will always encode strings of size $O(\log L)$, so the AMD encoded messages will be of size $O(\log L)$.

In the algorithm, we will denote the fixed length of the AMD-encoded fingerprint by F.

4.5.3 Remaining Notation

Transcripts We define Alice's *tentative transcript*, $\mathcal{T}_{\mathcal{A}}$, as the sequence of possible bits of π that Alice has either sent or received up to the current time. Similarly, we let $\mathcal{T}_{\mathcal{B}}$ denote Bob's transcript. For both Alice or Bob, we define a *verified transcript* to be the longest prefix of a transcript for which a verified fingerprint has been received. We denote the verified transcript for Alice as $\mathcal{T}^*_{\mathcal{A}}$, and for Bob as $\mathcal{T}^*_{\mathcal{B}}$. The notation $T \preccurlyeq T'$ signifies that a transcript T is a prefix of a transcript T'.

Rounds We define one of *Alice's rounds* as one iteration of the repeat loop in Alice's protocol. Alice's round consists of r_a channel steps, where r_a is the *round size* value maintained by Alice. Similarly, we define one of *Bob's rounds* as one iteration of the repeat look in Bob's protocol. Such a round consists of r_b channel steps, where r_b is the *round size* for Bob.

Other Notation For a transcript $\mathcal{T}$ and integer i, we define $\mathcal{T}[0:i]$ to be the first i bits of $\mathcal{T}$. For two strings x and y, we define $x \odot y$ to be the concatenation of x and y.

4.5.4 Algorithm Overview

To facilitate discussion of the algorithm, we first state some important properties of rounds (proven in Section 4.6). First, the size of any round is always a power of two. Second, the start of each of Bob's rounds always coincides with the start of one of Alice's rounds. This ensures that whenever Bob is listening for the message $\mathcal{F}_a'$, Alice will be sending such a message.

We first describe one of Alice's rounds in which 1) neither Alice nor Bob terminate; and 2) there are no adversarial bit flips. In such a round, Alice sends an encoded message containing two pieces of information. These are m_a, which is the number of failed rounds Alice has counted so far; and $|\mathcal{T}_a^*|$, which is the size of Alice's verified transcript.

When Bob decodes this message, he synchronizes several values with Alice. In particular, he sets his round size value, r_b, and mistake estimate value, m_b, so they equal the values Alice sent. Then, based on $|\mathcal{T}_a^*|$, Bob either increases the length of his verified transcript, or else decreases the length of his tentative transcript. After this synchronization, Alice and Bob both compute a certain number of bits of π and add these to their tentative transcripts. Finally Bob sends an encoded fingerprint to Alice. She verifies this fingerprint, and then adds the bits of π computed during this round to her verified transcript.

There are two key ways in which adversarial bit flips can alter the above scenario. First, when the encoded message Alice sends containing m_a and $|\mathcal{T}_a^*|$ is corrupted. In this case, Bob will send random bits for the remainder of the round. This ensures two things. First, whenever Alice is listening for a fingerprint from Bob, Bob will either be sending a fingerprint or random bits. Thus, w.h.p., the adversary will be unable to forge an encoding of a fake fingerprint by flipping bits. Second, Bob's error count updates at the same time as Alice's.

The other key way in which adversarial bit flips can alter the ideal scenario is as follows. The adversary flips bits in such a way that the encoded fingerprint, $\mathcal{F}_b'$ that Bob sends to Alice, fails to be a valid fingerprint for Alice's tentative transcript. In this case, Alice rewinds her tentative transcript, increments her error count, and updates her block size.

Handling Termination In previous work, since ϵ and L' are known, both parties know when to terminate (or *l*eave the protocol), and can do so at the same time. However, since we know neither parameter, termination is now more challenging.

In our algorithm, π is augmented with a certain number of additional bits that Alice sends to Bob. Each of these bits is set independently and uniformly at random by Alice. Alice terminates when her verified transcript is of length greater than L. Bob terminates when he receives a value $\mathcal{F}_a'$, where all bits are the same. This conditions ensures that 1) Bob is very unlikely to terminate

Protocol 21 Bounded Error Interactive Communication

Protocol for Alice

while $m_a = \frac{R_0^2}{4F^2} - 1$ **do**
  $\mathcal{F}_a \leftarrow \text{amdEnc}(m_a, r_a, |\mathcal{T}_a^*|)$
  Send $\mathcal{F}_a$
  Append $\pi[\mathcal{T}_a, r_a - 2F]$ to $\mathcal{T}_a$
  Receive Bob's F-bit message, $\mathcal{F}_b'$
  if IsCodeword($\mathcal{F}_b'$) **then**
    if $|\mathcal{T}_a^*| \geq L$ **then**
      Output $\mathcal{T}_a^*[0:L]$
      Terminate
    $\mathcal{F} \leftarrow \text{amdDec}(\mathcal{F}_b')$
    if MatchesFP($\mathcal{F}, \mathcal{T}_a$) **then**
// successful round
      $\mathcal{T}_a^* \leftarrow \mathcal{T}_a$
  else
// round failed
    $\mathcal{T}_a \leftarrow \mathcal{T}_a^*$
    $m_a \leftarrow m_a + 1$
    if $1 + m_a$ is a power of 4 **then**
      $r_a \leftarrow r_a/2$

Protocol for Bob

$\mathcal{T}_b \leftarrow null$; $\mathcal{T}_b^* \leftarrow null$ $m_b \leftarrow 0$; $r_b \leftarrow R_0$
while $m_b = \frac{R_0^2}{4F^2} - 1$ **do**
  Receive Alice's F-bit message, $\mathcal{F}_a'$
  if *all bits of* $\mathcal{F}_a'$ *are equal* **then**
// Alice has likely left
    Output $\mathcal{T}_b^*[0:L]$ and
Terminate
  if IsCodeword($\mathcal{F}_a'$) **then**
    $(m, r, \ell) \leftarrow \text{amdDec}(\mathcal{F}_a')$
// synchronize values
    $r_b \leftarrow r$
    $m_b \leftarrow m$
    if $\ell > |\mathcal{T}_b^*|$ **then**
      $\mathcal{T}_b^* \leftarrow \mathcal{T}_b$
    else
      $\mathcal{T}_b \leftarrow \mathcal{T}_b^*$
    Append $\pi[\mathcal{T}_b, r_b - 2F]$ to $\mathcal{T}_b$
    $\mathcal{F}_b \leftarrow \text{amdEnc}(\text{h}_L(\mathcal{T}_b))$
    Send $\mathcal{F}_b$
  else
// corruption occurred
    Send random bits for $r_b - F$ steps
    $m_b \leftarrow m_b + 1$
    if $1 + m_b$ is a power of 4 **then**
      $r_b \leftarrow r_b/2$

before Alice; and 2) Bob terminates soon after Alice, unless the adversary pays a significant cost to delay this.

4.6 Bounded T - Analysis

We now prove that with high probability, Algorithm 1 correctly simulates π when T is promised to be $O(L/\log L)$. Before proceeding to our proof, we define two bad events.

Hash Collision. Either Alice or Bob incorrectly validates a fingerprint and updates their verified transcript to include bits not in π.

Failure of AMD Codes The adversary corrupts an encoded message into the encoding or a different message. Or the encoding of some message, after possible adversary corruption, equals a bit string of all zeroes or all ones.

Throughout this section, we will assume neither event occurs. At the end of this section, we will show that the probability that either even occurs is polynomially small in L.

Lemma 4.6.1. Each player's round size is always a power of two.

Proof. This is immediate from the fact that the round size starts out as a power of 2 and the fact that each time it decreases, it decreases by a factor of 2. □

Lemma 4.6.2. m_a is monotonically increasing, and hence Alice's round size never increases.

Proof. This follows immediately from the fact that the only time m_a changes is on Line 15 of Alice's protocol, when it is incremented by 1. □

Lemma 4.6.3. Algorithm 21 has the following properties:

1. When Bob starts a round, Alice starts a round,
2. $m_b \leq m_a$ at all times that Alice remains in the protocol.

Proof. This follows by induction on m_a.

Base Case We first show that the lemma holds while $m_a = 0$. Note that m_b can only increase after Bob has spent a round sending random bits. During such a round, Alice will increment m_a before Bob increments m_b. Next, note that while $m_b = m_a = 0$, Alice and Bob both have the same round sizes, and so when Bob starts a round, Alice starts a round.

Inductive Step Consider the channel step, t, at which Alice increases m_a to some value $j > 0$. We must show that the lemma statement holds throughout the time while $m_a = j$. By the inductive hypothesis, up to time t, $m_b \leq m_a$, and when Bob started a round, Alice started a round. There are two cases for the value of m_b at the end of channel step t.

Case 1 $m_b < j$. In this case, Bob must not have received $\mathcal{F}_a$ at the beginning of the round he is in at channel step t. Hence, Bob transmits random bits during this entire round. Bob's round size is an integer multiple of Alice's round size (by Lemma 4.6.1). Thus, Bob will transmit random bits throughout Alice's round begun at channel step $t+1$. So Alice will not receive a matching fingerprint at the end of the round she began at step $t+1$, and so she will increment m_a before Bob increments m_b. This will happen before Bob completes the round he is in at time t, so both conditions of the lemma hold while $m_a = j$.

Case 2 $m_b = j$. Note that m_b can only increase after Bob has spent a round sending random bits. During such a round, Alice will increment m_a before Bob increments m_b. Thus, while $m_a = j$, $m_b = j$. Next, note that, if $m_b = m_a = j$ at step t, then Alice and Bob both ended their rounds at step t. Hence, during the time that $m_a = j$, when Bob starts a round, Alice starts a round. □

The following corollaries are immediate from the above lemma.

Corollary 4.6.4. When Bob ends a round, Alice ends a round.

Corollary 4.6.5. Bob's rounds are at least as large as Alice's rounds.

The following corollary holds from the above lemma and the fact that Bob's round sizes are at least as large as Alice's.

Corollary 4.6.6. While both parties remain in the protocol, whenever Bob is listening for a $\mathcal{F}_a$, Alice is sending it. Also, whenever Alice is listening for $\mathcal{F}_b$, either Bob is sending it, or Bob is sending random bits.

The following lemma also follows from Lemma 4.6.3.

