5-1-2016

Scalable and Robust Distributed Algorithms for Privacy-Preserving Applications

Mahdi Zamani

Mahdi Zamani

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

Jedidiah Crandall

Bryan Ford

Trilce Estrada

Scalable and Robust Distributed Algorithms for Privacy-Preserving Applications

by

Mahdi Zamani

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 Bryan Ford at EPFL, and Professor Jed Crandall at UNM, and Professor Trilce Estrada at UNM for serving on my committee and providing invaluable comments and feedback on my dissertation.

I am also indebted to many of my colleagues and friends who supported me through all these years. I would like to thank Joud Khoury of BBN Technologies, Josh Karlin of Google, Valerie King of University of Victoria, and Jed Crandall of UNM for their collaboration and support in several of my Ph.D. projects. It has truly been a priviledge working with you.

Lastly, I want to give special thanks to my wife, Mahnush, who has always helped me get through those toughest times. Not only has she been my first source of support, but also she has been one of my best collaborators in many of my Ph.D. projects. I cannot find words to express my gratitude to her. I feel truly lucky to have her in my life.

Scalable and Robust Distributed Algorithms for Privacy-Preserving Applications

by

Mahdi Zamani

Abstract

We live in an era when political and commercial entities are increasingly engaging in sophisticated cyber attacks to damage, disrupt, or censor information content and to conduct mass surveillance. By compiling various patterns from user data over time, untrusted parties could create an intimate picture of sensitive personal information such as political and religious beliefs, health status, and so forth. In this dissertation, we study scalable and robust distributed algorithms that guarantee user privacy when communicating with other parties either to solely exchange information or to participate in multi-party computations.

We consider scalability and robustness requirements in three privacy-preserving areas: secure multi-party computation (MPC), anonymous broadcast, and blocking-resistant Tor bridge distribution. MPC is one of the most generic problems in fault-tolerant computation. In MPC, a set of parties, each having a secret value (input), 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 certain fraction of the parties are controlled by a malicious adversary who makes the parties deviate arbitrarily from the protocol.

In the past few years, several scalable solutions to MPC have been proposed. Unfortunately, most of these protocols are not practical due to reasons such as large hidden factors in their communication/computation complexities, non-constructive proofs, and load-balancing issues. Thus, system builders, when faced with a problem that falls under

the rubric of MPC, are unable to make use of established algorithms with well-known theoretical guarantees. Instead, they are frequently required to design new ad-hoc heuristics. In this dissertation, we address this problem by designing resource-efficient protocols for MPC.

In particular, we propose decentralized algorithms for MPC that have communication and latency costs that scale well with the number of parties and tolerate arbitrary faults from a computationally-unbounded adversary who makes a large fraction of parties arbitrarily deviate from the protocol. Our algorithms do not require any trusted party and are load-balanced, meaning that each party handles a roughly equal amount of communication and computation. Our algorithms are scalable since they require each party to send (and compute) a polylogarithmic number of bits (and operations). We provide experimental results to show that our protocols improve significantly over previous work. We also show that using common cryptographic tools in our protocol one might be able to achieve practical results for multi-party computation.

Anonymity is an essential tool for achieving privacy; it enables individuals to communicate with each other without being identified as the sender or the receiver of the information being exchanged. An anonymity scheme not only can be used for anonymous communication, but also can be used as a black-box in many other privacy-preserving applications such as location-based services, auctions, e-voting. We show that our MPC algorithms can be effectively used to design a scalable anonymous broadcast protocol.

One challenging attack on anonymity systems is *traffic-analysis*, where a global adversary maps messages to their senders and recipients by monitoring the traffic exchanged between parties. Unfortunately, well-known anonymous services such as Tor [DMS04] and Crowds [RR98] are not secure against this type of attack. Moreover, most schemes that tolerate traffic analysis scale poorly with the network size, rendering them impractical for large networks.

One technique for achieving tracking-resistant anonymity is to perform secure shuffling among the participants. Shuffling a sequence of values is a fundamental tool for randomized algorithms; applications include fault-tolerant algorithms, cryptography, and coding theory. In this dissertation, we describe the problem of *multi-party shuffling (MPS)*, where a group of parties, each holding an input, want to agree on a random permutation of their inputs while keeping the permutation secret. An MPS protocol not only is useful for anonymous communication, but is also a useful primitive for achieving privacy and robustness in electronic voting, private information retrieval, secure auctions, and general multi-party computation.

Our final approach for preserving user privacy in cyberspace is to improve Tor; one of the most popular censorship circumvention systems in the world. Tor deploys circumvention relay nodes run by volunteers outside the censored network to provide indirect access to blocked websites. It also deploys a special type of relay nodes called bridges to prevent censors from discovering and blocking all relay nodes. Unfortunately, recent studies [Din11b, WL12, EFW+15] show that censors are able to enumerate bridges by colluding with corrupt users inside the censorship network to discover and shut down bridges. We describe a randomized bridge distribution algorithm, where all honest users are guaranteed to connect to Tor in the presence of an adversary corrupting an unknown number of users. Our simulations suggest that, with minimal resource costs, our algorithm can guarantee Tor access for all honest users after a small (logarithmic) number of rounds.

Table of Contents

List of Figures

Chapter 1

Introduction

We live in an era when political and commercial entities are increasingly engaging in sophisticated cyber-attacks to damage, disrupt, or censor information content and to conduct mass surveillance. User data collected by companies and state-sponsored agencies are constantly mined for several purposes such as medical, financial, and advertising applications. Additionally, distributed networks, 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.

Modern data sets are so large that organizations frequently outsource their analytics activities to the cloud or highly-decentralized networks. Unfortunately, this creates several privacy challenges because untrusted parties can abuse the data and extract sensitive private information from customers. The data can be used by others (e.g., providers and governments) for precise surveillance and hence, compromising user privacy. In this dissertation, we study provably-robust algorithms that can be used for efficient privacy-preserving communications and computations in both cloud-based settings and multi-party settings.

1.1 Summary of Contributions

We study three areas of privacy-preserving algorithms: scalable multi-party computation, anonymous broadcast, and Tor bridge distribution. In the following, we briefly introduce each problem and state our contributions.

Scalable Multi-Party Computation. In *secure multi-party computation (MPC)*, a group of parties, each having a private value, collaborate with each other to compute a known function over their inputs, without revealing these inputs to each other. Previous results show that any computable functionality can be computed securely using an MPC protocol [BGW88, Can00, CLOS02]. MPC generalizes many important problems in distributed computing including classic problems such as auctions, threshold cryptography, voting, and privacy-preserving data mining; and contemporary problems such as cloud computing and computing over peer-to-peer networks.

Unfortunately, known MPC protocols are inefficient and scale poorly with the number of parties. We are not aware of an MPC algorithm that has practical costs for large number of parties. We believe this problem is of increasing importance with the growth of modern networks. For example, 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 perform data mining over data spread over large numbers of machines?

In this dissertation, we describe scalable protocols for solving the MPC problem among a large number of parties. We consider both the synchronous and the asynchronous communication models. In the synchronous setting, our protocol is secure against a static malicious adversary corrupting less than a $1/3$ fraction of the parties. In the asynchronous setting, we allow the adversary to corrupt less than a $1/8$ fraction of parties. For any deterministic function that can be computed by an arithmetic circuit with m gates, both of our protocols require each party to send a number of field elements and perform an amount of computation that is $\tilde{O}(m/n + \sqrt{n})$. We also show that our protocols provide perfect and universally composable security. We also provide simulation results showing that our protocol improves significantly over previous work. For example, for one million parties, when compared to the state of the art, our protocol reduces the communication and computation costs by at least three orders of magnitude and slightly decreases the number of communication rounds.

Anonymous Broadcast. Anonymity is an essential tool for achieving privacy in today's world and enables individuals to communicate with each other without fear of surveillance. The Internet, as the today's most popular communication mechanism, is not as anonymous as it might appear. Every message sent directly from a computer is tagged with an address that allows the recipient to send a response. This address, perhaps combined with some information from the service provider, can be used to identify the source. Communication that happens over higher-level protocols, such as email, typically contains information that allows a message to be tracked back to the sender.

One of the most important goals of this dissertation is to apply our efficient MPC algorithms to the problem of provably-secure and scalable *anonymous broadcast*, where a set of parties want to anonymously send their messages to all parties. A protocol for anonymous broadcast not only can be used for anonymous communication [Cha81], but also can be used as a black-box in many other privacy-preserving applications such as private information retrieval [CKGS98], secure auctions [FA00], and anonymous voting [Gro04].

In this dissertation, we design efficient algorithms for anonymous broadcast that resist traffic-analysis attacks. Moreover, unlike the majority of previous work which rely on centralized trusted servers, our algorithms are fully-decentralized and do not require any trusted party. Our protocols tolerate up to $n/3$ statically-scheduled Byzantine parties that are controlled by a computationally unbounded adversary. We provide simulation results to show that our protocol improves significantly over previous work. We finally show that using a common cryptographic tool in our protocol one can achieve practical results for anonymous broadcast.

The problem of anonymous broadcast is tightly related to the more general problem of *secure multi-party shuffling (MPS)*, where multiple parties, each holding an input, want to agree on a random permutation of their inputs while keeping the permutation secret. An MPS protocol is important as a primitive in many other privacy-preserving applications such as location-based services, electronic voting, and secure auctions. Unfortunately, known techniques for solving MPS suffer from poor scalability, load-balancing issues, trusted party assumptions, and/or weak security guarantees.

We customize our scalable MPC protocol to construct an unconditionally-secure protocol for MPS that scales well with the number of parties and is load-balanced. As an application of our algorithm, we study the problem of privacy preserving *location-based services (LBS)*. With rapid advances in mobile communication technologies, LBS are seamlessly and ubiquitously integrating into our lives. Unfortunately, this creates several privacy challenges because untrusted parties can abuse locational data and extract sensitive information about customers. Using our MPS algorithm, we design a protocol for privacy-preserving LBS that is highly scalable and fault-tolerant, and, unlike the majority of previous work, does not require any trusted party.

Blocking-Resistant Tor Bridge Distribution. Another approach for preserving user privacy in the Internet is to use existing anonymity networks such as Tor [DMS04]. Tor is a popular network that relays Internet traffic through a world-wide network of volunteer nodes. Tor users can connect to the network and have their Internet data routed through the network before reaching any server, thus the servers are not able to distinguish between Tor users or locate them. The relays are publicly-known and hence may be blocked by censorship systems. To make blocking harder, Tor uses the notion of bridges [DM06]; volunteer nodes across the globe that direct user traffic to the relays, but there is no complete public list of them. An important challenge with the current bridge distribution mechanism of Tor is that malicious users can still obtain information about a large number of bridges and block them. This, unfortunately, has been and is preventing many users across the world from connecting to Tor [DM06].

One of the most effective ways to prevent an adversary from taking over a large fraction of the bridges is to use randomization. This is because the colluding adversary cannot predict the behavior of the randomized process in distributing a set of bridges to the users, and thus he cannot arrange his corrupt users in such a way that prevents some of the honest users from connecting to Tor.

In this dissertation, we describe a randomized bridge distribution algorithm, where all (honest) users are guaranteed to be able to connect to Tor in the presence of an adversary corrupting an unknown number of users $t < n$, where n is the total number of users. Our algorithm adaptively increases the number of bridges according to the behavior of the adversary and requires $\tilde{O}(t)$ bridges. We show that, using our algorithm, the number of times a user fails to connect to Tor via bridges is bounded by $O(\log t)$ with high probability.

1.2 Notions of Security

In this section, we define various notions of security that describe the adversary's corruption abilities and, in turn, the set of assumptions required for our proofs of security to hold. In terms of its computational power, an adversary can be categorized as one of the following types:

- **Computationally-bounded.** An adversary who is capable of only polynomial-time computation. A model with this type of adversary provides computational security.
- **Computationally-unbounded.** An adversary who has *unknown* (not necessarily infinite) amounts of resources. Thus, any algorithm in this model cannot assume any bound on the adversary's computational powers and it must maintain its security with *any* amount of computational power the adversary may have.