Lemma 4.6.7. Let $\mathcal{R}$ be one of Alice's rounds which starts and ends at the same time as one of Bob's rounds. Then, at the end of $\mathcal{R}$, either $m_a - m_b$ is the same as it was at the beginning of $\mathcal{R}$ or it equals 0 or 1.

Proof. If $\mathcal{F}_a$ is corrupted at the beginning of $\mathcal{R}$, Bob transmits random bits for the rest of $\mathcal{R}$, and both Alice and Bob increment their error counts at the end, so $m_a - m_b$ stays the same.

If $\mathcal{F}_a$ is not corrupted at the beginning of $\mathcal{R}$, then Bob sets m_b to m_a at the beginning of $\mathcal{R}$, so at the end, $m_a - m_b \leq 1$. By Lemma 4.6.3 (2), $m_a - m_b \geq 0$. □

4.6.1 Phases

We now give some definitions.

Definition 4.6.8. We define *phase* j to be all of Alice's rounds of size $R_0/2^j$.

Definition 4.6.9. We define Δ_j, for all $j > 0$, to be the value $m_a - m_b$ at the end of phase j.

Note that at the beginning of phase j, Alice's error count is $4^j - 1$. We now give a few lemmas about phases.

Lemma 4.6.10. For any $j > 0$, phase j contains at least $3\Delta_{j-1}$ of Alice's rounds,

Proof. Consider any $j > 0$. At the beginning of phase j, $m_a = 4^j - 1$. Also, at the beginning of phase j, by Lemma 4.6.3 (2), $m_b \leq m_a$. Hence, $0 \leq \Delta_{j-1} \leq 4^j - 1$. Note that m_a increases by at most 1 in each of Alice's rounds. Thus, $3\Delta_{j-1}$ rounds after the beginning of phase j, the value of m_a is at most:

$$\begin{aligned} 4^j - 1 + 3\Delta_{j-1} &\leq 4^j - 1 + 3(4^j - 1) \\ &< 4^{j+1} - 1 \end{aligned}$$

Thus after $3\Delta_{j-1}$ rounds, m_a is not large enough for Alice to advance to phase $j+1$. □

Progressive, Corrupted and Wasted Rounds Let $\mathcal{R}$ be one of Alice's rounds. We call $\mathcal{R}$ *progressive* if Alice does not update her error count during the round, or equivalently if her verified transcript length increases. We call $\mathcal{R}$ *corrupted* if the adversary flipped at least one bit in the round. We call $\mathcal{R}$ *wasted* if it is neither progressive nor corrupted. We want to bound the number of wasted rounds since this number represents amount by which m_a is potentially an overestimate of T.

We note that wasted rounds occur only when $r_b > r_a$. In this case, Bob is not listening when Alice sends him $\mathcal{F}_a$. As a result, Bob does not send Alice a valid fingerprint at the end of her round, and so her verified transcript does not increase, even though the adversary has not flipped any bits.

The following lemma bounds the number of wasted rounds in a phase, and gives other critical properties.

Lemma 4.6.11. Suppose at the beginning of phase j, $j > 0$, Bob is at the start of a round and his round size is at most $R_0/2^{j-1}$. Then

1. There are at most Δ_{j-1} wasted rounds in phase j;

2. $\Delta_j \in \{0,1,2\Delta_{j-1}\}$; and

3. Bob ends a round at the end of phase j.

Proof. If Bob's round size initially less than $R_0/2^{j-1}$, then it must equal $R_0/2^j$ in order to be a power of two. Hence Alice and Bob will have rounds that are the same size for the entire phase, and the lemma holds trivially.

We now consider the harder case where Bob's round size equals $R_0/2^{j-1}$.

By Definition 4.6.8, Alice has round size $R_0/2^j$ throughout phase j. By Lemma 4.6.3 (2), Bob's round size is always greater than or equal to Alice's round size. Thus, as soon as 1) Bob receives $\mathcal{F}_a$ in one of his rounds in phase j, or 2) Bob sets m_b equal to Alice's error count at the beginning of phase j, then Bob's round size will be $R_0/2^j$ for the remainder of the phase. Finally, by Lemma 4.6.3 (1), from that point on, Alice and Bob will begin, and thus end, all rounds at the same time.

Now consider Bob's rounds in phase j. Assume the adversary corrupts $\mathcal{F}_a$ in Bob's rounds 1 through i for some value $i \geq 0$, and then the adversary does not corrupt $\mathcal{F}_a$ in Bob's round $i+1$. We consider two cases.

Case 1: $i < \Delta_{j-1}$ Each of the first i rounds of Bob spans two rounds of Alice. By Lemma 4.6.10, these rounds are all contained in phase j. Consider each pair of Alice's rounds spanned by one of Bob's rounds. The first round in the pair is corrupted, but during the second, Bob is transmitting random bits and Alice will not receive a fingerprint from him. Thus, this round is wasted. Hence, there are i wasted rounds.

In round $i+1$, Bob synchronizes his round size with Alice since he receives $\mathcal{F}_a$. Thus, there are no more wasted rounds. Applying Lemma 4.6.7 for the remaining rounds of the phase, we see that at the end of the phase, $m_a - m_b = \Delta_j$ is either 0 or 1.

Case 2: $i \geq \Delta_{j-1}$ Bob increases m_b by 1 in each of his first i rounds. Note that at the beginning of phase j, Alice's error count is $4^j - 1$. Thus, after Bob's first i rounds, $m_b = (4^j - 1) - \Delta_{j-1} + i$. Hence when $i = \Delta_{j-1}$, $m_b = (4^j - 1)$. At that time, Bob sets his round size to $R_0/2^j$, and so Alice and Bob will have the same round sizes, and will hence begin and end all rounds at the same step, for the rest of phase j. Thus, there are no more wasted rounds. Note that in this case, at Bob's Δ_{j-1} round, $m_a - m_b$ will be $2\Delta_{j-1}$. Applying Lemma 4.6.7 for the remaining rounds of the phase, we see that $\Delta_j = 2\Delta_{j-1}$, or Δ_j is 0 or 1. □

Lemma 4.6.12. For every $j \geq 0$:

1. There are at most 2^{j-1} wasted rounds in phase j;

2. $\Delta_j \leq 2^j$; and

3. Bob ends a round at the end of phase j.

Proof. We prove this by induction on j.

Base Case At the beginning of phase 0, Bob is at the start of a round and his round size is R_0. Thus, by Lemma 4.6.11: there are 0 wasted rounds in phase 0; $\Delta_0 \leq 1$; and Bob ends a round at the end of phase 0.

Inductive Step Consider some $j > 0$. By the inductive hypothesis, $\Delta_{j-1} \leq 2^{j-1}$. At the beginning of phase j, $m_b = m_a - \Delta_{j-1} \leq (4^j - 1) - \Delta_{j-1}$, so that $r_b = R_0/2^{\lfloor \log_4(1+m_b) \rfloor} \leq R_0/2^{\lfloor \log_4(4^j - \Delta_{j-1}) \rfloor} \leq R_0/2^{j-1}$. The last line holds since $0 \leq \Delta_{j-1} \leq 2^{j-1}$.

Also, by the inductive hypothesis, Bob ended a round at the end of phase $j-1$, and so is starting a round at the beginning of phase j. Hence, we can apply Lemma 4.6.11 to phase j. From this lemma, it follows that 1) the number of wasted rounds in phase j is at most 2^{j-1}; 2) $\Delta_j \leq 2\Delta_{j-1} \leq 2^j$; and 3) Bob ends a round at the end of phase j. □

Note from the above lemma that Bob's rounds are never more than double the size of Alice's rounds. The following lemma sums up what we now know about Alice and Bob's rounds.

Lemma 4.6.13. The following are always true.

1. Bob's round size is either equal to Alice's round size or double Alice's round size.

2. If Bob's round size equals Alice's round size, then when Alice starts a round, Bob starts a round.

3. If Bob's round size is twice Alice's round size, then when Alice starts a round, either Bob starts a round, or Bob is in the middle of a round.

Proof. The lemma follows from Corollary 4.6.5, Lemma 4.6.3, and Lemma 4.6.12. □

4.6.2 Correctness and Termination

Lemma 4.6.14. It is always the case that $\mathcal{T}_a^* \preccurlyeq \pi$, where π is the padded transcript.

Proof. This holds by Lemma 4.6.25 and Lemma 4.6.26 and the fact that Alice never adds any string to $\mathcal{T}_a^*$ that is not verified by an encoded fingerprint from Bob. □

Lemma 4.6.15. At the beginning and end of each of Alice's rounds,

$$\mathcal{T}_b^* \preccurlyeq \mathcal{T}_a^* = \mathcal{T}_a \preccurlyeq \mathcal{T}_b;$$

where at most one of the inequalities is strict. Moreover, at the end of a channel step in which Bob receives $\mathcal{F}_a$ correctly,

$$\mathcal{T}_b^* = \mathcal{T}_b = \mathcal{T}_a^*.$$

Proof. We prove this by induction on Alice's round number.

Base Case At the beginning of the algorithm, all transcripts are *null*, so $\mathcal{T}_b^* = \mathcal{T}_a^* = \mathcal{T}_a = \mathcal{T}_b$. Moreover if Bob receives $\mathcal{F}_a$ correctly in this round, then $\mathcal{T}_b^* = \mathcal{T}_b = \mathcal{T}_a^*$.

Inductive Step We must show that the lemma holds for the j-th round. By the inductive hypothesis, at the end of the $j-1$-th round,

$$\mathcal{T}_b^* \preccurlyeq \mathcal{T}_a^* = \mathcal{T}_a \preccurlyeq \mathcal{T}_b,$$

with at most one of the inequalities being strict. Clearly the statement about the inequalities will thus hold at the beginning of the j-th round.

Alice's j-th round starts with Alice sending Bob $\mathcal{F}_a$.

Case 1: Bob does not receive $\mathcal{F}_a$ If Bob does not receive $\mathcal{F}_a$, then either 1) he was listening and it was corrupted; or 2) he was not listening for it. If he was listening and $\mathcal{F}_a$ was corrupted, then Bob transmits random bits for the remainder of his round, which will be the remainder of Alice's round by Lemma 4.6.13. By the same lemma, if Bob was not listening, then he must be in the middle of a round that is twice as large as Alice's. In either case, Bob transmits random bits for the remainder of Alice's j-th round.

Thus, Alice does not receive a matching fingerprint from Bob at the end of her j-th round. Thus, at the end of her round, $\mathcal{T}_a \leftarrow \mathcal{T}_a^*$ and $\mathcal{T}_b$ and $\mathcal{T}_b^*$ are unchanged. Hence, it continues to hold that:

$$\mathcal{T}_b^* \preccurlyeq \mathcal{T}_a^* = \mathcal{T}_a \preccurlyeq \mathcal{T}_b;$$

and at most one of the inequalities is strict.

Case 2: Bob receives $\mathcal{F}_a$ If Bob receives $\mathcal{F}_a$, then he learns the length of $\mathcal{T}_a^*$ and also Alice's round size. By the inductive hypothesis, either $\mathcal{T}_a^* = \mathcal{T}_b^*$ or $\mathcal{T}_a^* = \mathcal{T}_b$. Based on the length of $\mathcal{T}_a^*$, Bob either updates $\mathcal{T}_b^*$ or rewinds $\mathcal{T}_b$, so that $\mathcal{T}_b^* = \mathcal{T}_b = \mathcal{T}_a^*$. This establishes the second part of the lemma for the j-th round.

Next Alice and Bob continue their rounds which are the same size. If Alice receives a correct fingerprint from Bob at the end of her round, then the following holds:

$$\mathcal{T}_b^* \preccurlyeq \mathcal{T}_a^* = \mathcal{T}_a = \mathcal{T}_b.$$

If Alice does not receive a correct fingerprint from Bob at the end of her round, then the following holds:

$$\mathcal{T}_b^* = \mathcal{T}_a^* = \mathcal{T}_a \preccurlyeq \mathcal{T}_b.$$

In either case, the first part of the lemma statement holds at the end of Alice's j-th round. □

Lemma 4.6.16. Bob leaves after Alice. When Alice leaves, $|\mathcal{T}_b^*| \geq L$.

Proof. Bob leaves only when he receives an $\mathcal{F}_a'$ that is all zeroes or all ones. By Lemma 4.6.26, $\mathcal{F}_a'$ is never such a string, and the adversary cannot convert $\mathcal{F}_a$ to such a string by bit flipping. It follows that Bob receives such a string only after Alice has left.

Alice leaves only when 1) she has received an encoded fingerprint from Bob; and 2) $|\mathcal{T}_a^*| \geq L$. If Alice receives a correctly encoded fingerprint from Bob, then by Lemma 4.6.26, Bob must have sent one, and hence Bob must be in a round where he received $\mathcal{F}_a$ correctly. By Lemma 4.6.15, at that channel step, $\mathcal{T}_b^* = \mathcal{T}_b = \mathcal{T}_a^*$. Hence at the step when Alice receives the encoded fingerprint from Bob, $\mathcal{T}_b^* = \mathcal{T}_a^*$. Thus, when Alice leaves, $|\mathcal{T}_b^*| \geq L$. □

Lemma 4.6.17. When either party terminates, their output is correct.

Proof. The proof follows from Lemmas 4.6.14, 4.6.15, and 4.6.16, and the fact that when either party terminates, they output the first L bits of their verified transcript. □

4.6.3 Cost

Lemma 4.6.18. After Alice leaves, the adversary must flip at least one bit for each of Bob's rounds that does not result in Bob leaving.

Proof. After Alice has left, there is silence on the channel in the steps when Bob is listening for Alice's encoded message. This means that if there is no bit flipping by the adversary, the channel transmits the same bit in every channel step, causing Bob to read a string of all zeroes or all

ones, and terminate. Thus, the adversary must flip at least one bit each time Bob is listening for a codeword. □

Lemma 4.6.19. There are at most $2^j - 1$ wasted rounds prior to the end of phase j, for all $j \geq 0$.

Proof. This follows trivially by repeated applications of Lemma 4.6.12 (1). □

Throughout this section, we assume the worst case, that the adversary corrupts at most one bit per corrupted round.

Lemma 4.6.20. At all times, $m_a \leq T + \sqrt{T}$. In particular, there are no more than $\sqrt{T}$ wasted rounds.

Proof. By way of contradiction, assume $m_a > T + \sqrt{T}$ at some step, in some phase j, $j \geq 0$. Then the number of wasted rounds at this step must be greater than $\sqrt{T}$. But by Lemma 4.6.19, the number of wasted rounds at the end of phase j is no more than $2^j - 1$. Thus, we have $\sqrt{T} < 2^j - 1$, or $T < (2^j - 1)^2$.

But m_a is no larger than the number of corrupted rounds plus the number of wasted rounds. By the above paragraph, $T < (2^j - 1)^2$ and the number of wasted rounds is no more than $2^j - 1$. Thus $m_a < (2^j - 1)^2 + (2^j - 1)$. Moreover, we know that in phase j, $m_a \geq 4^j - 1$. Thus, we know

$$4^j - 1 < (2^j - 1)^2 + (2^j - 1).$$

Simplifying, we get $2^j < 1$, which is a contradiction for any $j \geq 0$. □

Let m_a^* denote Alice's error count when she leaves the algorithm, and m_b^* denote Bob's error count when he himself leaves the algorithm.

Lemma 4.6.21. Alice terminates in at most $L + O(\sqrt{LF(1 + m_a^*)})$ steps.

Proof. We first calculate the cost of the rounds that are not progressive for Alice. The number of non-progressive rounds that she has executed is m_a^*. Her cost for these rounds is at most the following.

$$\begin{aligned}\sum_{i=1}^{m_a^*} \frac{R_0}{2^{\lfloor \log_4 i \rfloor}} &\leq 2R_0 \sum_{i=1}^{m_a^*} \frac{1}{2^{\log_4 i}} \\ &= 2R_0 \sum_{i=1}^{m_a^*} \frac{1}{\sqrt{i}} \\ &\leq 2R_0 \int_0^{m_a^*} \frac{1}{\sqrt{i}} \\ &= 4R_0 \sqrt{m_a^*}\end{aligned}$$

In every progressive round, except possibly the last, Alice's block size is at least $R_0 2^{-\log_4(1+m_a^*)}$. Thus in all but possibly the last progressive round, Alice always adds bits to her verified transcript at a rate of at least

$$\frac{R_0 2^{-\log_4(1+m_a^*)} - 2F}{R_0 2^{-\log_4(1+m_a^*)}}.$$

Thus, the total number of bits Alices sends in all but the last progressive round is no more than

$$L \cdot \frac{R_0 2^{-\log_4(1+m_a^*)}}{R_0 2^{-\log_4(1+m_a^*)} - 2F}.$$

We will make use of the inequality

$$\frac{1}{1-\delta} \leq 1 + 2\delta \qquad \text{for } 0 < \delta \leq 1/2$$

and let $\delta = 2F / R_0 2^{-\log_4(1+m_a^*)}$. Note that $\delta \leq 1/2$, since Alice's round size is always at least $4F$.

Then we have that the total number of bits sent by Alice in all but the last progressive round is no more than

$$L + \frac{4LF}{R_0 2^{-\log_4(1+m_a^*)}}.$$

Adding in the last progressive round, we get that the total number of bits sent by Alice in progressive rounds is no more than

$$L + \frac{4LF}{R_0 2^{-\log_4(1+m_a^*)}} + R_0 2^{-\log_4(1+m_a^*)}.$$

Putting this together with the number of bits send in non-progressive rounds, we have that the total number of bits send by Alice is no more than

$$
\begin{aligned}
L + 4R_0\sqrt{m_a^*} + \frac{4LF}{R_0 2^{-\log_4(1+m_a^*)}} + R_0 2^{-\log_4(1+m_a^*)} &\le L + 5R_0\sqrt{m_a^*} + 4\sqrt{LF}(2^{\log_4(1+m_a^*)}) \\
&\le L + 10\sqrt{LFm_a^*} + 4\sqrt{LF(1+m_a^*)} \\
&\le L + 14\sqrt{LF(1+m_a^*)} \qquad \square
\end{aligned}
$$

Lemma 4.6.22. Bob terminates in at most $L + 14\sqrt{LF(1+m_a^*)} + 8\sqrt{LFm_b^*}$ steps.

Proof. Since Bob never leaves before Alice, Bob's cost must be at least as much as Alice's. We now compute Bob's additional cost.

At the time of Alice's departure, $r_a = R_0/2^{\lfloor \log_4(1+m_a^*)\rfloor}$. By Lemma 4.6.13, $r_b \le 2R_0/2^{\lfloor \log_4(1+m_a^*)\rfloor}$. Let m_b' denote Bob's error count when Alice leaves the algorithm. Then $1 + m_b' \ge 4^{\lfloor \log_4(1+m_a^*)\rfloor - 1}$. Bob's final error count is m_b^*. Thus, Bob's additional cost is at most

$$
\begin{aligned}
\sum_{i=m_b'}^{m_b^*-1} \frac{R_0}{2^{\lfloor \log_4(1+i)\rfloor}} &\le 2R_0 \sum_{i=1}^{m_b^*} \frac{1}{2^{\log_4 i}} \\
&= 2R_0 \sum_{i=1}^{m_b^*} \frac{1}{i^2} \\
&\le 4R_0\sqrt{m_b^*} \\
&\le 8\sqrt{LFm_b^*}
\end{aligned}
$$

Combining this with Alice's cost gives the result. $\square$

Lemma 4.6.23. The algorithm ends in at most $12L$ time steps.

Proof. By Lemma 4.6.22, Bob terminates in at most $L + 14\sqrt{LF(1+m_a^*)} + 8\sqrt{LFm_b^*}$ steps. Moreover, m_a^* and m_b^* are no more than $R_0^2/4F^2 - 1$. Thus, the algorithm terminates in at most the following number of steps.

$$
\begin{aligned}
L + 14\sqrt{LF(1+m_a^*)} + 8\sqrt{LFm_b^*} &\le L + 22\sqrt{\frac{LFR_0^2}{4F^2}} \\
&= L + 22\sqrt{\frac{L^2}{4}} \\
&= 12L\,. \qquad \square
\end{aligned}
$$

Lemma 4.6.24. If $T \leq \frac{L}{8F} - 1$ then both players terminate with the correct output in at most $L + O(\sqrt{LF(T+1)})$ steps.