In terms of corruption abilities, an adversary can be categorized as one of the following types:

- **Fail-stop (*a.k.a.,* non-adversarial fault).** The adversary can only cause the parties to crash randomly.
- **Passive (*a.k.a.,* semi-honest or honest-but-curious).** The adversary can read the internal state of the corrupted players and their communication, trying to obtain some information he is not entitled to. This is called *eavesdropping*. Passive model is the weakest adversarial model.
- **Active (*a.k.a.,* malicious or Byzantine).** The adversary can additionally make the corrupted parties deviate from the protocol specification in order to falsify the outcome of the protocol. Such This is the strongest adversarial model.
- **Semi-malicious.** The adversary follows the protocol execution (similar to a semi-honest adversary), but can choose its random coins (and inputs) in any arbitrary manner.
- **Covert.** The adversary can make the corrupted parties deviate arbitrarily from the protocol specification, but only if this deviation cannot be *detected* [AL10]. Covert model is similar to the malicious model except in that it assumes the adversary and all its corrupted parties will be detected once at least one corrupted party is detected so the adversary is limited to only *covert* (hidden) attacks. One disadvantage of this model is that it cannot distinguish between fail-stop faults and non-participation attacks (if possible).

In terms of when the set of corrupted parties are chosen, an adversary can be categorized as one of the following types:

- **Static.** The adversary is restricted to choose its set of dishonest parties at the start of the protocol and cannot change this set later on.
- **Adaptive.** The adversary can choose its set of dishonest parties at any moment during the execution of the protocol.

With respect to the adversary's computational power, the security provided by a secure algorithm can be categorized as one of the following:

- **Information-theoretic security (perfect security).** An algorithm is perfectly-secure if given a ciphertext, every message in the message space is exactly as likely to be the plaintext, *i.e.*, the plaintext is independent of the ciphertext. This implies that even a computationally-unbounded adversary cannot learn anything about the plaintext. Such an algorithm is not vulnerable to future developments of quantum computing.

- **Unconditional security.** An algorithm is unconditionally-secure if it makes no assumption about the resources (computing power, memory, etc) available to the adversary. From an information-theoretic perspective, such a system is considered information-theoretically secure (or perfectly-secure).
- **Computational (cryptographic) security.** The information-theoretic notion of security is sometimes too strong (and hence inefficient) to be useful in practice. Computational security asks only that no polynomial-time adversary can tell which of the messages that could potentially be plaintexts corresponding to a ciphertext is more likely than the other to be the actual plaintext. In other words, the key space is big enough to make brute-force attacks impossible for such an adversary.

A security proof model describes the required assumptions for the proof of security of a secure algorithm to hold. The following are the most useful security proof models:

- **Strongest security model.** An algorithm is *secure in the strongest model* if it does not make any assumption for its proof of security. In other words, it proves that given whatever information and computational power an adversary desires, it is incapable of breaking the scheme even in the slightest way.
- **Standard model (*a.k.a.,* plain model).** An algorithm is *secure in the standard model* if the only assumption it makes for its proof of security is the computational power of the adversary (called the standard assumption).
- **Random oracle model.** A random oracle is an abstract black-box that generates truly random numbers. An algorithm is *secure in the random oracle model* if it assumes the existence of a random oracle for its proof of security.
- **Common reference string (CRS) model.** An algorithm is secure in the CRS model if it assumes that all parties have access to a common random string taken from a predetermined distribution.
- **Public key infrastructure (PKI) model.** An algorithm is secure in the PKI model if it assumes identities of all parties can be reliably verified using techniques of public-key cryptography (PKC). In other words, an algorithm secure in the PKI model assumes that the channels are already authenticated via PKC.

Chapter 2

Scalable Multi-Party Computation

2.1 Introduction

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.

2.1.1 Our Contribution

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.

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].

2.1.2 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.

2.1.3 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.

Dani *et al.* [DKMS12] introduced the notion of using quorums to decrease message cost in MPC. 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.

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.

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.

2.1.4 Proof of Security Techniques

The first step in proving the security of a cryptographic protocol is to define what *secure* means. In this section, we describe two standard definitions of security commonly used for proving security of multi-party protocols.

2.1.4.1 Simulation Paradigm

The *simulation paradigm* (*a.k.a.*, the *real/ideal model*) is a security proof technique first described by Goldreich *et al.* in [GMW87]. In this paradigm, the security of a protocol is proven by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario (*simulation*), which is secure by definition. In the ideal scenario, there is an incorruptible trusted party to whom the parties send their inputs. 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.

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	$\tilde{O}(n)$	$\tilde{\Omega}(n)$ + $\tilde{\Omega}(\kappa m d^3)^\dagger$	$\tilde{O}(1)$	$O(n\ell \cdot \text{polylog}(n))$	No
[BCP14]	Perfect	$(1/3-\epsilon)n$	No	Yes	$\text{poly}(n) + \tilde{O}(Time(\Pi))$	$\text{poly}(n) + \tilde{O}(Time(\Pi))$	$\tilde{O}(Time(\Pi))$	$O(\ell)$	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	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d + \text{polylog}(n))$	$O(\ell)$	Yes
Our result (async)	Perfect	$(1/8-\epsilon)n$	Yes	No	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d + \text{polylog}(n))$	$O(\ell)$	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.

A protocol is *secure* if any adversary in the real model (where no trusted party exists) can do no more harm than if it was involved in the ideal model [Gol00, Section 4.3]. To prove this, an entity called a *simulator* is defined in the ideal model. The simulator is an algorithm that simulates the interaction between each (possibly adversarial) party and the trusted party (*i.e.*, the functionality).

2.1.4.2 Universal Composability

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 (see [Gol04] Section 7.7.2). 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*.

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 $\mathcal{P}_1, ..., \mathcal{P}_\ell$ are all UC-secure, and $\mathcal{P}$ is UC-secure in the hybrid model, then $\mathcal{P}$ is UC-secure in the real model.

2.2 Perfectly-Secure MPC

Before describing our protocol, we define the standard tools and notation used in the protocols of this section.

2.2.1 Preliminaries

Notation. We denote the set of integers $\{1, ..., n\}$ 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 2.2.3.

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 paper, 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 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 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 paper, 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 $\mathsf{polylog}(n)$.

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*.

2.2.2 Our Protocol

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.

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.

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

Quorum Building. Creating a sufficient number of quorums. King *et al.* [KLST11] describe a randomized protocol called Build-Quorums that achieves this goal with high probability.

Circuit Evaluation. Securely evaluating each gate over secret-shared inputs by the parties inside a quorum. In Section 2.2.2.2, 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 2.2.2.2.

Protocol 2.1 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).

Algorithm 2.1 Perfectly-Secure 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 $Q_{2i+1}$.
```

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

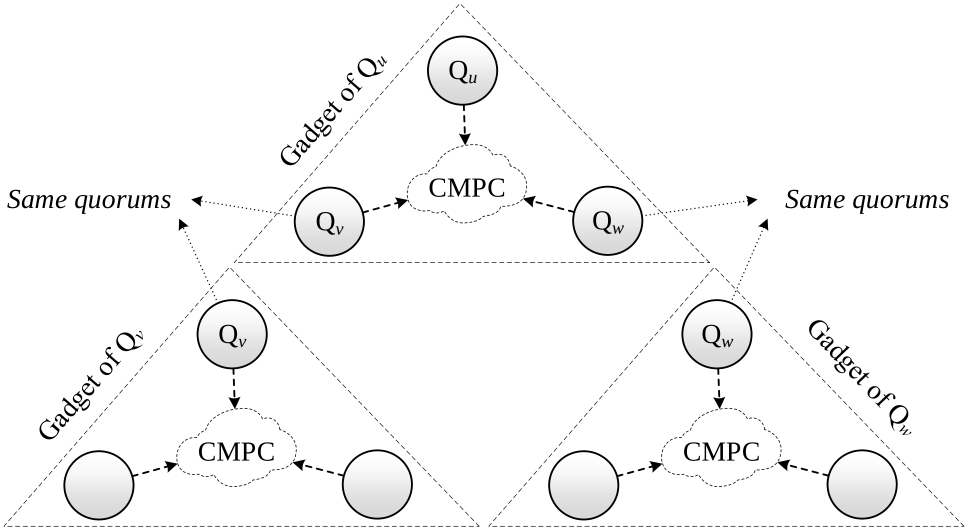


Figure 2.1: The gate gadgets for gate u and its left and right children

$Q_{(i \bmod n)}$. 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 $\widehat{y}_u = y_u + r_u$.

2.2.2.1 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.

2.2.2.2 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

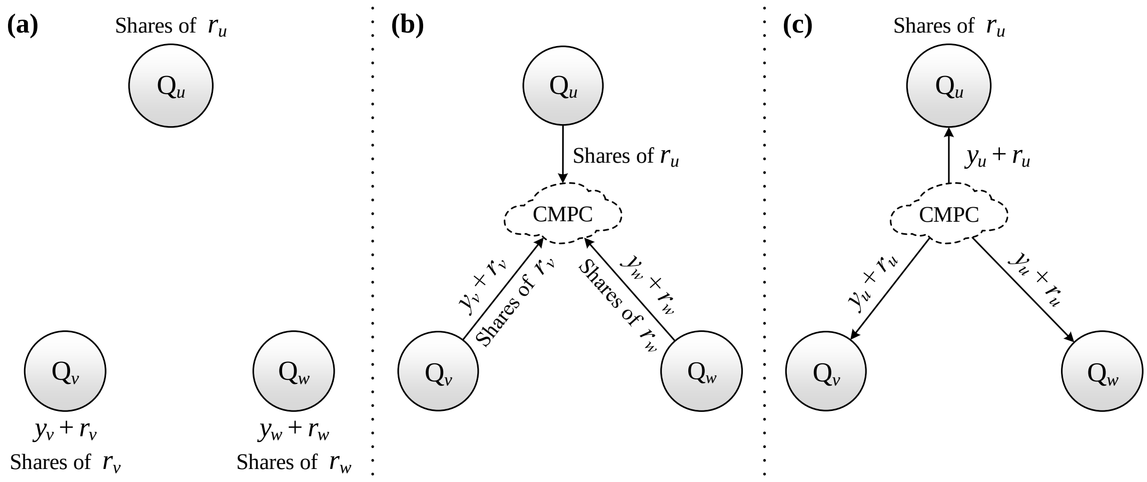


Figure 2.2: Evaluation of gate u: (a) generating r_u, (b) providing inputs to CMPC, (c) receiving the masked outputs

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. 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.

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

Algorithm 2.2 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 $r_w)$.

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.

2.2.2.3 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.

Algorithm 2.3 Gen-Rand

Goal. A set of parties $P_1, \dots, P_N$ 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 $\rho_{1j}, \dots, \rho_{N'j}$ be these shares. P_j computes $r_j = \sum_{k=1}^{N'} \rho_{kj}$.

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 $\mathbb{F}_p$ 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 $E(x) = e_0 + e_1 x + \dots + e_\varepsilon x^\varepsilon$ of degree ε, and $Q(x) = q_0 + q_1 x + \dots + q_k x^k$ 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 $e_0, \dots, e_\varepsilon, q_0, \dots, q_k$ 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 $O(n^3)$.

2.2.3 Security Proof

We first describe the UC framework in Section 2.2.3.1, and then give a sketch of our proof in Section 3.4.4.2. We prove the UC-security of Protocol 2.1 in sections 2.2.3.3 to 2.2.3.5. Finally, we calculate the resource costs of this protocol in Section 2.2.4.

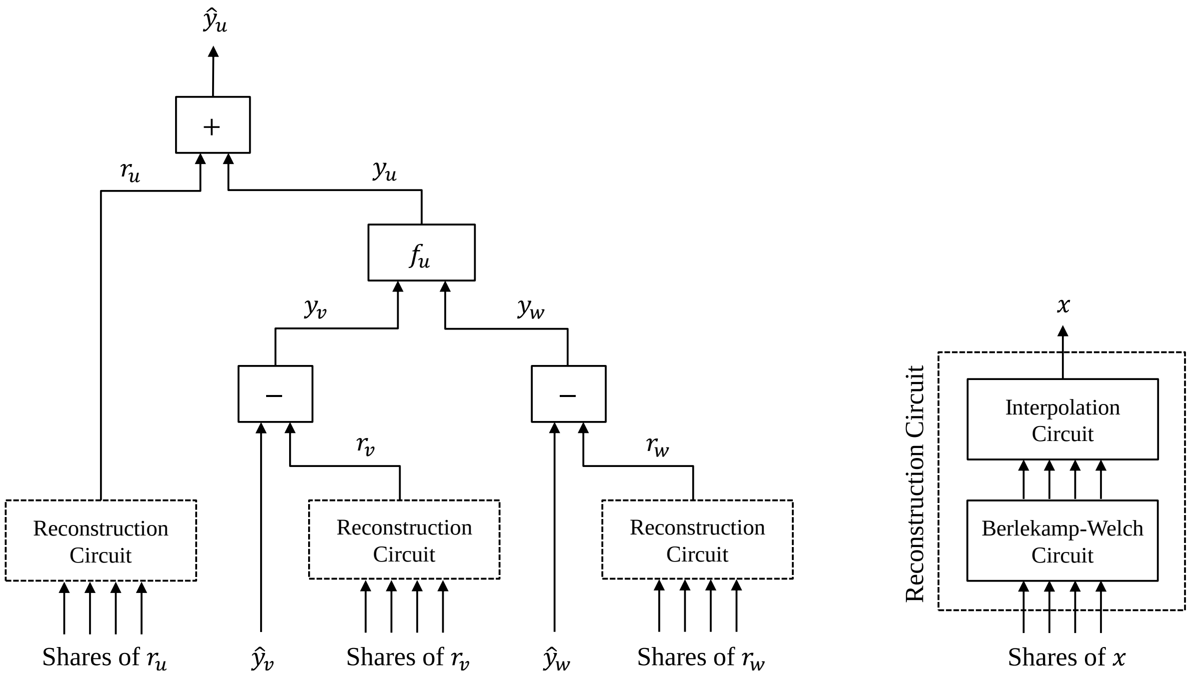


Figure 2.3: Circuit of gate *u*

2.2.3.1 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 $\mathcal{P}_1, ..., \mathcal{P}_\ell$ are all UC-secure, and $\mathcal{P}$ is UC-secure in the hybrid model, then $\mathcal{P}$ is UC-secure in the real model.

2.2.3.2 Proof Sketch

Before proceeding to the proof, we remark that the error probability in Theorem 10 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.[2]

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 2.1. 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 5, we show Step 2, that the adversary can not distinguish the execution of the hybrid model from the ideal model.

[2] 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
$F_{\mathsf{VSS\text{-}Share}}$	Protocol VSS-Share
$F_{\mathsf{VSS\text{-}Reconst}}$	Protocol VSS-Reconst
F_{CMPC}	Protocol CMPC
$F_{\mathsf{Gen\text{-}Rand}}$	Protocol Gen-Rand
F_{Input}	Input Commitment stage of Protocol 2.1
$F_{\mathsf{Circuit\text{-}Eval}}$	Protocol Circuit-Eval
F_{Output}	Output Reconstruction and Output Propagation stages of Protocol 2.1

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 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 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 3.2 that correspond to the subprotocols used in Protocol 2.1. We then prove that Protocol 2.1 is t-secure in the (F_{BA}, $F_{\mathsf{VSS\text{-}Share}}$, $F_{\mathsf{VSS\text{-}Reconst}}$, F_{CMPC}, $F_{\mathsf{Gen\text{-}Rand}}$, F_{Input}, $F_{\mathsf{Circuit\text{-}Eval}}$, F_{Output})-hybrid model. Finally, we use Theorem 1 to infer the UC-security of Protocol 2.1.

In order to prove the t-security of Protocol 2.1 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 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.[3] The view of the adversary is then defined as the combined view of all corrupted parties.

2.2.3.3 Security of Input Commitment

The ideal functionality F_{Input} creates a set S containing the index of the parties whose inputs have been accepted 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 1, we show the Input Commitment stage in Protocol 2.1 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 1. *The Input Commitment stage of Protocol 2.1 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}. 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.

We prove the t-security of the Input Commitment stage in the ($F_{\mathsf{VSS\text{-}Share}}$, $F_{\mathsf{VSS\text{-}Reconst}}$,$F_{\mathsf{CMPC}}$)-hybrid model which is similar to the Input Commitment stage of Protocol 2.1 except that every call to its subprotocols is replaced with a call to their corresponding functionality. We define the corresponding simulator $\mathcal{S}_{\mathsf{Input}}$ in Protocol 2.4.

Let V_1 denote the view of the adversary from the hybrid execution, and V_2 be its view from the simulation. Since our simulator is straight-line and black-box, it follows from Theorem 1 that the Input Commitment stage is UC-secure. □

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

Algorithm 2.4 $\mathcal{S}_{\mathsf{Input}}$

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 $F_{\mathsf{VSS\text{-}Share}}$ 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 $F_{\mathsf{VSS\text{-}Share}}$ 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.

2.2.3.4 Security of Circuit Evaluation

We first prove the security of Gen-Rand. The ideal functionality $F_{\mathsf{Gen\text{-}Rand}}$ is given in Protocol 3.4. 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.

Algorithm 2.5 $F_{\mathsf{Gen\text{-}Rand}}$

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 $F_{\mathsf{VSS\text{-}Share}}$ to send a share r_i of r to P_i.

Lemma 2. *The protocol Gen-Rand is UC-secure.*

Proof. We prove the t-security of Gen-Rand in the $F_{\mathsf{VSS\text{-}Share}}$-hybrid model which is similar to Protocol 3.3 except that every call to VSS-Share is replaced with a call to the ideal functionality $F_{\mathsf{VSS\text{-}Share}}$. The corresponding simulator $\mathcal{S}_{\mathsf{Gen\text{-}Rand}}$ is given in Protocol 3.5.

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 $\mathcal{S}_{\mathsf{Gen\text{-}Rand}}$ generates

Algorithm 2.6 $\mathcal{S}_{\textsf{Gen-Rand}}$

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 $F_{\textsf{VSS-Share}}$ with dummy input 0. Let $s_1^i, ..., s_N^i$ denote the outputs.
2. For every $P_j \in I_u$,
    a) Run $F_{\textsf{VSS-Share}}$ 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.

the shares from dummy input 0 instead of actual inputs. Since $F_{\textsf{VSS-Share}}$ 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 $F_{\textsf{Circuit-Eval}}$ is given in Protocol 2.7.

Algorithm 2.7 $F_{\textsf{Circuit-Eval}}$

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 $F_{\textsf{Gen-Rand}}$ with inputs $\rho_1, ..., \rho_N$ to generate $r_u^{(1)}, ..., r_u^{(N)}$.
3. Run $F_{\textsf{CMPC}}$ to locally compute the following functionality:
    a) $r_u \leftarrow F_{\textsf{VSS-Reconst}}$ over $r_u^{(1)}, ..., r_u^{(N)}$.
    b) $r_v \leftarrow F_{\textsf{VSS-Reconst}}$ over $r_v^{(1)}, ..., r_v^{(N)}$.
    c) $r_w \leftarrow F_{\textsf{VSS-Reconst}}$ 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) $\widehat{y}_u \leftarrow f_u(y_1, y_2) + r_u$

Lemma 3. *The protocol Circuit-Eval is UC-secure.*

Proof. We first show that for each gate $u \in C$, $F_{\textsf{Circuit-Eval}}$ correctly computes $\widehat{y}_u = y_u + r_u$. Based on $F_{\textsf{Input}}$ and $F_{\textsf{Gen-Rand}}$, 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 $F_{\textsf{VSS-Reconst}}$ 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 $F_{\textsf{Input}}$, 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 $F_{\textsf{CMPC}}$. We now argue based on the definition of the function computed by $F_{\textsf{CMPC}}$ that the output of $F_{\textsf{CMPC}}$ 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 $1/3$ 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 ($F_{\textsf{Gen-Rand}}$,$F_{\textsf{CMPC}}$)-hybrid model which is similar to Protocol 2.2 except that every call to CMPC and Gen-Rand is replaced with a call to $F_{\textsf{CMPC}}$ and $F_{\textsf{Gen-Rand}}$ respectively. The corresponding simulator $\mathcal{S}_{\textsf{Circuit-Eval}}$ is given in Protocol 2.8.

Algorithm 2.8 $\mathcal{S}_{\textsf{Circuit-Eval}}$

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 $F_{\textsf{Gen-Rand}}$ 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 $F_{\textsf{CMPC}}$ to compute the functionality defined in Line 3 of $F_{\textsf{Circuit-Eval}}$ with the following inputs: the input of every party in $I_\triangle$ as described in $F_{\textsf{Circuit-Eval}}$, 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.

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 $F_{\mathsf{Gen\text{-}Rand}}$, 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 $F_{\mathsf{Gen\text{-}Rand}}$. 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 $F_{\mathsf{Gen\text{-}Rand}}$. 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 $\mathcal{S}_{\mathsf{Circuit\text{-}Eval}}$ is straight-line and black-box,

Circuit-Eval is UC-secure. □

2.2.3.5 Security of Output Stages

The ideal functionality for the Output Reconstruction and the Output Propagation stages of Protocol 2.1 are given in Protocol 2.9.

Algorithm 2.9 F_{Output}

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

Functionality:

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

Lemma 4. *The Output Reconstruction and Output Propagation stages of Protocol 2.1 are UC-secure.*

Proof. We first show that the two stages correctly compute F_{Output}. Let z be the output gate of C. By Lemma 3, 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 $1/3$ 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.

We now prove the t-security of the output stages in the $F_{\mathsf{VSS\text{-}Reconst}}$-hybrid model. The corresponding simulator $\mathcal{S}_{\mathsf{Output}}$ is given in Protocol 2.10. 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. □

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

Algorithm 2.10 $\mathcal{S}_{\mathsf{Output}}$

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 $F_{\mathsf{VSS\text{-}Reconst}}$ 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 $i \in \{2, \ldots, n\}$, 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}.

Lemma 5. *Protocol 2.1 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 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 2.1 is t-secure since in lemmas 1, 3, and 4 we showed that all stages of the Protocol 2.1 are t-secure. Based on Theorem 1, since we have proved the t-security of Protocol 2.1 using a straight-line black-box simulator, the protocol is also UC-secure. □

2.2.4 Cost Analysis

We now analyze the resource costs of Protocol 2.1.

Lemma 6. *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. Since each quorum has $\log n$ parties, an additional $\mathsf{polylog}(n)$ messages are sent by each quorum during VSS-Share and VSS-Reconst to check whether the input is correctly secret-shared. □

Lemma 7. *If all honest parties follow Protocol 2.1, then with high probability, each party sends at most $\tilde{O}(m/n + \sqrt{n})$ messages.*

Proof. By Theorem 4, 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 6, 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 $\mathsf{polylog}(n)$ 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 $\mathsf{polylog}(n)$ messages per node it represents. It follows that each party sends $\tilde{O}(m/n + \sqrt{n})$ messages. □