Proof. Let T_a denote the number of bits flipped by the adversary while Alice is still in the protocol, and T_b the bits flipped after Alice has left. Then $T_a + T_b = T$.

By Lemma 4.6.20, $m_a^* \leq T_a + \sqrt{T_a}$. By Lemmas 4.6.3 and 4.6.18, $m_b^* \leq m_a^* + T_b$. Since $T_a + T_b = T$ it follows that

$$m_a^* \leq T + \sqrt{T} \leq 2T \leq \frac{L}{4F} - 2 < \frac{R_0^2}{4F^2} - 1$$

and similarly

$$m_b^* < \frac{R_0^2}{4F^2} - 1$$

Thus, Alice and Bob will both terminate by outputting the bits of π by Lemma 4.6.17.

Plugging $m_a^* \leq 2T$ and $m_b^* \leq 3T$ into Lemma 4.6.22 gives the total number of steps required. □

Lemma 4.6.25. With high probability in L, there are no hash collisions.

Proof. By Lemma 4.6.23, the algorithm ends in at most 12L steps. Also, there are at least $4F = \Theta(\log L)$ steps in a round. Thus, the algorithm has at most $O(L \log L)$ rounds. Each round has one fingerprint. By Theorem 4.5.1 and the setting of our fingerprint sizes, each fingerprint fails with probability at most $1/L^2$. Thus, a simple union bound gives the result. □

Lemma 4.6.26. With high probability in L, any bit flipping of a AMD encoded message is detected.

Proof. We noted in the previous lemma that the algorithm terminates in $O(L \log L)$ rounds. Each round has two AMD encoded messages. By Theorem 4.5.2 and the setting of our encoding sizes, each AMD encoding fails with probability at most $1/L^2$. Again, a union bound gives the result. □

4.7 Unbounded T - Algorithm

Algorithm 1 uses fingerprints of a fixed size, F in order to check its transcripts. Each of these has a $1/L^2$ chance to fail due to a hash collision. Since the algorithm only computes about $O(L/\log L)$ fingerprints, a union bound tells us that with high probability the algorithm succeeds, below its threshold value of T. When T is large, many more checks may need to be made, and eventually there will be a good chance that there is a hash collision. Since the algorithm cannot really recover from a hash collision, we cannot afford this. On the other hand, we cannot simply start out with larger fingerprints, both because this would be too expensive if T turned out to be small, and also

because even bigger fingerprints are still of a fixed size and eventually become unreliable. A natural solution is is to allow the fingerprints to grow, adapting the size to the value of T seen so far, and this is indeed what we will do.

Protocol 22 Interactive Communication: Iteration j

Protocol for Alice

Parameters: N_j, F_j, ρ_j

for $i = 1$ to N_j **do**
  $\mathcal{F}_a \leftarrow \text{ecEnc}(\text{amdEnc}(|\mathcal{T}_a^*|))$
  Send $\mathcal{F}_a$
  if $|\mathcal{T}_a^*| < L$ **then**
    for the next $\lfloor F_j/\rho_j \rfloor$ bits of π **do**
      if sender **then**
        Send next bit ρ_j times
        Append to $\mathcal{T}_a$
      else
        Receive ρ_j bits
        Append majority bit to $\mathcal{T}_a$
  else
    Transmit F_j random bits.
  Receive Bob's cF_j-bit message, $\mathcal{F}_b'$
  if IsCodeword($\mathcal{F}_b'$) **then**
    if $|\mathcal{T}_a^*| \geq L$ **then**
      Output $\mathcal{T}_a^*[0:L]$
      Terminate
    $\mathcal{F} \leftarrow \text{amdDec}(\mathcal{F}_b')$
    if MatchesFP($\mathcal{F}, \mathcal{T}_a$) **then**
// successful round
      $\mathcal{T}_a^* \leftarrow \mathcal{T}_a$
    else
// round failed
      $\mathcal{T}_a \leftarrow \mathcal{T}_a^*$

Protocol for Bob

Parameters: N_j, F_j, ρ_j

for $i = 1$ to N_j **do**
  if $|\mathcal{T}_b^*| \geq L$ **then**
    Wait cF_j channel steps
    Receive F_j bits
    if fewer than $F_j/3$ alternations in the received string **then**
      Output $\mathcal{T}_b^*[0:L]$
      Terminate
    else
      $\mathcal{F}_b \leftarrow \text{ecEnc}(\text{amdEnc}(\text{h}_j(\mathcal{T}_b^*)))$
      Send $\mathcal{F}_b$
  else
    Receive Alice's cF_j-bit message $\mathcal{F}_a'$
    if IsCodeword(ecDec($\mathcal{F}_a'$)) **then**
      $\ell \leftarrow \text{amdDec}(\text{ecDec}(\mathcal{F}_a'))$
      if $\ell > |\mathcal{T}_b^*|$ **then**
        $\mathcal{T}_b^* \leftarrow \mathcal{T}_b$
      else
        $\mathcal{T}_b \leftarrow \mathcal{T}_b^*$
      for the next $\lfloor F_j/\rho_j \rfloor$ bits of π **do**
        if sender **then**
          Send next bit ρ_j times
          Append to $\mathcal{T}_b$
        else
          Receive ρ_j bits
          Append majority bit to $\mathcal{T}_b$
      $\mathcal{F}_b \leftarrow \text{ecEnc}(\text{amdEnc}(\text{h}_j(\mathcal{T}_b)))$
      Send $\mathcal{F}_b$
    else
      Transmit $(c+1)F_j$ random bits.

Protocol 23 Interactive Communication

Protocol for Alice	**Protocol for Bob**
// Iteration 0	// Iteration 0
1: Run Alice's protocol from Alg 1	1: Run Bob's protocol from Alg 1
2: **if** not terminated **then**	2: **if** not terminated **then**
3: transmit random bits until channel step $12L$	3: transmit random bits until channel step $12L$
// End of Iteration 0	// End of Iteration 0
4: $j \leftarrow 1$	4: $j \leftarrow 1$
5: **while** still present **do**	5: **while** still present **do**
// Iteration j	// Iteration j
6: $F_j \leftarrow \beta(j + \log L)$	6: $F_j \leftarrow \beta(j + \log L)$
7: $\rho_j \leftarrow 2^j \lceil \frac{F_j}{F} \rceil \wedge F_j$	7: $\rho_j \leftarrow 2^j \lceil \frac{F_j}{F} \rceil \wedge F_j$
8: $N_j \leftarrow 2^{j-1} \lceil 8L/F \rceil$	8: $N_j \leftarrow 2^{j-1} \lceil 8L/F \rceil$
9: Run Alice's protocol from Algorithm 2, with parameters N_j, F_j, ρ_j	9: Run Bob's protocol from Algorithm 2, with parameters N_j, F_j, ρ_j
// End of Iteration j	// End of Iteration j
10: $j \leftarrow j + 1$	10: $j \leftarrow j + 1$

4.7.1 Helper Functions

As in Algorithm 1, we make black-box use of the Naor and Naor hash family, as well as AMD codes to protect information. However, in Iteration j we need the failure probabilities for both these primitives to be $1/(2^j L^2)$. Thus, we want the fingerprint size to grow with j. We will denote the hash function which has a collision probability of at most $1/(2^j L^2)$ by h_j. [1] It is easy to see that $O(j)$ extra bits are required for this, so that the fingerprint size is $O(j + \log L)$.

Algorithm 1 works well when the adversary can only afford to flip a fraction of a bit per block of the algorithm. In this case, it doesn't matter that he can corrupt an entire round of the protocol by flipping a single bit. However, when the adversary has a larger budget, it becomes crucial to force him to pay a larger price to corrupt a round. To this end, we wrap each fingerprint and protocol bit in a linear error-correcting code.

To be concrete, we will use a repetition code for each protocol bit, and a Reed-Solomon code [RS60] to provide the already AMD-encoded messages with a degree of error correction. This enables us to encode a message so that it can be recovered even if the adversary corrupts a third

[1] By abuse of notation, we will *not* subscript all the other helper functions with j; it should be clear from context that the version of the function used is the one that operates on strings of the correct size and has the correct failure probability

of the bits. We will denote the encoding and decoding functions by ecEnc and ecDec respectively. The following theorem, a slight restatement from [RS60], gives the properties of these functions.

Theorem 4.7.1. [RS60] There is a constant $c > 0$ such that for any message m, $|\text{ecEnc}(m)| \leq c|m|$. Moreover, if m' differs from $\text{ecEnc}(m)$ in at most one-third of its bits, then $\text{ecDec}(m') = m$.

Finally, we observe that the linearity of ecEnc and ecDec ensure that when the error correction is composed with the AMD code, the resulting code has the following properties:

1. If at most a third of the bits of the message are flipped, then the original message can be uniquely reconstructed by rounding to the nearest codeword in the range of ecEnc.

2. Even if an arbitrary set of bits is flipped, the probability of the change not being recognized is at most δ, *i.e.* the same guarantee as the AMD codes.

This is because ecDec is linear, so when noise η is added by the adversary to the codeword x, effectively what happens is the decoding function $\text{ecDec}(x+\eta) = \text{ecDec}(x) + \text{ecDec}(\eta) = m + D(\eta)$, where m is the AMD-encoded message. But now $\text{ecDec}(\eta)$ is an obliviously selected string added to the AMD-encoded codeword.

4.7.2 Algorithm

Let $N_1 := \lceil 8L/F \rceil$ be the number of rounds in Iteration 1. Let $N_j := 2^{j-1}N_1$ be the number of rounds in Iteration $j > 1$. Let $F_j = 2\beta j + F$ be the size of the fingerprints in Iteration j, where β is the constant from the Naor and Naor hash function. Thus the hash collision probability of a single fingerprint is $2^{-2j}L^{-2}$. Each round of the iteration begins with Alice sending Bob a (1/3)-error-corrected, AMD-encoded synchronization message of length cF_j, followed by simulation of the protocol for F_j channel steps, followed by Bob sending Alice a (1/3)-error-corrected, AMD-encoded fingerprint of length cF_j. Here c is the constant factor blowup we get from the ECC and AMD encodings, but for technical reasons we will further ensure that it is at least 5. Thus, the total round length is $(2c+1)F_j \geq 11F_j$. We will let α equal $(2c+1)$.

As in Algorithm 1, Alice will decide whether to update her verified transcript and advance to the next block of π or to rewind to redo the current block, based on whether she receives a fingerprint from Bob that matches the fingerprint of her own transcript. Similarly, Bob will decide whether to join in the simulation of π or to transmit random bits until the end of the round based on receiving or failing to receive Alice's synchronization message at the round's start. Where the round differs from a round in Algorithm 1, is in the actual simulation of π. For the whole iteration, a fixed number of

bits of π will be simulated per round. Each bit will be repeated $\rho_j = 2^{j-1}\lceil F_j/F\rceil \wedge F_j$ times [1]. The receiving party will use majority filtering to infer the transmitted bit. Since F_j time steps in the round are allocated to protocol simulation, this allows $\lfloor F_j/\rho_j \rfloor$ bits of π to be simulated.

Notice that the number of rounds doubles from one iteration to the next. Also, the number of repetitions of each simulated bit also roughly doubles between iterations, at least until it hits its cap, which is a constant fraction of the length of the round. This is the so-called doubling trick, (though in our case perhaps it should be quadrupling) which results in the overall cost being dominated by the cost in the last (or second to last) iteration.

4.8 Unbounded T - Analysis

We now analyze the main algorithm presented in Section 4.7. As in Section 4.6, we begin by noting that a hash collision or an AMD code failure will cause the algorithm to fail. Additionally, the algorithm could fail during the padding rounds, if the adversary happens to flip bits in such a way as to cause Alice's random bits to look like silence, resulting in Bob's premature departure.

In Section 4.8.3 we will show that with high probability each of these events does not occur. Meanwhile, throughout this section we will assume without further mention that we are in the good event where none of the undesirable events occur.

4.8.1 Alice and Bob are both present

Lemma 4.8.1. For every $j \geq 1$, Alice and Bob are always synchronized. That is, they begin the iteration as well as every round therein at the same time.

Proof. Alice and Bob synchronize themselves after Iteration 0 by both starting Iteration 1 at channel step $12L + 1$. Thereafter, for each $j \geq 1$, they have the same round sizes αF_j and number of rounds N_j in Iteration j, so that they remain synchronized. □

We will call a round *corrupted* if enough bits are flipped in the round that the bits of π being simulated cannot be recovered or verified by Alice. We will call it *uncorrupted* or *progressive* if it is not corrupted in the above sense.

Lemma 4.8.2. Each round is either corrupted at a cost of at least $\rho_j/2$ to the adversary or results in $\lfloor F_j/\rho_j \rfloor$ bits of progress in π.

[1] We remind the reader that $x \wedge y$ denotes the minimum of x and y, while $x \vee y$ denotes their maximum.

Proof. Since each simulated protocol bit is sent ρ_j times, with majority filtering at the receiving end, it costs the adversary $\rho_j/2$ to corrupt the repetition-encoded bit. It costs the adversary at least $cF_j/3 \geq \rho_j/2$ to corrupt Alice's synchronization message or Bob's fingerprint since these are protected by error-correction. Thus it costs the adversary at least $\rho_j/2$ to corrupt the round. Otherwise, since there are F_j steps allocated to sending protocol bits, and each one is repeated ρ_j times, the protocol is successfully simulated for $\lfloor \frac{F_j}{\rho_j} \rfloor$ bits. □

The following lemma is the equivalent of Lemmas 4.6.14 to 4.6.17 for Iteration j. Its proof is nearly identical to the proofs in Section 4.6.2 (indeed, it is simpler, since Iteration j does not have the synchronization problems faced by Algorithm 1) and we omit it.

Lemma 4.8.3. Iteration j has the following properties:

1. It is always the case that $\mathcal{T}_a^* \preccurlyeq \pi$, where π is the padded transcript.

2. At the beginning and end of each round,

$$\mathcal{T}_b^* \preccurlyeq \mathcal{T}_a^* = \mathcal{T}_a \preccurlyeq \mathcal{T}_b;$$

   where at most one of the inequalities is strict. Moreover, at the end of a channel step in which Bob receives $\mathcal{F}_a$ correctly,

$$\mathcal{T}_b^* = \mathcal{T}_b = \mathcal{T}_a^*.$$

3. Bob leaves after Alice. When Alice leaves, $|\mathcal{T}_b^*| \geq L$.

4. When either party terminates, their output is correct.

Lemma 4.8.4. There are at most $N_j/4$ uncorrupted rounds in Iteration j

Proof. Since each uncorrupted round results in $\lfloor F_j/\rho_j \rfloor$ bits of progress in π, $\lceil L\rho_j/F_j \rceil$ rounds are sufficient for Alice's transcript length to exceed L. One additional uncorrupted round is sufficient for Bob to catch up to Alice if necessary, using her synchronization message, and for Alice to infer from Bob's fingerprint that Bob's transcript length has exceeded L, resulting in Alice's departure. After that, if a round is uncorrupted, then Bob will perceive silence on the channel, resulting in Bob's departure. Thus $\lceil L\rho_j/F_j \rceil + 2$ uncorrupted rounds are enough for both parties to terminate. Finally note that for all $j \geq 1$,

$$\frac{\rho_j}{F_j} \leq \frac{2^{j-1}}{F} \wedge 1 \leq \frac{2^{j-1}}{F}$$

It follows that (for sufficiently large L) there are at most $2^j L/F = N_j/4$ uncorrupted rounds in Iteration j. □

The following corollary is immediate.

Corollary 4.8.5. If j is not the last iteration, then at least 3/4 of the rounds are corrupted.

Although the adversary can flip any number of bits in a round, we will only charge him the minimum number of bit-flips required for the outcome we see in the round, *i.e.* , we will charge him 0 for uncorrupted rounds and $\rho_j/2$ for corrupted rounds. Let T_j denote the number of corruptions charged to the adversary in Iteration j. Clearly, for $j > 0$

$$T_j \leq \frac{1}{2} N_j \rho_j \tag{4.1}$$

Also, we know from Section 4.5 that if the algorithm does not end in Iteration 0, then $T_0 \geq L/8F$. In this case, we will generously only charge the adversary that amount. In other words, if Iteration 1 is reached, either by both Alice and Bob, or by Bob alone, $T_0 = \lceil L/8F \rceil$.

Lemma 4.8.6. If j is not the last iteration then $T_j \geq \frac{3}{8} N_j \rho_j$

Proof. This follows from Corollary 4.8.5, since it costs the adversary at least $\rho_j/2$ to corrupt a round. □

Lemma 4.8.7. If j is not the last iteration then

$$3T_{j-1}/2 \leq T_j \leq 64T_{j-1}$$

Proof. If $j = 1$

$$T_1 \geq \frac{3}{8} N_1 \rho_1 \geq \frac{3L}{F} \geq 24T_0 > 3T_0$$

and

$$T_1 \leq N_1 \rho_1/2 \leq \frac{8L}{F} = 64T_0 .$$

If $j > 1$, then by (4.1) and Lemma 4.8.6,

$$\frac{3}{2} \frac{3N_j \rho_j/8}{N_{j-1} \rho_{j-1}/2} \leq \frac{T_j}{T_{j-1}} \leq \frac{N_j \rho j/2}{3N_{j-1} \rho_{j-1}/8} \leq 64$$

since $N_{j-1} = N_j/2$ and $\rho_{j-1} \leq \rho_j \leq 4\rho_{j-1}$. □

Lemma 4.8.8. The cost to either player due to uncorrupted rounds in Iteration $j \leq \log F$ is at most

$$7\alpha \sqrt{LT_{j-1}F}$$

Proof. Each uncorrupted round costs the players αF_j. Since there are at most $N_j/4$ uncorrupted rounds, the resulting cost is no more than $\frac{\alpha}{4} N_j F_j$. Since $j \le \log F$, $\rho_j = 2^{j-1}\lceil F_j/F \rceil$ and $F_j \le 2F$. Combining these we have

$$F_j \le F\sqrt{2^{2-j}\rho_j}$$

so that

$$\begin{aligned}
\frac{\alpha}{4} N_j F_j &\le \alpha N_{j-1} F_{j-1} \\
&\le \alpha N_{j-1} F \sqrt{2^{3-j}\rho_{j-1}} \\
&\le \alpha F \sqrt{N_{j-1} 2^{3-j}} \sqrt{N_{j-1}\rho_{j-1}} \\
&\le \alpha F \sqrt{2N_1} \sqrt{8T_j/3} \\
&\le \alpha \sqrt{128 L T_j F/3} \\
&\le 7\alpha \sqrt{L T_j F}\,.
\end{aligned}$$

□

Lemma 4.8.9. If $j > \log F$, the cost to either player due to uncorrupted rounds in Iteration j is at most

$$3\alpha T_{j-1}$$

Proof. When $j > \log F$, $F_j = \rho_j$ and by Lemma 4.8.6,

$$\frac{\alpha}{4} N_j F_j = \frac{\alpha}{4} N_j \rho_j \le \alpha N_{j-1}\rho_{j-1} \le \frac{8\alpha}{3} T_{j-1} \le 3\alpha T_{j-1}\,.$$

□

Lemma 4.8.10. The cost to the players from corrupted rounds in Iteration j is at most $4\alpha\sqrt{2LT_jF}$ if $j \le \log F$ and $2\alpha T_j$ otherwise.

Proof. Suppose there are k corrupted rounds. Then the cost to the players is $k\alpha F_j$, while the adversary's cost is $k\rho_j/2$. If $j \ge \log F + 1$, $F_j = \rho_j$ and we easily see that the players' cost is at most $2\alpha T$. When $j \le \log F$, since $k \le N_j$,

$$\begin{aligned}
k\alpha F_j &= \alpha \sqrt{k\rho_j F 2^{1-j}} \sqrt{N_j F_j} \\
&\le \alpha \sqrt{T_j F 2^{2-j}} \sqrt{2^j N_1 F} \\
&\le 2\alpha \sqrt{8 L T_j F}\,.
\end{aligned}$$

□

Collecting the various costs and noting that $T_j \le 64 T_{j-1}$, we see that for a suitably large constant γ, we have

Lemma 4.8.11. The total cost to the players from Iteration j is at most $\gamma\sqrt{LT_{j-1}\log L}$ if $j \leq \log F$ and γT_{j-1} otherwise.