Lemma 8. *If all honest parties follow Protocol 2.1, with high probability, the total latency is $O(d\,\mathsf{polylog}(n))$ where d is depth of the circuit the protocol computes.*

Proof. Based on Theorem 4, the latency for creating quorums is $\mathsf{polylog}(n)$ which implies that the Input Commitment stage also has $\mathsf{polylog}(n)$ 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. □

2.2.5 An Efficient Technique for Share Renewal

In our MPC protocol, we solved the share renewal problem by masking the result in the current quorum and unmasking it in next quorum. However, we believe simple constructions of this method can be expensive in terms of communication and computation costs because it relies on the existence of an unmasking circuit securely evaluated by parties in the first quorum. Such a circuit must implement an error-correcting technique which requires many multiplication gates. Boyle *et al.* [BGT13] overcome this problem by sending their encrypted inputs to only one quorum which does all of the computation using FHE. Unfortunately, this results in large computation and communication costs for parties in that quorum. In this section, we design a simple and efficient method for share renewal by relaxing the resiliency requirement from $t < n/3$ to $t < n/6$.

For simplicity of our presentation, we let N denote the number of parties in each quorum. In linear secret sharing, to secret-share a value, the dealer picks a random polynomial $\phi(x) = \sum_{i=0}^{t} \phi_i x^i$ of degree t such that $\phi(0) = s$, and sends each party P_i a share $s_i = \phi(i)$. We represent each secret-shared value s by $\langle s \rangle = (s_1, ..., s_n)$ meaning that each party P_i holds a share s_i generated by the VSS scheme during its sharing phase. Using the natural component-wise addition of representations, we define $\langle a\rangle + \langle b\rangle = \langle a + b\rangle$. For multiplication, we define $\langle a\rangle \cdot \langle b\rangle = \mathsf{Multiply}(\langle a\rangle, \langle b\rangle)$, where Multiply is a protocol defined later in this section. The gate evaluation algorithm of our MPC protocol is modified in such a way that for every addition over two shared values $\langle a\rangle$ and $\langle b\rangle$ performed above, parties computes $\langle c\rangle = \langle a\rangle + \langle b\rangle$. For every multiplication, they run $\langle c\rangle = \mathsf{Multiply}(\langle a\rangle, \langle b\rangle)$. Parties run protocol RenewShares described later in this section over $\langle y\rangle$ and $\langle y'\rangle$ to re-share them in the quorum associated with the parent gate.

The high-level idea is to first generate a random polynomial that passes through the origin, and then add it to the polynomial that corresponds to the shared secret. The result is a new polynomial that represents the same secret but has coefficients that are chosen randomly and completely independent of the coefficients of the original polynomial. Combined with the VSS scheme of Katz *et al.* [KKK08] in a group of n parties with $t < n/3$ dishonest parties, this protocol has one round of communication and requires each party to send $O(n)$ field elements.

This idea was first proposed by Ben-Or *et al.* [BGW88]. The solution provided in [BGW88] requires a zero-knowledge proof, where each party is asked to prove distribution of shares over a polynomial with zero free-coefficient. Unfortunately, such a proof is either round-expensive (as in [BGW88]) or requires a weaker adversarial model for the problem to be solved efficiently (*e.g.*, see [HJKY95]). On the other hand, by relaxing the resiliency bound by only one less dishonest party, we can generate a random polynomial that passes through the origin without requiring the zero-knowledge step.

Let $\phi(x)$ be the original polynomial. The idea is to first generate a random polynomial $\rho(x)$ of degree $\mathsf{deg}(\phi) - 1$, and then compute a new polynomial $\phi_0(x) = x \cdot \rho(x)$ that is of degree $\mathsf{deg}(\phi)$ and passes through the origin. Finally, the fresh polynomial is computed from $\phi(x) + \phi_0(x)$. The polynomial $\rho(x)$ can be simply generated by asking parties to agree on a secret-shared uniform random value (using the protocol Gen-Rand described in [ZMS14]) over a random polynomial of degree $\mathsf{deg}(\phi) - 1$. Figure 2.4 depicts this idea for the special case of $d = 1$.

Theorem 1. *Let Q and Q' be two quorums of size N, where Q holds a shared value $\langle s\rangle = (s_1, ..., s_N)$ over a polynomial ϕ of degree $d = N/3$. There exists a protocol that can generate a new shared value $\langle s'\rangle = (s'_1, ..., s'_N)$ in Q' such that $s' = s$. The protocol is secure against a computationally-unbounded Byzantine adversary corrupting less than a* 1/6 *fraction of the parties in each quorum.*

The secure multiplication protocol (denoted by Multiply) is based on a well-known

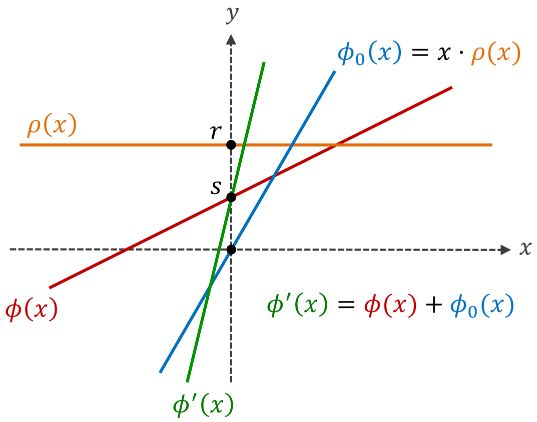

y
$\phi_0(x) = x \cdot \rho(x)$
$\rho(x)$
r
s
x
$\phi(x)$
$\phi'(x) = \phi(x) + \phi_0(x)$
$\phi'(x)$

Figure 2.4: Share renewal technique [ZMS14]

technique proposed by Beaver [Bea91]. The technique generates a shared multiplication triple $(\langle u\rangle, \langle v\rangle, \langle w\rangle)$ such that $w = u \cdot v$. The triple is then used to convert multiplications over shared values to additions. The only difference between Multiply and Beaver's multiplication method is that Beaver generates shared random elements u and v on polynomials of degree d and multiplies them to get a polynomial of degree $2d$ for w. Then, a degree reduction algorithm is run to reduce the degree from $2d$ to d. Instead, we choose polynomials of degree $d/2$ for u and v to get a polynomial of degree d for w. In our protocol, since we require less of 1/6 fraction of the parties be dishonest in each quorum, we can do this without revealing any information to the adversary. We note that the first step of Multiply is independent of the inputs and thus, can be performed in an offline phase to generate a sufficient number of multiplication triples.

Algorithm 2.11 Multiply

Usage. Initially, parties hold two shared values $\langle a\rangle$ and $\langle b\rangle$ that are on a polynomial of degree $d = N/3$. The protocol computes a shared value $\langle c\rangle$ such that $c = a \cdot b$.

Multiply($\langle a\rangle, \langle b\rangle$) :

1. Parties run Gen-Rand to generate two shared random values $\langle u\rangle = (u_1, ..., u_N)$ and $\langle v\rangle = (v_1, ..., v_N)$ both on polynomials of degree $d/2$. Then, each party P_i computes $w_i = v_i \cdot u_i$.
2. Each party P_i computes $\varepsilon_i = a_i + u_i$ and $\delta_i = b_i + v_i$ and runs Reconst(ε_i) and Reconst(δ_i) to learn ε and δ. Party P_i computes $c_i = w_i + \delta a_i + \varepsilon b_i - \varepsilon\delta$.

Clearly, ε and δ can be safely revealed to all parties so that each party can compute $\varepsilon\delta$ locally. The only difference between Multiply and Beaver's multiplication method is that Beaver generates shared random elements u and v on polynomials of degree d and multiplies

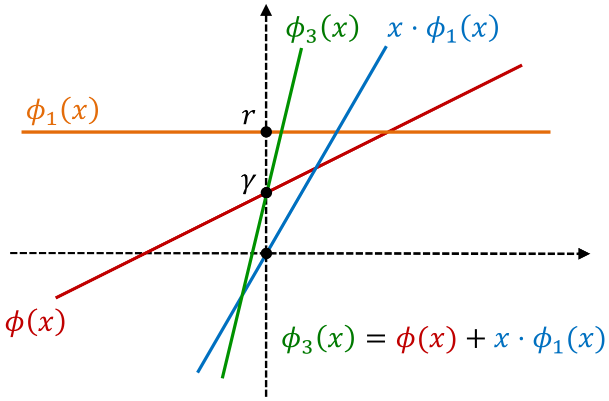


Figure 2.5: Share renewal technique

them to get a polynomial of degree $2d$ for w. Then, a degree reduction algorithm is run to reduce the degree from $2d$ to d. Instead, we choose polynomials of degree $d/2$ for u and v to get a polynomial of degree d for w. In our protocol, since we require less than $N/6$ of parties be dishonest in each quorum, we can do this without revealing any information to the adversary. We note that the first step of Multiply is independent of the inputs and thus, can be performed in an offline phase to generate a sufficient number of multiplication triples.

Algorithm 2.12 Renew-Shares

Usage. Let Q be the quorum associated with gate G in the circuit, and Q' be the quorum associated with a parent of G. Initially, parties in Q hold a sharing $\langle s \rangle = (s_1, ..., s_N)$ of a secret $s \in \mathbb{Z}_p$ over a random polynomial $\phi(x)$ of degree $d = N/3$. Using this protocol, parties in Q jointly generate a *fresh* sharing of s in Q'.

RenewShares($\langle s \rangle, Q'$):

For all $i \in [N]$,

1. Party $P_i \in Q$ runs Gen-Rand to jointly generate a sharing $\langle r \rangle = (r_1, ..., r_N)$ of a uniform random value $r \in \mathbb{Z}_p$ over a polynomial $\rho(x)$ of degree $d - 1$.
2. $P_i \in Q$ computes $s'_i = s_i + i \cdot r_i$, and sends s'_i to party $P_i \in Q'$.

2.2.5.1 Security Proof

Definition 1. **[t-secrecy]** A sharing defined over a polynomial ϕ is said to have *t-secrecy* if

1. it is *t-private* meaning that no set of at most t parties can compute $\phi(0)$, and

2. it is *t-resilient* meaning that no set of t or less parties can prevent the other $n - t$ remaining parties from correctly reconstructing $\phi(0)$.

Proposition 1. *The Shamir's secret sharing scheme [Sha79] has the following properties:*

1. *A sharing defined over a polynomial ϕ of degree d has d-secrecy if the adversary knows less than d points on ϕ.*

2. *Let ϕ_1 and ϕ_2 be independent random polynomials of degree d_1 and d_2 that correspond to sharings with d_1-secrecy and d_2-secrecy respectively. $\phi_3 = \phi_1 + \phi_2$ is a polynomial of degree $d_3 = \max(d_1, d_2)$ that corresponds to a sharing with d_3-secrecy.*

3. *Let ϕ_1 and ϕ_2 be polynomials of degree d that correspond to two sharings both with d-secrecy. $\phi_3 = \phi_1 \cdot \phi_2$ is a polynomial of degree $2d$ that corresponds to a sharing with d-secrecy.*

Proof. The first property can be easily observed from Definition 1 because in order to uniquely reconstruct a polynomial of degree d, at least $d + 1$ points are required. Since the adversary knows less than d points on ϕ, and all elements of $\mathbb{Z}_p$ are equally likely to be the missing point(s), the adversary cannot uniquely reconstruct the polynomial and thus, he learns nothing about the secret. The second property is correct due to the linearity of Shamir's secret sharing scheme. Without loss of generality, let $d_1 \leq d_2$. Intuitively, if we assume that the sharing defined by ϕ_3 does not have d_2-secrecy. Then, parties compute $\phi_2 = \phi_3 - \phi_1$. Thus, they can find $\phi_2(0)$ (or similarly, prevent others from learning $\phi_2(0)$). This contradicts with the fact that ϕ_2 corresponds to a sharing with d_2-secrecy. For a complete proof, we refer the reader to Claim 3.4 of [AL11]. The third property is correct because considering an arbitrary party P_i holding two shares a_i and b_i on polynomials ϕ_1 and ϕ_2 respectively, P_i learns nothing from $a_i \cdot b_i$ other than what is revealed from a_i and b_i since P_i computes it with no interaction with other parties. So, the resulting shared value also has d-secrecy. □

Corollary 1. *If a shared value has d-secrecy, then it also has d'-secrecy, where $d' < d$.*

The correctness of Multiply is already shown by Beaver in [Bea91]. The security of Multiply directly follows from Proposition 1 described above and Lemma 1 proved below.

Lemma 1. *Let $\langle a \rangle$ and $\langle b \rangle$ be two secret-shared values both with $N/3$-secrecy and $\langle c \rangle =$ Multiply$(\langle a \rangle, \langle b \rangle)$. The sharing $\langle c \rangle$ has $N/3$-secrecy.*