4.8.2 Bob plays alone

After Alice's verified transcript has length at least L, in each subsequent round, she transmits her synchronization message, and then random bits to indicate her continued presence. Once Alice has left, there is silence on the channel. To corrupt this silence, the adversary must make it look like a corrupted synchronization message followed by random bits. Since a random string of length F_j has, on average, $F_j/2$ alternations of bits, Bob considers the string to represent silence if it has fewer than $F_j/3$ alternations. Thus, to corrupt such a round the adversary must pay at least $F_j/3$.

Alice leaves when she has received word that Bob has a verified transcript of length at least L, and a single extra uncorrupted round thereafter will cause Bob to leave as well. Thus, if iteration j was not Bob's last one, the adversary must have corrupted every round. If $1 \leq k < N_j$ rounds are corrupted, Bob pays at most $(k+1)\alpha F_j \leq 2k\alpha F_j$ and the adversary pays $kF_j/3$. If $k = 0$, we will generously account for the lone uncorrupted round from Iteration j in Iteration $j-1$ by noting that $\alpha(N_{j-1}F_{j-1} + F_j) \leq 2\alpha(N_{j-1}F_{j-1})$ Finally a calculation identical to that in Lemma 4.8.10 shows that Bob's cost for an iteration j that he played alone is no more than

$$\gamma\sqrt{LT_{j-1}\log L}$$

if $j < \log F$ and

$$\gamma T_{j-1}$$

otherwise.

4.8.3 Failure Probabilities

In this section we bound the probabilities of the events that cause the algorithm to fail.

Lemma 4.8.12. With high probability in L, there is no hash collision during Iteration j.

Proof. The fingerprint size has been selected large enough that the probability of a hash collision for a single hash is $\frac{1}{2^{2j}L^2}$. Since there are $N_j = 2^{j+2}L/F$ rounds in Iteration j, by a union bound, the probability of a hash collision during the iteration is $O\left(\frac{1}{2^j L\log L}\right)$. □

Lemma 4.8.13. With high probability in L, any bit flipping of an AMD encoded message during Iteration j is detected.

Proof. The size of the AMD encoding has been selected so that the probability of a failure to detect a single instance of tampering is $\frac{1}{2^{2j}L^2}$. Since there are two AMD encodings per round and $2^{j+2}L/F$ rounds, again the probability that such a failure occurs during the iteration is $O\left(\frac{1}{2^j L \log L}\right)$. □

Lemma 4.8.14. With high probability in L, Alice leaves before Bob.

Proof. Bob does not terminate until he thinks Alice has left, and he does not even start checking for whether she seems to have left until after his transcript has length at least L. Since Bob's transcript lags behind that of Alice, this means that by the time Bob is checking for whether Alice has left, Alice either really has left, in which case it is fine for Bob to leave, or she is transmitting i.i.d. random bits in batches of length F_j, between fingerprints. Since the adversary cannot see the bits, any bit flips on his part do not alter the fact that the string received by Bob is a uniformly random bit string of length F_j. Such a string has $F_j/2$ alternations (consecutive bits that differ) in expectation. Bob leaves if he sees fewer than $F_j/3$ alternations. If the string is random, the likelihood of Bob seeing fewer than $F_j/3$ alternations is, by Chernoff's bound, at most $e^{-F_j/18} \leq \frac{1}{2^{2j}L^2}$ provided $\beta = \frac{F_j}{2j+\log L}$ was chosen suitably large. Since there are at most N_j chances in Iteration j for the adversary to try this attack, a union bound again shows that Bob leaves after Alice, except with probability $O\left(\frac{1}{2^j L \log L}\right)$. □

4.8.4 Putting everything together

We will now prove our main theorem by putting all these costs together and calculating the total cost to either player and the failure probability of the algorithm. As before, T denotes the number of bits flipped by the adversary.

Theorem 4.8.15. The algorithm succeeds with probability at least $1 - 1/L\log L$. If it succeeds, then each player's cost is at most

$$L + O(\sqrt{LT\log L} + T)$$

Proof. First we note that for each $j \geq 0$ (Iteration 0 being Algorithm 1), the probability that Algorithm 3 fails during iteration j is at most $O\left(\frac{1}{2^{2j}L\log L}\right)$. Thus the overall probability that it fails at all is

$$O\left(\sum_{j=0}^{\infty} \frac{1}{2^j L \log L}\right) = O\left(\frac{1}{L\log L}\right)$$

Thus, with high probability the algorithm succeeds.

Let J denote the last iteration in which the player participates. If $J = 0$ then Lemma 4.6.24 already proves that the players' total cost is at most $L + O(\sqrt{L(T+1)\log L})$. Suppose $J \geq 1$. For each j, let $Cost(j)$ denote the player's cost from Iteration j. We know that

- $Cost(0) = 12L \leq L + \gamma\sqrt{LT_0 \log L}$ where $T_0 = L/(8F)$
- $Cost(j) \leq \gamma\sqrt{LT_{j-1}\log L}$ if $1 \leq j \leq \log F$
- $Cost(j) \leq \gamma T_{j-1}$ if $j > \log F$

When $J \leq \log F$, the player's total cost is

$$\begin{aligned}
\sum_{j=0}^{J} Cost(j) &\leq Cost(0) + \sum_{j=1}^{J} Cost(j) \\
&\leq L + \gamma\sqrt{LT_0\log L} + \sum_{j=1}^{J} \gamma\sqrt{LT_{j-1}\log L} \\
&\leq L + \gamma\sqrt{L\log L}\left(\sqrt{(2/3)^{J-1}T_{J-1}} + \sum_{j=1}^{J}\sqrt{(2/3)^{J-1-j}T_{J-1}}\right) \\
&\leq L + \gamma\sqrt{LT_{J-1}\log L}\left(\sqrt{(2/3)^{J-1}} + \sum_{j=0}^{J-2}\sqrt{(2/3)^{j}}\right) \\
&\leq L + \frac{\sqrt{3}\gamma}{\sqrt{3}-\sqrt{2}}\sqrt{LT_{J-1}\log L} \\
&= L + \gamma'\sqrt{LT_{J-1}\log L} \\
&\leq L + \gamma'\sqrt{LT\log L}
\end{aligned}$$

On the other hand, $T_{\lfloor \log F\rfloor} = \Theta(N_{\lfloor \log F\rfloor}\rho_{\lfloor \log F\rfloor}) = \Theta(L\log L)$, so that $\sqrt{LT_{\lfloor \log F\rfloor}\log L} = \Theta(T_{\lfloor \log F\rfloor})$ and for $J > \log F$ we have

$$\begin{aligned}
\sum_{j=0}^{J} Cost(j) &\leq Cost(0) + \sum_{j=1}^{\lfloor \log F\rfloor} Cost(j) + \sum_{j=\lfloor \log F\rfloor+1}^{J} Cost(j) \\
&\leq L + \gamma'\sqrt{LT_{\lfloor \log F\rfloor}\log L} + \sum_{j=\lfloor \log F\rfloor+1}^{J} \gamma T_{j-1} \\
&\leq L + \gamma'' T_{\lfloor \log F\rfloor} + \sum_{j=\lfloor \log F\rfloor+1}^{J} \gamma T_{j-1} \\
&\leq L + O(T)
\end{aligned}$$

Thus the players' cost is always $L + O\left(\sqrt{L(T+1)\log L} + T\right)$. □

4.9 Some Additional Remarks

Need for Private Channels

The following theorem justifies our assumption of private channels.

Theorem 4.9.1. Consider any algorithm for interactive communication over a public channel that works with unknown T and always terminates in the noise-free case. Any such algorithm succeeds with probability at most $1/2$.

Proof. The adversary chooses some protocol π with transcript length L and some separate "corrupted" protocol π_c such that 1) π_C has transcript length L and 2) Bob's individual input for π_c is equivalent to his individual input for π. The goal of the adversary will be to convince Bob that π_c is the protocol, rather than π. Note that we can always choose some appropriate pair π and π_c meeting the above criteria.

Assume that if π_c is the protocol and there is no noise on the channel, then Bob will output π_c with probability at least $1/2$; if not, then the theorem is trivially true. Then, the adversary sets π to be the input protocol. Next, the adversary simulates Alice in the case where her input protocol is π_c, and sets the bits received by Bob to be the bits that would be sent by Alice in such a case.

Since the the algorithm eventually terminates, Bob will halt after some finite number of rounds, X. Using the above strategy, Bob will incorrectly output π_c with probability at least $1/2$ and the value of T will be no more than X.

Note that in the above, we critically rely on the fact that T is unknown to Bob. □

Communication Rate Comparison.

In Haeupler's algorithm [Hae14], the noise rate ϵ is known in advance and is used to design an algorithm with a communication rate of $1 - O(\sqrt{\epsilon \log\log 1/\epsilon})$. Let L' be the length of π'. Then in his algorithm, $L' = O(L)$, and so the adversary is restricted to flipping $\epsilon L' = O(L)$ bits. Thus, in his model, T and L' are always $O(L)$. In our model, the values of T and L' are not known in advance, and so both T and L' may be asymptotically larger than L.

How do our results compare with [Hae14]? As noted above, a direct comparison is only possible when $T = O(L)$. Restating our algorithm in terms of ϵ, we have the following theorem.

Theorem 4.9.2. If the adversary flips $O(L)$ bits and the noise rate is ϵ then our algorithm guarantees a communication rate of $1 - O\left(\sqrt{\frac{\log L}{L}} + \sqrt{\epsilon \log L}\right)$.