Proof. In the first step of Multiply, algorithm Gen-Rand creates two shared values $\langle u \rangle = (u_1, ..., u_N)$ and $\langle v \rangle = (v_1, ..., v_N)$ that correspond to degree $N/6$ polynomials. Lemma 1 proves that $\langle u \rangle$ and $\langle v \rangle$ both have $N/6$-secrecy. For each party P_i, based on Lemma 1, $w_i = u_i \cdot v_i$ defines a new sharing that is on a polynomial of degree $N/3$ and has $N/6$-secrecy. Moreover, $\langle a \rangle$ and $\langle b \rangle$ both are on polynomials of degree $N/3$ and have $N/3$-secrecy. Thus, based on Lemma 1, $c_i = w_i + \delta a_i + \varepsilon b_i - \varepsilon\delta$ defines a new sharing $\langle c \rangle$ that is on a polynomial of degree $N/3$ and has $N/3$-secrecy. □

We first prove the correctness of Renew-Shares. Figure 2.5 shows a sketch of the proof. From the correctness of Gen-Rand, $r_i \in \mathbb{Z}_p$ is a share of a global random value $r \in \mathbb{Z}_p$. Let $\phi_o(x) = x \cdot \rho(x)$. Clearly, $\mathsf{deg}(\phi_o) = \mathsf{deg}(\rho) + 1 = d$. Now, define $\phi'(x) = \phi(x) + \phi_o(x)$. Since $\phi(0) = s$ and $\phi_o(0) = 0$, we have $\phi'(0) = s$. Hence, each party P_i can locally compute a new share of s denoted by s'_i from $s'_i = s_i + i \cdot r_i$, where $s'_i = \phi'(i)$, $s_i = \phi(i)$, and $i \cdot r_i = \phi_o(i)$. We now prove the security of Renew-Shares.

Lemma 2. *Let ϕ_1 and ϕ_2 be two polynomials. $\phi_2(x) = x \cdot \phi_1(x)$ if and only if $\phi_2(0) = 0$.*

Proof. If $\phi_2(x) = x \cdot \phi_1(x)$, then $\phi_2(0) = 0$. Assuming $d = \mathsf{deg}(\phi_2)$, we write $\phi_2(x) = a_0 + a_1 x + ... + a_d x^d$. If $\phi_2(0) = 0$, then $a_0 = 0$ and there exists a polynomial $\phi_1(x) = a_1 + ... + a_d x^{d-1}$ such that $\phi_2(x) = x \cdot \phi_1(x)$. □

Lemma 3. *Let $\langle s \rangle$ and $\langle s' \rangle$ be two shared values defined over random polynomials ϕ_1 and ϕ_2 respectively, where $\phi_2(x) = x \cdot \phi_1(x)$. If $\langle s \rangle$ has t-secrecy, then $\langle s' \rangle$ also has t-secrecy.*

Proof. Let d_1 and d_2 be the degrees of ϕ_1 and ϕ_2 respectively. Clearly, $d_2 = d_1 + 1$. Let S_t be a set of at most $t \leq d_1$ points on ϕ_1 the adversary learns via a coalition of at most t malicious parties. By Lemma 2, the only information the adversary learns about ϕ_2 is a set of at most $t+1$ points $S_t \cup \{(0,0)\}$ on ϕ_2. Since $t + 1 \leq d_2$, by Proposition 1 and Corollary 1, $\langle s' \rangle$ has t-secrecy. □

Theorem 2. Let Q and Q' be two quorums of size N, where Q holds a shared value $\langle s \rangle = (s_1, ..., s_N)$ over a polynomial ϕ of degree $d = N/3$. The protocol Renew-Shares generates a new shared value $\langle s' \rangle = (s'_1, ..., s'_N)$ in Q' such that $s' = s$. The protocol is secure against a Byzantine adversary corrupting $T < N/6$ parties in each quorum.

Proof. Let $\rho(x)$ be the polynomial associated with $\langle r \rangle = (r_1, ..., r_N)$ generated in the first step of Renew-Shares. Since ϕ has degree $d = N/3$, based on Corollary 1, $\langle s \rangle$ has d-secrecy. Since $\mathsf{deg}(\rho) = \mathsf{deg}(\phi) - 1 = d - 1$, the sharing $\langle r \rangle$ has $(d-1)$-secrecy. By Corollary 1,

$x \cdot \rho(x)$ defines a new sharing with $(d-1)$-secrecy. Finally, by Proposition 1, the protocol computes a new sharing $\langle s' \rangle = (s'_1, ..., s'_N)$ with $(d-1)$-secrecy, where $s'_i = s_i + i \cdot r_i$. Based on the correctness of Renew-Shares shown proved before, $s' = s$. Since $T < N/6$ parties are dishonest in Q and Q', via a coalition of at most $2T < N/3$ dishonest parties in Q and Q', the adversary can learn up to $2T$ shares of $\langle s \rangle$. Moreover, since $\langle s \rangle$ and $\langle s' \rangle$ both have $(N/3-1)$-secrecy, the last step of the protocol reveals nothing about s. Based on the correctness and secrecy of Gen-Rand invoked in the first step of Renew-Shares, r and its shares are uniformly random and independent from any other random value generated in the protocol. For all $i \in [N]$, since r_i is independent of s_i, we conclude that s'_i is independent of s_i. □

We now prove the secrecy of our modified circuit evaluation step by induction. The adversary cannot obtain any information about the inputs and outputs during the computation of each gate of the circuit. Let Q, and Q' be two quorums involved in the computation of a gate, where Q provides an input to the gate, and Q' computes the gate. Consider a party P. Let S be the set of all shares P receives during the protocol. We consider two cases. First, if $P \notin (Q \cup Q')$, then elements of S are independent of the shares Q sends to Q'. Moreover, elements of S are independent of the output of Q' since Q' also re-shares its output(s). Hence, S reveals nothing about the inputs and outputs of the gate.

Second, if $P \in (Q \cup Q')$, then the inductive invariant is that the collection of all shares held by dishonest parties in Q and Q' does not give the adversary any information about the inputs and the outputs. As the base case, it is clear that the invariant is valid for input gates. The induction step is as follows. The adversary can obtain at most $2(N/6) = N/3$ shares of any shared value during the computation step; $N/6$ from dishonest parties in Q and $N/6$ from dishonest parties in Q'. By the secrecy of the VSS scheme, at least $N/3 + 1$ shares are required for reconstructing the secret. By the secrecy of RenewShares and Multiply, when at most $N/3$ of the shares are revealed, the secrecy of the computation step is proved using universal computability of multi-party protocols.

2.3 Computationally-Secure MPC

In this section, we present our cryptographic protocol for general MPC. Before describing our protocol, we define standard tools and notation used in the protocol.

2.3.1 Preliminaries

Notation. An event occurs *with high probability*, if it occurs with probability at least $1 - 1/n^c$, for any $c > 0$ and all sufficiently large n. A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is said to be *negligible* if $\epsilon(k) < 1/k^c$, for any $c > 0$ and all sufficiently large k. A problem with solution $\mathcal{S}$ is *computationally intractable* with respect to the security parameter κ, if for every adversary $\mathcal{A}$ given information I, the probability $Pr[\mathcal{A}(I) = \mathcal{S}] = \epsilon(\kappa)$. We denote the set of integers $\{1, ..., n\}$ by $[n]$. Let $\mathbb{Z}_p$ denote the additive group of integers modulo a prime p, $\mathbb{Z}_p[x]$ denote all integer polynomials[4] in variable x modulo p, and $\mathbb{Z}_p^*$ denote the multiplicative group of integers modulo p. Throughout this chapter, we assume g is a generator of the multiplicative group $\mathbb{G}$ of prime order p.

2.3.1.1 Computationally-Secure VSS

In our protocol, we use the cryptographic VSS scheme of Kate et al. [KZG10] called *eVSS* (stands for efficient VSS), which is based on Shamir's scheme [Sha79] and proposes a commitment scheme that is computationally-hiding under the *Discrete Logarithm (DL) assumption* and computationally-binding under the *t-Strong Diffie-Hellman (t-SDH) assumption*. Since eVSS generates commitments over elliptic curve groups, it requires smaller message sizes than other DL-based VSS scheme such as [GRR98].

Definition 2. [DL Assumption] Given $g, g^x \in \mathbb{G}$, computing x is computationally intractable.

Definition 3. [t-SDH Assumption] Let $a \in \mathbb{Z}_p^*$. Given $g, g^a, g^{(a^2)}, ..., g^{(a^t)} \in \mathbb{G}$, finding $c \in \mathbb{Z}_p$ and $g^{1/(a+c)}$ is computationally intractable.

Definition 4. [t-polyDH Assumption] Let $a \in \mathbb{Z}_p^*$. Given $g, g^a, g^{(a^2)}, ..., g^{(a^t)} \in \mathbb{G}$, finding $\phi(x) \in \mathbb{Z}_p[x]$ and $g^{\phi(a)}$ is computationally intractable.

Boneh and Boyen [BB04] and Kate et al. [KZG10] show that solving t-SDH and t-polyDH problems with $t < O(\sqrt[3]{p})$ each requires $\Omega(\sqrt{p/t})$ in expected time.

Theorem 3. [KZG10] *There exists a synchronous linear (n, t)-VSS scheme for $t < n/2$ that is secure against a computationally-bounded static adversary under the DL, t-SDH, and t-polyDH assumptions. In worst case, the protocol requires two broadcasts and four rounds of communication.*

[4] i.e., polynomials with integer coefficients.

The eVSS scheme consists of a setup algorithm denoted by VSetup that generates an algebraic structure and a public-private key pair required for the protocol. VSetup can be either run by a trusted party or a distributed authority [KZG10].

2.3.1.2 Fully Homomorphic Encryption

A *fully homomorphic encryption (FHE)* scheme allows to perform non-interactive secure computation, which is very useful in designing communication efficient MPC protocols. Gentry [Gen09] proposed the first FHE scheme, which is based on the hardness of lattice problems and is very computationally intensive. Although in the past few years the efficiency of FHE schemes has been improved by several orders of magnitude [vDGHV10, BGV12, GHS12a], current FHE schemes are still far from being used in practice.

The impracticality of current FHE schemes is primarily due to noise management techniques (like bootstrapping) that are used to deal with a noise term in ciphertexts that increases slightly with homomorphic addition and exponentially with homomorphic multiplication. On the other hand, if the circuit has a sufficiently small multiplicative depth, then it is possible to use the current FHE schemes in practice without using the expensive noise management techniques. Such a scheme is sometimes called *somewhat homomorphic encryption (SHE)*, which requires significantly less amount of computation than an FHE with noise management.

Similar to Damgard et al. [DPSZ12], we use SHE in a setup phase to generate multiplication triples that can be used later to perform secure multiplication in constant number of rounds. In this technique, SHE is only used to evaluate circuits of depth one and only in the setup phase. We use the fast FHE scheme of Brakerski-Gentry-Vaikuntanathan (BGV) [BGV12] that is based on *ring learning with error (R-LWE) assumption* and provides an effective approach for controlling the noise level of ciphertexts. LWE [Reg05] is a post-quantum lattice problem that asks to recover a secret given a sequence of approximate random linear equations on the secret. R-LWE is a special case of LWE with practical key sizes and yet strong hardness guarantees [LPR10]. In order to make homomorphic computations of BGV faster, we use the ciphertext packing technique of Smart and Vercauteren [SV11] and the optimizations of Gentry et al. [GHS12b].

Theorem 4. [BGV12] *There exists a fully homomorphic encryption public key cryptosystem $\mathcal{E}^{(D)} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ secure under the R-LWE assumption and against a semi-honest adversary such that $\mathcal{E}^{(D)}$ is homomorphic for all circuits of depth at most D, and all* *algorithms of the scheme have computation complexity polynomial in the depth and size of the circuit and the security parameter*[5].

In the case of a malicious adversary, a threshold FHE (TFHE) scheme is required, where is replaced with a protocol TGen for agreeing on a common public key as well as a sharing of the secret key and Dec is replaced with a threshold decryption protocol TDec. In this chapter, we use the TFHE scheme of Asharov et al. [AJLA+12] that is based on the FHE construction of BGV.

Theorem 5. [AJLA+12] *There exists a threshold fully homomorphic encryption public key cryptosystem* $\mathcal{TE}^{(D)} = (\mathsf{TGen}, \mathsf{Enc}, \mathsf{TDec}, \mathsf{Eval})$ *secure under the R-LWE assumption and against a static malicious adversary corrupting* $t \leq n$ *parties such that* $\mathcal{TE}^{(D)}$ *is homomorphic for all circuits of depth* $\leq D$.

2.3.2 Our Protocol

Let f be a deterministic function computed by an arithmetic circuit C of depth d and depth $|C|$. Let $G_1, \ldots, G_{|C|}$ denoted the gates of C, where instead of only + or ×, each gate $G_\ell \in \mathfrak{C}$ for $\ell \in [|C|]$ can compute an arithmetic circuit C_{G_ℓ} that has at most two inputs and at most two outputs[6]. In C, every gate with indegree zero is called an *input gate*, and every gate with outdegree zero is called an *output gate*. Let $G_1, \ldots, G_n$ be the input gates. Consider n parties $P_1, P_2, \ldots, P_n$ with inputs $x_1, \ldots, x_n \in \mathbb{Z}_p$, who want to jointly evaluate C over their inputs. Let denote the number of gates in C. Our main algorithm proceeds as follows.

[5] Such a scheme is also called a *leveled* FHE scheme with d be the maximum number of levels in the circuit.

[6] An *arithmetic circuit* is a directed acyclic graph, where every node with indegree zero is called an *input gate* and every other gate is labeled by either + (called a *addition gate*) or × (called a *multiplication gate*).

Algorithm 2.13 Computationally-Secure MPC

1. **Setup:**

   a) *Quorum Building*: Parties run the quorum building algorithm to agree on n quorums $Q_1, ..., Q_n$. For all $\ell \in [|C|]$, gate G_ℓ is assigned to $Q_{(\ell \bmod n)}$.

   For all $i \in [N]$ and $j \in [n]$, party $P_i \in Q_j$ performs the following:

   (b) *Key Generation*: P_i runs algorithms TGen and VSetup.

   c) *Triple Generation*: P_i runs $\mathsf{InitTriple}$ to jointly create a sufficient number of multiplication triples (u_i, v_i, w_i).

2. **Input Commitment:** For all $i \in [n]$, party P_i secret shares its input in Q_i associated with input gate G_i.

3. **Circuit Evaluation:** The circuit C is evaluated level-by-level starting from the input gates. For each gate G in C, the associated quorum Q_G computes C_G in the following way. Let $\langle\alpha\rangle = (\alpha_1, ..., \alpha_N)$ and $\langle\beta\rangle = (\beta_1, ..., \beta_N)$ be the sharings associated with the inputs of G. For each gate g in C_G, and each party $P_i \in Q_G$,

   a) if g is an addition gate, then P_i computes $\gamma_i = \alpha_i + \beta_i$,

   b) if g is a multiplication gate, then P_i runs $\gamma_i = \mathsf{Multiply}(\alpha_i, \beta_i)$,

   c) if g is the output gate of C_G with output value γ_i , then P_i runs $\mathsf{RenewShares}(\gamma_i, Q_{G'})$ for the quorum associated with G', which is the corresponding parent of G.

4. **Output Propagation:** Each party in each quorum associated with an output gate runs $z = \mathsf{Reconst}(\gamma_i)$, and then runs $\mathsf{Output}(z)$.

In the rest of this section, we define various algorithms used in Algorithm 2.13. Due to space limitations, we only give the proof of some of the algorithms in this section. The rest of the proofs can be found in Section 2.3.3.

2.3.2.1 Setup Phase

In our protocol, each gate G of the circuit is assigned a quorum that is in charge of computing the function associated with G. Consider a quorum Q of $N = O(\log n)$ parties $P_1, P_2, ..., P_N$ with inputs $\alpha_1, ..., \alpha_N \in \mathbb{Z}_p$ respectively, who want to jointly compute a circuit C_G corresponding to a gate G of C, while ensuring no parties learn anything about the inputs other than what is revealed from the output of the circuit. Throughout this section, we assume all parties belong to Q unless otherwise stated. Also, we use the notation C_a to denote an encryption of a plaintext $a \in \mathbb{Z}_p$ ciphered by the encryption algorithm of Theorem 5, meaning that $C_a = \mathsf{Enc}(a)$.

As described in Algorithm 2.13, inputs of G are securely shared in Q using VSS scheme, and the gate is evaluated over these secret-shared inputs. If G is an addition gate, then each party simply computes a share of the output by adding the input shares. However, if G is a multiplication gate, then the product of input shares is not necessarily a valid share of the product. We address this by generating a set of multiplication triples for each quorum using FHE in the setup phase similar to [DPSZ12]. As a result of this procedure, each party $P_i \in Q$ holds a sufficient number of multiplication triples (u_i, v_i, w_i), where u_i and v_i are shares of uniform random values $u, v \in \mathbb{Z}_p$, and w_i is a share of $w = u \cdot v$. These shares are all computed using TFHE such that no party learns anything about u , v , and w beyond their own shares. Algorithm InitTriple implements the triple generation functionality.

Algorithm 2.14 InitTriple

Usage. Each party P_i jointly computes a triple (u_i, v_i, w_i), where $\langle u\rangle = (u_1, \ldots, u_N)$, $\langle v\rangle = \langle v_1, \ldots, v_N\rangle$, and $\langle w\rangle = (w_1, \ldots, w_N)$, where $u, v \in \mathbb{Z}_p$ are chosen uniformly at random, $w = u \cdot v$.

InitTriple():

1. For all $i \in [N]$, party P_i chooses values $a_i, b_i \in \mathbb{Z}_p$ uniformly at random, and broadcasts the pair $(\mathsf{Enc}(a_i), \mathsf{Enc}(b_i))$.
2. Let $\{(C_{a_j}, C_{b_j})\}_{j=1}^{N}$ be the set of pairs P_i receives from the previous step[7]. P_i computes

$$C_u = \sum_{j=1}^{N} C_{a_j},\ C_v = \sum_{j=1}^{N} C_{b_j}, \text{ and } C_w = C_u \cdot C_v.$$

   Parties runs $\mathsf{CipherShare}(C_u)$, $\mathsf{CipherShare}(C_v)$, and $\mathsf{CipherShare}(C_w)$ to generate three sharings $\langle C_u\rangle$, $\langle C_v\rangle$, and $\langle C_w\rangle$.
3. For all $i \in [N]$, party P_i runs $u_i = \mathsf{DecPrivate}(C_{u_i})$, $v_i = \mathsf{DecPrivate}(C_{v_i})$, and $w_i = \mathsf{DecPrivate}(C_{w_i})$.

[7] Throughout this section, if the party receives less than N messages, it assumes a default value (in this case 0) for unreceived messages. Clearly, the party always receives at least $2t$ messages from honest parties.

Algorithm 2.15 CipherShare

Usage. Initially, all parties in Q hold a common ciphertext C_u. Using the algorithm, parties jointly convert C_u into a sharing $\langle C_u \rangle = (C_{u_1}, ..., C_{u_N})$, where u_i is a VSS share of $u \in \mathbb{Z}_p$.

CipherShare(C_u):

For all $i \in [N]$,

1. Party P_i runs Gen-Rand to jointly generate a sharing $\langle r \rangle = (r_1, ..., r_N)$ of a uniform random value $r \in \mathbb{Z}_p$ over a polynomial of degree $N/3 - 1$.
2. P_i computes $C_{u_i} = C_u + \mathsf{Enc}(i \cdot r_i)$. The party sends its share C_{u_i} to all parties in Q via one-to-one communication.
3. Let $C_{u_1}, ..., C_{u_N}$ be the messages P_i receives from the previous step. P_i runs the Welch-Berlekamp algorithm to recover the correct polynomial $\phi'(x)$ of degree $N/3$. For all $j \in [N]$, if $\phi'(j) \neq C_{u_j}$, then P_i concludes that P_j is dishonest, and ignores its share C_{u_j}.

We now prove the correctness of CipherShare. From the correctness of Gen-Rand, r_i is a share of a global random value $r \in \mathbb{Z}_p$. Let $\phi_1(x) \in \mathbb{Z}_p[x]$ be a random polynomial of degree $N/3 - 1$ such that $r_i = \phi_1(i)$. Using $\phi_1(x)$, we define a new polynomial $\phi_2(x) \in \mathbb{Z}_p[x]$ such that $\phi_2(x) = x \cdot \phi_1(x)$. Clearly, $\phi_2(x)$ has degree $N/3$ and passes through the origin. Finally, we use $\phi_2(x)$ to define a new polynomial $\phi_3(x) \in \mathbb{Z}_p[x]$ such that $\phi_3(x) = w + \phi_2(x)$. It is clear that $\phi_3(x)$ passes through the point $(0, w)$, and for all $i \in [N]$, $w_i = \phi_3(i)$ is a valid VSS share of w. Thus, using the homomorphic properties of Enc,

$$\mathsf{Enc}(w) + \mathsf{Enc}(i) \cdot \mathsf{Enc}(u_i) = \mathsf{Enc}(w + i \cdot u_i) = \mathsf{Enc}(w + \phi_1(i)) = \mathsf{Enc}(w_i).$$

The third step is correct because the Welch-Berlekamp algorithm can be represented as an arithmetic circuit and thus, can be computed over cipher inputs using the homomorphic properties of the TFHE scheme.

Algorithm 2.16 DecPrivate

Usage. Initially, all parties in Q hold a common ciphertext C_u. Using this algorithm, parties in Q jointly decrypt C_u for a party $P_j \in Q$. Initially, each party $P_i \in Q$ holds a share sk_i of the joint secret key $sk = \sum_{i=1}^{N} sk_i$ created by TGen during the setup phase of the protocol.

DecPrivate(C_u):

1. For all $i \in [N]$, party $P_i \in Q$ sends the pair (C_u, w_i) to party $P_j \in Q$, where w_i is calculated using C_u and sk_i as in the first step of TDec (see algorithm TFHE.Dec of [AJLA+12]).
2. Let $\{(C_u^{(1)}, w_1), ..., (C_u^{(N)}, w_N)\}$ be the set of pairs party $P_j \in Q$ receives from the previous step. From $\{C_u^{(1)}, ..., C_u^{(N)}\}$, party P_j chooses the element with majority as C_u, and computes the output using C_u and $w_1, ..., w_\ell$ as in the second step of TDec (algorithm TFHE.Dec in [AJLA+12]).

Note on Reconstruction of Secret-Shared Values. In the malicious setting, it is possible that dishonest parties send spurious shares during secret reconstruction phase. In eVSS [KZG10] (used in the cryptographic version of our protocol), this is solved by asking all parties to broadcast a proof (called *witness*) during reconstruction to verify broadcast shares. In our protocol, reconstruction is postponed to after circuit evaluation. Since the witnesses are generated in the sharing phase at the beginning of the computation, and the witnesses do not have necessary homomorphic properties, we cannot use them in our reconstruction phase. Instead, we correct corruptions using a BCH decoding algorithm (*e.g.*, the algorithm of Berlekamp and Welch [BW86]) as in normal secret reconstruction [Bea91]. This technique is also used in the VSS of Katz et al. [KKK08].

2.3.3 Security Proof

2.3.3.1 Proof of InitTriple

First, we prove u and v are uniform randoms, and they are common among all parties. By the correctness of the broadcast protocol, all honest parties receive the pairs $\{(C_{a_j}, C_{b_j})\}_{j=1}^{N}$. Let $u = \sum_{j=1}^{N} a_j$ and $v = \sum_{j=1}^{N} b_j$. Since C_{a_i} and C_{b_i} are homomorphic ciphertexts, $\sum_{j=1}^{N} C_{a_j} = C_{\sum_{j=1}^{N} a_j} = C_u$ and $\sum_{j=1}^{N} C_{b_j} = C_{\sum_{j=1}^{N} b_j} = C_v$. For all $i \in [N]$, each honest party P_i chooses a_i and b_i uniformly at random, so u and v are uniform randoms independent of the a_i's and b_i's sent by all parties. The correctness of the rest of the algorithm is based on the correctness of CipherShare and DecPrivate. Since DecPrivate is correct, the third step of the algorithm reveals u_i, v_i, and w_i only to P_i. The CipherShare algorithm requires each party to calculate a random polynomial on encrypted values. Lemma 4 shows how homomorphic properties of Enc can be applied to an eVSS sharing.