Proof. When $T < L$ we also have $T < \sqrt{L(T+1)\log L}$ and our algorithm guarantees that for some $\gamma > 0$,

$$L' = L + \gamma\sqrt{L(T+1)\log L}$$

Let $\epsilon = T/L'$ and $R = L/L'$ be the effective noise and communication rates respectively. Then,

$$\begin{aligned} R = \frac{L}{L'} &= 1 - \frac{L' - L}{L'} \\ &\geq 1 - \frac{\gamma\sqrt{L(T+1)\log L}}{L'} \\ &\geq 1 - \gamma\frac{\sqrt{L\log L} + \sqrt{LT\log L}}{L'} \\ &\geq 1 - \gamma\left(\frac{\sqrt{R\log L}}{\sqrt{L'}} + \sqrt{R\epsilon\log L}\right) \\ &\geq 1 - \gamma\sqrt{\log L}\left(\frac{1}{\sqrt{L}} + \sqrt{\epsilon}\right), \end{aligned}$$

where the last line follows because $1/\sqrt{L'} \leq 1/\sqrt{L}$ and $R \leq 1$. □

We note that the additive term $\sqrt{\frac{\log L}{L}}$ arises from the fact that because we do not know the error rate ahead of time, we cannot get a communication rate of 1 even when the effective error rate turns out to be zero.

A Note on Fingerprint Size.

A natural question is whether more powerful probabilistic techniques than union bound could enable us to use smaller fingerprints as done in [Hae14]. The variability of block sizes poses a challenge to this approach since Alice and Bob must either agree on the current block size, or be able to recover from a disagreement by having Bob stay in the listening loop so he can receive Alice's message. If their transcripts diverge by more than a constant number of blocks, it may be difficult to make such a recovery, and therefore it seems challenging to modify our algorithm to use smaller fingerprints. However, it is a direction for further investigation.

A Lower Bound

In this section, we prove a lower bound that demonstrates the near optimality of our upper bound by assuming the following conjecture by Haeupler holds [Hae14]. We now restate Haeupler's conjecture.

Conjecture 1. *(*Haeupler [Hae14], 2014) The maximal rate achievable by an interactive coding scheme for any binary error channel with random or oblivious errors is $1 - \Theta(\sqrt{\epsilon})$ for a noise rate $\epsilon \to 0$. This also holds for for fully adversarial binary error channels if the adversary is computationally bounded or if parties have access to shared randomness that is unknown to the channel.

For the remainder of this section, we ***assume that Haeupler's conjecture holds*** for any algorithm that succeed with high probability in L with an expected cost of at most L' under adversarial noise. For ease of exposition, we omit such statements in all of our claims below. By *r*obust interactive communication, we mean interactive communication tolerates T errors.

We begin by showing the near optimality with respect to the communication rate achieved:

Theorem 4.9.3. Any algorithm for robust interactive communication must have $L' = L + \Omega\left(T + \sqrt{LT}\right)$ for some $T \geq 1$.

Proof. Let $T \geq 1$ be any value such that $T/L' = o(1)$. Then, Haeupler's Conjecture applies and the expected total number of bits sent is $L' \geq L/(1 - d\sqrt{\epsilon})$ for some constant $d > 0$. Noting that $1/(1 - d\sqrt{\epsilon}) \geq 1 + d\sqrt{\epsilon}$ by the well-known sum of a geometric series, this implies that $L' \geq L/(1 - d\sqrt{\epsilon}) \geq (1 + d\sqrt{\epsilon})L = (1 + d\sqrt{T/L'})L$ since $\epsilon = T/L'$.

This implies that $L/L' \leq 1/(1 + d\sqrt{T/L'})$. Now observe that $1/(1 + x) = 1/(1 - (-x)) \leq 1 - x + x^2$ for $|x| < 1$, again by the sum of a geometric series. Plugging in $d\sqrt{T/L'}$ for x, we have $1/(1 + d\sqrt{T/L'}) \leq 1 - d\sqrt{T/L'} + d^2(T/L')$. Therefore, $L/L' \leq 1 - d\sqrt{T/L'} + d^2(T/L') = 1 - d\sqrt{T/L'}(1 - d\sqrt{T/L'}) \leq 1 - d'\sqrt{T/L'}$ for some $d' > 0$ depending only on d.

We then derive: $L \leq L'(1 - d'\sqrt{T/L'}) = L' - d'\sqrt{L'T}$. It follows that $L' \geq L + d'\sqrt{L'T} = L + \Omega(\sqrt{LT})$ since $L' \geq L$.

Finally, we show that $\sqrt{LT} = \Theta(T + \sqrt{LT})$. Assume that given any algorithm A for interactive computation, we create a new algorithm A' that has expected value of $L' = O(L)$. To do this, A' checks based on ϵ and L whether or not Haeupler's algorithm [Hae14] will send fewer bits in expectation than A. If so it runs Haeupler's algorithm. Note that the expected number of bits sent by A' is no more than the expected number of bits sent by A.

Note that $T = \epsilon L'$ and for algorithm A', the expected value of $L' = O(L)$. This implies that implies that $T = \epsilon O(L)$ or $T = O(L)$. Since $T < L$, it holds that $\sqrt{LT} = \Theta(T + \sqrt{LT})$ which completes the proof. □

4.10 Conclusion

We have described the first algorithm for interactive communication that tolerates an unknown but finite amount of noise. Against an adversary that flips T bits, our algorithm sends $L + O\left(\sqrt{L(T+1)\log L} + T\right)$ bits in expectation where L is the transcript length of the computation. We prove this is optimal up to logarithmic factors, assuming a conjectured lower bound by Haeupler. Our algorithm critically relies on the assumption of a private channel, an assumption that we show is necessary in order to tolerate an unknown noise rate.

Several open problems remain including the following. First, can we adapt our results to interactive communication that involves more than two parties? Second, can we more efficiently handle an unknown amount of stochastic noise? Finally, for any algorithm, what are the optimal tradeoffs between the overhead incurred when $T = 0$ and the overhead incurred for $T > 0$?

Chapter 5

Conclusion and Open Problems

In this dissertation, we have studied secure distributed protocols in the unbounded adversarial model and the rational model. We designed a protocol that solves the MPC problem in polylogarithmic communication and computation cost and is secure against an adversary than can corrupt a third of the parties. We adapted our synchronous MPC protocol to the asynchronous setting when the fraction of the corrupted parties are less than 1/8. Additionally, we presented a scalable protocol that solves the secret sharing problem between rational parties in polylogarithmic communication and computation cost.

Furthermore, we presented a protocol that can solve the interactive communication problem over a noisy channel when the noise rate in unknown. In this problem, we have focused on the cost of the protocol in the resource-competitive analysis model. Unlike classic models, resource-competitive models consider the cost that the adversary must pay to succeed in corrupting the protocol.

In the rest of this chapter, we propose a few open problems related to the research done in this dissertation. Our final goal is to build a resource-competitive MPC that works over noisy channels. Each subsection is a step toward this goal.

5.1 Two Party Computation Over a Noisy Channel

Can we run a *secure two party computation (2PC)* over a noisy channels? Most 2PC papers assume the channel between the parties is noise-free. However, in practice that is not the case and there might be random or even adversarial noise over the channel. This means that we need a mechanism to simulate the 2PC protocol that is correct and secure over a noise-free channel such that it is also correct and secure over noisy channel. One simple method for that is to use error-correcting code for each message of the 2PC protocol. As we talked about it in Section 4, this method is not efficient

since we need to send at least $\log n$ extra bits for each message even if the message itself is only one bit.

The idea is how to adapt an efficient protocol for interactive communication such as our result in Section 4 to simulate a secure 2PC. Note that this method and similar methods critically rely on computing blocks of the protocol more than once. Since the overall picture of the algorithm is that compute one block of the protocol, if an error happens, compute the same block again. If there is no error, move to the next block.

However, if parties have a 2PC protocol that is correct over a noise-free channel, they cannot simply use a recent and efficient algorithm to simulate it over the noisy channel. This is because it might breach the privacy by computing some part of the 2PC protocol twice or more. Therefore, the interesting question is to see if we can extend our results for interactive communication while make sure we do not give up privacy. Our goal here is to design a 2PC protocol that works over noisy channel.

5.2 Cost-Competitive MPC Over Secure Channels

One interesting direction is to see if a standard MPC protocols can be analyzed in the cost-competitive setting. The cost to the adversary in MPC can be modeled as the cost of obtaining control over each node for each round. The goal is designing a protocol that costs more when the adversary is making many corruptions and costs less when the adversary remains passive. The difference between interactive computation and MPC is that in interactive computation, we assume parties are honest, but the channel is not reliable. In MPC, however, the computation is performed in the presence of dishonest parties but with secure channels. Thus, addressing privacy is the most challenging problem in this setting since it is not easy to detect when the adversary eavesdrops. We suggest two approaches for this problem.

Cheap Curiosity Versus Expensive Bysantine: The first approach is to design a MPC protocol that handles Byzantine nodes as efficiently as honest-but-curious nodes. Our intuition for this approach is that most algorithms cost less in dealing with honest-but-curious parties and addressing Byzantine faults is more expensive. To deal with the Byzantine case, the verification step is the most expensive part. The idea is to start the protocol with a weak but cheap verification scheme by assuming the parties are honest but curious in the start. Then parties can check if the other parties cheat or not. The hope is to develop an easy sub-protocol for this check. Parties keep a measure for how honest other parties are. Then, based on this honesty measure, they can run a more complicated

but more expensive method for verification. Our goal is to decrease the cost in practice or at-least have a protocol in which enforces the parties to remain honest by making it expensive for them to cheat.

Limited Communication Model: The second approach is to exploit the similarities between taking control over a node to do active corruptions and flipping bits of its channels, specifically when the degree of each nodes and the message size is logarithmic. This modeling can help us to reduce the problem to the interactive computation problem over noisy channels and use a parametric algorithms. We assume our algorithm spends α (a parameter) and we like to guarantee that any adversary must spend $\omega(\alpha)$ (e.g. α^3) to break privacy.

5.3 Cost-Competitive MPC over Noisy Channels

Finally, is it possible to build a scheme for MPC in the cost-competitive model, where the adversary is able to take control over both parties and channels? In this setting, the adversary's total cost is the cost of obtaining control over a party plus the cost of corrupting channels in each round. Generally speaking, solving MPC when the adversary has the power to corrupt the channels is a difficult task. Jain *et al.* [JKL11] show how to convert any n-party protocol into one that is resilient to a $1/n$-fraction of adversarial error while increasing the communication by only a constant factor. They prove that this result is optimal.