The first and second steps of the algorithm perform communications and computations over encrypted values only so, they are secure based on the security of the encryption scheme. The security of the third step follows by the security of DecPrivate. Finally, although the adversary has access to up to $T \leq N/3$ shares of each sharing $\langle u \rangle$, $\langle v \rangle$, and $\langle w \rangle$, based on the security of eVSS scheme, it does not have enough information to reconstruct u, v, and w, respectively. The communication cost of InitTriple can be computed based on the cost of CipherShare and DecPrivate. Thus, it is equal to $\mathsf{poly}(N)\mathsf{polylog}(N)$. □

Lemma 4. *If* $\langle w \rangle = (w_1, ..., w_N)$ *is a sharing of* $w \in \mathbb{Z}_p$*, then* $\langle \mathsf{Enc}(w) \rangle = (\mathsf{Enc}(\mathsf{w_1}), ..., \mathsf{Enc}(\mathsf{w_N}))$.

Proof. Let $\phi(x)$ be the polynomial corresponding to the sharing $\langle w \rangle = (w_1, ..., w_N)$. This means that $\phi(x)$ passes through the point $(0, w)$, and $\phi(i) = w_i$. We define a new polynomial

$\phi'(x) = \mathsf{Enc}(\phi(x))$. Using the homomorphic properties of Enc, we have $\phi'(i) = \mathsf{Enc}(w_i)$, thus $\phi'(0) = \mathsf{Enc}(w)$. □

2.3.3.2 Proof of CipherShare

We have already proved the correctness in Section 2.3.2.1. Based on the security of Gen-Rand, and the security of theorem 5, C_w and C_{u_i}, and computations on them are secure. Moreover, we need to prove that the adversary cannot decrypt the ciphertexts. Let $\phi_1(x) \in \mathbb{Z}_p[x]$ be a random polynomial of degree $\tau - 1$ such that $r_i = \phi_1(i)$. Using $\phi_1(x)$, we define a new polynomial $\phi_2(x) \in \mathbb{Z}_p[x]$ such that $\phi_2(x) = x.\phi_1(x)$. Clearly, $\phi_2(x)$ has degree τ, and passes through the origin. Finally, we use $\phi_2(x)$ to define a new polynomial $\phi_3(x) \in \mathbb{Z}_p[x]$ such that $\phi_3(x) = \gamma + \phi_2(x)$. It is clear that $\phi_3(x)$ passes through the point $(0, \gamma)$. Using the additive homomorphic property of eVSS secret sharing, $\phi_3(i) = \gamma_i + \phi_2(i)$. Thus, for all $i \in [N]$, $\gamma'_i = \phi_3(i)$ is a valid eVSS share of $\gamma' = \gamma$.

Based on the correctness and security of Gen-Rand, the value r and its shares are uniformly random and independent of any value in the protocol, and they generate a random polynomial $\phi_1(x)$ of degree $N/3 - 1$. $\phi_2(x) = x.\phi_1(x)$ is a new polynomial of degree $N/3$. In this case at most $N/3$ of the shares of this new polynomial is revealed to the adversary because dishonest parties can generate T shares in addition to the known fact that $\phi_2(x)$ passes through the origin. Thus, there is absolutely nothing the adversary can learn about $\phi_2(x)$. The same argument is valid for $\phi_3(x)$. The communication cost of CipherShare is equal to the communication cost of Gen-Rand plus N^2 extra messages sent in step 2 of the algorithm. The computation cost of CipherShare is equal to the computation cost of Gen-Rand plus $\mathsf{poly}(N)$ evaluations on encrypted data in step 3. Thus, the computation cost of CipherShare is equal to $\tilde{O}(\kappa + \log p)$. □

2.3.3.3 Proofs of DecPrivate and Reconst

Based on the result of [AJLA+12], DecPrivate is correct and secure, its communication cost is N, and its computation cost is $\tilde{O}(\kappa)$. The correctness and security of Reconst follows from the correctness of Lagrange interpolation and Welch-Berlekamp decoding algorithm [BW86, MS81]. It is easy to see that the communication cost of Reconst is $O(N^2)$, and the communication cost of Reconst is $O(N^4)$ based on Lemma 5.

Lemma 5. [Welch-Berlekamp] *Given a set of n points $S = \{(x_1, y_1) \mid x_i, y_i \in \mathbb{F}_p\}_{i=1}^{n}$ as input, the Welch-Berlekamp algorithm can be represented as an arithmetic circuit of multiplicative depth $poly(n)$ with computation cost of $O(n^3)$.*

2.3.3.4 Proof of Algorithm 2.13

In the following, we prove algorithm Algorithm 2.13 and our main theorem.

Setup. The correctness and security follows the proof of Theorems 4, 5, and 6, and the InitTriple algorithm.

Input Commitment. The correctness and security follows the proof of eVSS in Q_i. After this phase, each Q_i has a correct sharing of P_i's input. This is the base case for our proof of circuit evaluation phase**.**

Output Propagation. Once the computation phase of an output gate is finished, each party P_i of the quorum associated with the gate holds a share γ_i of the output value $\langle\gamma\rangle$, where $i \in [N]$. Since the number of honest parties in the quorum is $N - T > N - N/6 + 1 = 5N/6 + 1 > 2N/3$, honest parties have enough information to reconstruct the output value via $\mathsf{Reconst}(\gamma_i)$ and propagate it via $\mathsf{Output}(z)$. The correctness and security of output propagation follows from the proofs of $\mathsf{Reconst}(\gamma_i)$ and $\mathsf{Output}(z)$.

Costs. The communication and computation costs for the setup phase is equal to the cost of the quorum formation algorithm of Theorem 4 ($\tilde{O}(1)$ for each parties) plus the cost of TGen, VSetup, and InitTriple that are executed for each of the n quorums by their $N = O(\log n)$ parties. TGen communication cost is $O(N^2 polylog(N) D\kappa^2)$, and its computation cost is $O(N^3 D\kappa^2)$, where $D = \mathsf{poly}(N)$ is the multiplication depth of the circuit corresponding to the Welch-Berlekamp algorithm. Assuming the CRS model, algorithm VSetup has no communication cost, but the computation cost is $O(N)$. Based on the costs of InitTriple, the communication cost of the setup phase is $\tilde{O}(n\kappa^2)$, and its computation cost is $\tilde{O}(n\kappa^2)$, assuming $p = \mathsf{poly}(n)$.
The communication cost the Input Commitment phase is equal to the communication cost of running n different invocation of eVSS sharing. Therefore, the communication cost is equal to $O(n\log^2 n)$, and the communication cost is $O(n\log^2 n)$. The communication cost for the Circuit Computation phase is equal to the communication and computation cost of running m different instantiation of Multiply and Renew-Shares, assuming each gate has constant number of multiplication operations in its circuit. Hence, the communication complexity is $O(m\log^3 n)$, and the computation complexity is $O(m\log^4 n)$. The communication cost for the Output Propagation phase is equal to the communication and computation cost of running different instantiation of Reconst and Output for each output gate. Thus, its communication and computation costs are equal to $O(n\log^2 n)$ for one output gate. Assuming

one output gate and constant number of multiplication in each gate, the communication cost for the online phase is equal to$O(m \log^3 n)$, and the computation cost for online phase is $O(m \log^4(n))$. □

2.4 Conclusion

We described scalable protocols for solving the secure multi-party computation (MPC) problem among a large number of parties. We designed both a computationally unbounded and a computationally bounded adversary. In the former setting, our protocol is secure against a static malicious adversary corrupting less than a $1/3$ fraction of the parties. In the latter setting, we allow the adversary to corrupt less than a $1/6$ fraction of parties. For any deterministic function that can be computed by an arithmetic circuit with m gates, both of our protocols require each party to send a number of field elements and perform an amount of computation that is $\tilde{O}(m/n + \sqrt{n})$. We also show that our protocols provide perfect and universally-composable security.

Chapter 3

Anonymous Broadcast

Anonymous communication allows individuals to communicate with each other without fear of surveillance. An anonymity system attempts to conceal the relation between messages and their intended recipients, between messages and their actual senders, or both (*full anonymity*). Although the study of anonymous communication technology is often motivated by high-stakes use cases such as battlefield communication, espionage, or political protest against authoritarian regimes, anonymity actually plays many well-accepted roles in established democratic societies. For example, paying cash, voting, and opinion polling are everyday examples of anonymous activity.

In this chapter, we study the problem of secure anonymous broadcast, where a set of n parties want to anonymously send their messages to all parties. Anonymous broadcast is an important tool for achieving privacy in several distributed applications such as anonymous communication [Cha81], private information retrieval [CKGS98], secure auctions [FA00], and MPC. One challenging attack on anonymity systems is *traffic-analysis*, where a global adversary maps messages to their senders and recipients by monitoring the traffic exchanged between parties. Such a powerful adversary was assumed to be unrealistic in the past but it is believed to be realistic today especially if the service provider is controlled or compromised by a state-level surveillance authority [FF14]. Unfortunately, well-known anonymous services such as Crowds [RR98] and Tor [DMS04] are not secure against traffic analysis attacks. Moreover, most schemes that tolerate traffic analysis scale poorly with the network size, rendering them impractical for large networks.

We design a decentralized anonymous broadcast protocol that scales well with the number of parties and is robust against an active adversary. One motivating application for this protocol is a decentralized version of Twitter that enables provably-anonymous broadcast of messages.

Two widely-accepted architectures for providing general anonymity against an active adversary are *Mix Networks (Mix-Nets)* and *Dining Cryptographers Networks (DC-Nets)*, both of which were originally proposed by Chaum [Cha81, Cha88]. Mix-Nets require semi-trusted infrastructure nodes and are known to be vulnerable to traffic analysis and active attacks [PW86]. DC-Nets [Cha88, GJ04, vABH03, WP90], on the other hand, provide ABC protocols among a group of parties without requiring trusted parties. The core idea of DC-Nets is that a protocol for multi-party computation can be used to perform sender and receiver anonymous broadcast. For example, if party p_i wants to broadcast a message m_i anonymously, then all other parties participate in a multi-party sum with input zero, while party p_i participates with input m_i. All parties learn the sum, which is m_i while all inputs remain private. This ensures that no party can trace the output message m_i to its input, keeping p_i anonymous.

Although DC-Nets are provably-secure against traffic analysis, they face several challenges. First, a reservation mechanism is required to schedule which party is broadcasting without compromising the anonymity of the sender. Second, DC-Nets are susceptible to collisions, which degrade throughput. A jamming adversary may even use collisions to render the channel useless by continuously transmitting in every round. Third, typical DC-Nets are not scalable given that the bit complexity required to anonymously broadcast a single bit among n parties is $\Omega(n^2)$.

State-of-the-art approaches that address some of these challenges include [CGWF13, GJ04, vABH03]. The majority of these methods scale poorly with network size, rendering them impractical for large networks. Recently, Zamani et al. [ZSMK13, KKSZ13] proposed the first ABC protocol, where each party sends $o(n)$ bits to broadcast a bit among n parties. Their protocol uses multi-party computation to achieve full anonymity and logarithmic-size groups of parties to achieve $\tilde{O}(1)$ communication and computation costs. Unfortunately, their protocol has polylogarithmic rounds of communication and is not practical due to large logarithmic factors hidden in the complexity notation.