The difference between their work and what we propose here is threefold. First, the goal here is solving the problem not for any n-party protocol but for a specific MPC problem in mind. Designing a specialized solution might be easier than a generalized case. Second, we assume the channels are private, while Jain *et al.* consider a general noisy channel. As we showed in Section 4, privacy might be a crucial assumption here. Third, the focus of this problem is on the cost-competitive model, which probably allows the protocol to spend more resources if the adversary is willing to spend more.

Bibliography

[Abr74] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974.

[ADGH06] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 53–62. ACM, 2006.

[AHS91] James Aspnes, Maurice Herlihy, and Nir Shavit. Counting networks and multi-processor coordination. In *Proceedings of STOC'91*, pages 348–358. ACM, 1991.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An $0(n \log n)$ sorting network. In *Proceedings of STOC'83*, pages 1–9, New York, NY, USA, 1983. ACM.

[AL09] Gilad Asharov and Yehuda Lindell. Utility dependence in correct and fair rational secret sharing. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 559–576, Berlin, Heidelberg, 2009. Springer-Verlag.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*, page 14. John Wiley Interscience, March 2004.

[BCD+09] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. *Financial Cryptography and Data Security*, pages 325–343, 2009.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 52–61, New York, NY, USA, 1993. ACM.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation. Cryptology ePrint Archive, Report 2014/404, 2014.

[BE14] Mark Braverman and Klim Efremenko. List and Unique Coding for Interactive Communication in the Presence of Adversarial Noise. In *Foundations of Computer Science (FOCS)*, pages 236–245, 2014.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 1991.

[BGH13] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast Byzantine agreement. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 57–64, New York, NY, USA, 2013. ACM.

[BGT13] Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation: how to run sublinear algorithms in a distributed setting. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, TCC'13, pages 356–376, Berlin, Heidelberg, 2013. Springer-Verlag.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the Twentieth ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[BK12] Zvika Brakerski and Yael Tauman Kalai. Efficient Interactive Coding against Adversarial Noise. In *Foundations of Computer Science (FOCS)*, pages 160–166, 2012.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM.

[BN13] Zvika Brakerski and Moni Naor. Fast Algorithms for Interactive Coding. In *Symposium on Discrete Algorithms (SODA)*, pages 443–456, 2013.

[BR11] Mark Braverman and Anup Rao. Towards Coding for Maximum Errors in Interactive Communication. In *Symposium on Theory of Computing (STOC)*, pages 159–166, 2011.

[Bra12a] Mark Braverman. Coding for Interactive Computation: Progress and Challenges. In *Communication, Control, and Computing (Allerton)*, pages 1914–1921, 2012.

[Bra12b] Mark Braverman. Towards Deterministic Tree Code Constructions. In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 161–167, 2012.

[BTH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *Proceedings of the Advances in Crypotology 13th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'07, pages 376–392, Berlin, Heidelberg, 2007. Springer-Verlag.

[BW86] E Berlekamp and L Welch. Error correction for algebraic block codes, US Patent 4,633,470, December 1986.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications: Thesis*. PhD thesis, Weizmann Instituite of Science, 1995.

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

[Can01] Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, FOCS '01, pages 136–145, Oct 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[CCG+14] Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. Optimally resilient and adaptively secure multi-party computation with low communication locality. Cryptology ePrint Archive, Report 2014/615, 2014.

[CD89] B. Chor and C. Dwork. Randomization in Byzantine agreement. *Advances in Computing Research*, 5:443–498, 1989.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *Advances in Cryptology–EUROCRYPT 2008*, pages 471–488. Springer, 2008.

[CDG88] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 87–119, London, UK, UK, 1988. Springer-Verlag.

[CFGN96] R. Canetti, U. Friege, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. Technical report, Cambridge, MA, USA, 1996.

[CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In Yehuda Afek, editor, *Distributed Computing*, volume 8205 of *Lecture Notes in Computer Science*, pages 388–402. Springer Berlin Heidelberg, 2013.

[CL95] Jason Cooper and Nathan Linial. Fast perfect-information leader-election protocol with linear immunity. *Combinatorica*, 15:319–332, 1995.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, pages 42–51, 1993.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*, Irvine, pages 160–179, Berlin, Heidelberg, 2009. Springer-Verlag.

[DHM+15] Varsha Dani, Tom Hayes, Mahnush Mohavedi, Jared Saia, and Maxwell Young. Interactive Communication with Unknown Noise Rate. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 575–587, 2015.

[DI06] I. Damgård and Y. Ishai. Scalable secure multiparty computation. *Advances in Cryptology - CRYPTO 2006*, pages 501–520, 2006.

[DIK+08] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology – CRYPTO '08*, pages 241–261, 2008.

[DKMS12] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Brief announcement: breaking the $o(nm)$ bit barrier, secure multiparty computation with a static adversary. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, PODC '12, pages 227–228, New York, NY, USA, 2012. ACM.

[DKMS14] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2014.

[DMRS11] V. Dani, M. Movahedi, Y Rodriguez, and J. Saia. Scalable Rational Secret Sharing. In *Proceedings of the thirtieth annual ACM symposium on Principles of distributed computing*. ACM, 2011.

[DN07] I. Damgård and J.B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, pages 572–590. Springer-Verlag, 2007.

[Far88] R.W. Farebrother. *Linear Least Squares Computations*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1988.

[Fei99] Uriel Feige. Noncryptographic selection protocols. In *FOCS*, pages 142–153, 1999.

[FGOS15] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard Schulman. Optimal Coding for Streaming Authentication and Interactive Communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015.

[FHK14] Ofer Feinerman, Bernhard Haeupler, and Amos Korman. Breathe before speaking: efficient information dissemination despite noisy, limited and anonymous communication. In *Principles of Distributed Computing (PODC)*, pages 114–123. ACM, 2014.

[FKN10] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In *Proceedings of the 7th international conference on Theory of Cryptography*, TCC'10, pages 419–436, Berlin, Heidelberg, 2010. Springer-Verlag.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 148–161, New York, NY, USA, 1988. ACM.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[GH13] Mohsen Ghaffari and Bernhard Haeupler. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding, 2013. Available at: `http://arxiv.org/abs/1312.1763`.

[GHS14] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal Error Rates for Interactive Coding I: Adaptivity and Other Settings. In *Symposium on Theory of Computing (STOC)*, pages 794–803, 2014.

[GHY88] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 135–155, London, UK, UK, 1988. Springer-Verlag.

[GK06] S. Gordon and J. Katz. Rational secret sharing, revisited. *Security and Cryptography for Networks*, pages 229–241, 2006.

[GKLL09] R. Geambasu, T. Kohno, A.A. Levy, and H.M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th conference on USENIX security symposium*, pages 299–316. USENIX Association, 2009.

[GMS11] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and Explicit Coding for Interactive Communication. In *Foundations of Computer Science (FOCS)*, pages 768–777, Oct 2011.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

[Gol00] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 101–111, New York, NY, USA, 1998. ACM.

[Hae14] Bernhard Haeupler. Interactive channel capacity revisited. In *Foundations of Computer Science (FOCS)*, pages 226–235. IEEE, 2014.

[HKI+12] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *Information Security and Cryptology – ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 202–216. Springer Berlin Heidelberg, 2012.

[HT04] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, page 632. ACM, 2004.

[JKL11] Abhishek Jain, Yael Tauman Kalaiy, and Allison Lewkoz. Interactive coding for multiparty protocols, 2011.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM Journal on Computing*, 39(5):2090–2112, March 2010.

[KLST11] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable Byzantine agreement through quorum building with full information. In *Distributed Computing and Networking*, volume 6522 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2011.

[Klu95] Michael Richard Klugerman. *Small-depth Counting Networks and Related Topics*. PhD thesis, Cambridge, MA, USA, 1995. Not available from Univ. Microfilms Int.

[KN08] G. Kol and M. Naor. Games for exchanging information. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 423–432. ACM, 2008.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[KP92] Michael Klugerman and C. Greg Plaxton. Small-depth counting networks. In *Proceedings of STOC'92*, pages 417–428, 1992.

[KS10] V. King and J. Saia. Breaking the O(n^2) bit barrier: scalable byzantine agreement with an adaptive adversary. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 420–429. ACM, 2010.

[KSSV06a] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999, Philadelphia, PA, USA, 2006.

[KSSV06b] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 87–98, Washington, DC, USA, 2006. IEEE Computer Society.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. *J. ACM*, 53(6):881–917, November 2006.

[LT06] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. *Advances in Cryptology - CRYPTO 2006*, pages 180–197, 2006.

[MS14] Cristopher Moore and Leonard J. Schulman. Tree Codes and a Conjecture on Exponential Sums. In *Innovations in Theoretical Computer Science (ITCS)*, pages 145–154, 2014.

[MSZ15] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Scalable multi-party shuffling. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015)*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: randomized algorithms and probabilistic analysis*. Cambridge University Press, New York, 2005.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing (SICOMP)*, 22(4):838–856, 1993.

[ORS09] Rafail Ostrovsky, Yuval Rabani, and Leonard J. Schulman. Error-Correcting Codes for Automatic Control. *IEEE Transactions on Information Theory*, 55(7):2931–2941, 2009.

[Pec06] Marcin Peczarski. An Improvement of the Tree Code Construction. *Information Processing Letters*, 99(3):92–95, 2006.

[PSR02] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *INDOCRYPT 2002, Lecture Notes in Computer Science*, volume 2551, pages 93–107. Springer-Verlag, 2002.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[Sch92] L.J. Schulman. Communication on Noisy Channels: A Coding Theorem for Computation. In *Foundations of Computer Science (FOCS)*, pages 724–733, Oct 1992.

[Sch93] Leonard J. Schulman. Deterministic Coding for Interactive Communication. In *Symposium on Theory of Computing (STOC)*, pages 747–756, 1993.

[Sha48] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *INDOCRYPT 2000, Lecture Notes in Computer Science*, volume 1977, pages 117–129. Springer-Verlag, 2000.

[WC81] M.N. Wegman and J.L. Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[ZTW11] Yun Zhang, Christophe Tartary, and Huaxiong Wang. An efficient rational secret sharing scheme based on the chinese remainder theorem. In Udaya Parampalli and Philip Hawkes, editors, *Information Security and Privacy*, volume 6812 of *Lecture Notes in Computer Science*, pages 259–275. Springer Berlin Heidelberg, 2011.