To the best of our knowledge, every sender and receiver anonymous broadcast protocol that does not rely on a trusted party consists of at least three steps.

1. **Input.** Initially, each party holds a message. The party distributes its message or a representation of it among all or a subset of parties. This step requires sending $\Omega(n)$ messages.

2. **Multi-Party Shuffling.** All or a subset of parties participate in a multi-party protocol

to obliviously generate a random permutation of the sequence of message they hold.

3. **Output.** All or a subset of parties holding a sequence of messages broadcast them to all parties. This step requires sending $\Omega(n^2)$ bits for delivering n shuffled bits.

Since much cannot be done to improve the cost of the output phase, we will mainly focus on the multi-party shuffling step in this chapter. Multi-party shuffling can be used as a black-box in multi-party computation problems. Boyle et al. [BGT13] use oblivious shuffling to randomly choose inputs for a sublinear function evaluation, where a function is evaluated over $o(n)$ inputs chosen uniformly at random in the presence of an active adversary. Laur et al. [LWZ11] and Goodrich et al. [GMOT12] describe how multi-party shuffling can be used for implementing oblivious database operations and oblivious storage.

3.1 Related Work

Some protocols are built upon a relaxed notion of anonymity called *k-anonymity* [vABH03, YF05, LBCZ+13], where the adversary is assumed to be unable to identify the actual sender/receiver of a message from a set of k parties (called *anonymity set*). Even though *k-anonymity* often increases efficiency significantly, choosing small k's can result in severe privacy problems. For example, attackers often have background knowledge and it is shown that small anonymity sets are likely to leak privacy when attackers have such knowledge [MKGV07]. For example, a person located in New Mexico is more likely to search for a restaurant serving chili stew than a person in Vermont.

Von Ahn *et al.* [vABH03] develop a cryptographic broadcast protocol based on DC-Nets that is resistant to a static active adversary. A set of n parties with private inputs compute and share the sum of their inputs without revealing any parties' input. The authors introduce k-anonymity, which means no polynomial-time adversary may distinguish the sender/receiver of a message from among k honest senders/receivers. To achieve k-anonymity, they partition the set of parties into groups of size $M = O(k)$ and execute a multi-party sum protocol inside each group. The jamming detection mechanism is weak against an adversary who may waste valuable resources by adaptively filling up to M channels. In the case where n-anonymity is desired, the protocol requires $O(n^3)$ messages to be sent per anonymous message and the total bit complexity is $O(n^4)$. The protocol has latency that is $O(1)$ on average when the number of broadcasts is large, but which can be $O(n)$ in worst case for a single broadcast.

Golle and Juels [GJ04] employ cryptographic proofs of correctness to solve the jamming problem in DC-Nets assuming a static Byzantine adversary. The protocol detects jamming with high probability in $O(1)$ rounds, requiring a total communication and computation complexity of $O(n^2)$ bits. Their protocol assumes the existence of a reliable broadcast and a centralized trusted authority for key management distribution.

The Xor-trees approach of [DO00] extends DC-Nets to achieve $O(n)$ amortized bit complexity, which is optimal. In this protocol, only a single user is allowed to send at any one time in a Xor-tree. Hence, the protocol is subject to performance degradation due to collisions as the number of users increases. The protocol assumes the existence of a public-key infrastructure and a non-Byzantine polynomial-time adversary. The total bit complexity of the protocol is $O(n^2t^2)$ bits in worst case, where t is the number of dishonest parties. The latency of the protocol is $O(n)$ in worst case. However, a sender may broadcast large payloads to amortize the costs. The amortized latency of the protocol is $O(1)$.

The Verdict protocol of [CGWF13] (which is based on Dissent [CGF10]) has a client-server architecture and uses verifiable DC-Nets, where participants use public-key cryptography to construct ciphertext, and knowledge proofs to detect and exclude jamming parties before disruption. The protocol assumes the existence of a few highly-available servers, where at least one server is honest. All servers must be alive, however, for the protocol to work. An interesting aspect of Verdict is that it is robust to a large fraction of Byzantine parties (up to $n-2$). The paper demonstrates empirically that the system scales well with the number of parties, when the number of servers is fixed. The Tarzan protocol of Freedman and Morris [FM02] provides resistance against traffic analysis, but only against a passive adversary.

The Aqua protocol of Le Blond *et al.* [LBCZ+13] provides k-anonymity with traffic-analysis resistance against passive global attacks and active local attacks. The protocol achieves anonymity in a way similar to Tor (onion routing) and achieves unobservability through traffic obfuscating, which is to add artificial delay or artificial traffic (called chaff) to the connection. Laur et al. [LWZ11] describe a multi-party shuffling protocol that can be used for anonymizing a set of inputs. Although the communication and round complexity of their protocol scales well with the number of inputs, they scale exponentially with the number of parties and hence, the method cannot be used in our model, where n is relatively large.

Goodrich [Goo11] proposes an efficient data-oblivious randomized shellsort algorithm. Unfortunately, when implemented in a multi-party setting, this protocol requires $O(m)$ rounds of communication to sort m values and has communication complexity

$O(\ell^2 n^2 m \log m)$, where ℓ is the message size. Zhang [Zha11] and Hamada *et al.* [HKI+12] develop constant-round MPC sorting protocols that scale well with the number of inputs but scale poorly with the number of parties.

State-of-the-art approaches that address some of these challenges include [CWF12, GJ04, vABH03]. The majority of these methods are cryptographic in nature, and scale poorly with network size, rendering them impractical for large networks. We are not aware of any unconditionally-secure anonymous protocol that scales better than $O(n^2)$ bits per anonymous bit delivered.

In this dissertation, we first address the scalability and jamming limitations of DC-Nets. Our jamming-resistant protocol provides anonymity at a total bit complexity of $\tilde{O}(n)$ per anonymous message and a total latency of $\mathsf{polylog}(n)$. Our result is motivated by a vision of creating peer-to-peer versions of microblogging services with large number of users such as Twitter, but with provable anonymity guarantees. In Twitter, users can tolerate a higher messaging latency when compared to interactive web browsing applications. Therefore, trading-off latency for bandwidth cost and load-balancing is a promising goal for such applications.

3.2 Preliminaries

In this section, we describe standard tools used throughout this section.

Sorting Networks. A *sorting network* is a network of *comparators*. Each comparator is a gate with two input wires and two output wires. When two values enter a comparator, it outputs the lower value on the top output wire, and the higher value on the bottom output wire. Ajtai et al. [AKS83b] describe an asymptotically-optimal $O(\log n)$ depth sorting network called *AKS*. Unfortunately, the AKS network is not practical due to large constants hidden in the depth complexity. Leighton and Plaxton [LP90] propose a practical $O(\log n)$-depth *probabilistic sorting network* that sorts with very high probability meaning that it sorts all but $\epsilon n!$ of the $n!$ possible input permutations, where $\epsilon = 1/2^{2^{c\sqrt{\log n}}}$, for any constant $c > 0$. While this circuit is sufficient for us to prove our main results, one can instead use the $O(\log^2 n)$-depth sorting network of Batcher [Bat68] for sorting all $n!$ permutations at the expense of $O(\log^2 n)$ protocol latency.

Secure Comparison. Given two linearly secret-shared values $a, b \in \mathbb{Z}_p$, Nishide and Ohta [NO07] propose an efficient protocol for computing a sharing of $\rho \in \{0, 1\}$ such

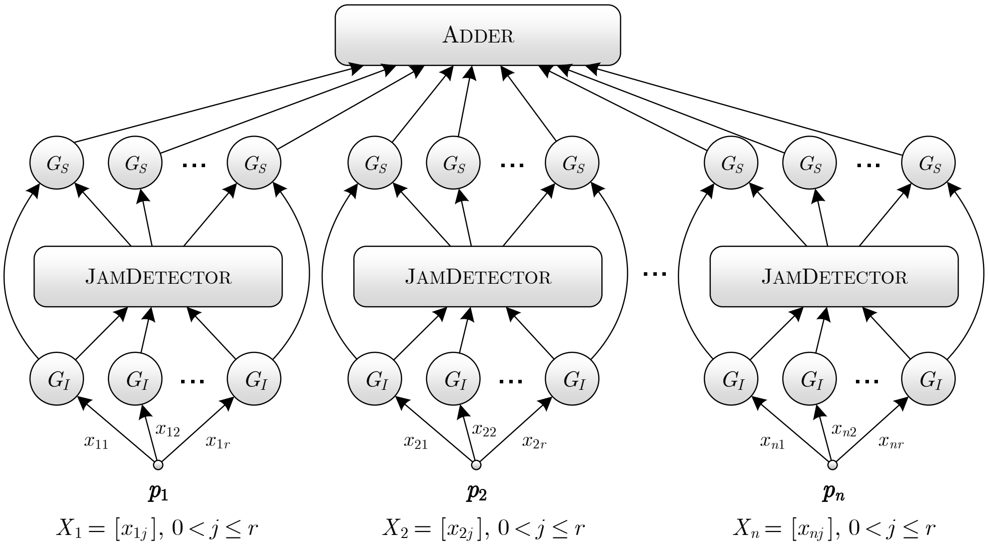


Figure 3.1: Jamming-resistant circuit for n parties

that $\rho = (a \leq b)$. Their protocol has $O(1)$ rounds and requires $O(\ell)$ invocations of a secure multiplication protocol, where ℓ is the bit-length of elements to be compared. We refer to this protocol by Compare. We also describe a fast multiplication protocol to be used along with the comparison protocol of [NO07] for implementing fast comparator gates.

3.3 Anonymous Broadcast via DC-Nets

In this section, we design a fully-anonymous and scalable anonymity system based on DC-Nets. We prove the following theorem in this section.

Theorem 6. *Assume there are n parties in a fully-connected network with private channels, up to $t < n/3$ of which are controlled by an adversary, and each party has a constant-size message to broadcast. If all honest parties follow the protocol of this section, then with high probability,*

1. *Each honest party broadcasts its message to all other honest parties with probability $1/k$, where $k > 1$ is a constant,*

2. *The communication is fully-anonymous,*

3. *Each party sends $\tilde{O}(n)$ bits and performs $\tilde{O}(n)$ computations,*

4. *The latency of the protocol is* $\mathsf{polylog}(n)$.

Assume n parties $P_1, ..., P_n$, where for $1 \leq i \leq n$, party P_i has a message m_i to broadcast to the network anonymously. Party P_i chooses a number $l \in [1, r]$ uniformly at random and forms a vector $X_i = [x_{ij}]$, where $x_{il} = m_i$ and $x_{ij} = 0$ (for all $1 \leq j \leq r$ and $j \neq l$). For some constant $k > 1$, each of the $r = kn$ positions in X_i is referred to as a *slot*. For simplicity, we assume r is an integer power of two. The parties then run an MPC algorithm to compute a function $f(X_1, X_2, ..., X_n)$ such that every party learns the vector addition $\sum_{i=1}^{n} X_i$ and none of the parties can send more than one non-zero input. In the following, we explain the circuit that computes function f.

Our Circuit Figure 3.3 shows the circuit, which consists of two major sub-circuits: JamDetector that detects jamming inputs and Adder that computes the component-wise addition. We now describe each part of the circuit in detail.

- **Input gates.** In Figure 3.3, gates labeled G_I are called *input gates* and compute the identity function $G_I(x_{ij}) = x_{ij}$. Input gates are necessary for ensuring consistency among all inputs a party sends during the protocol execution.
- **Jam detector.** Each party is associated with exactly one JamDetector sub-circuit. The sub-circuit has r inputs and r outputs: all outputs are set to zero if no jamming is detected for the corresponding party otherwise all outputs are set to a non-zero value. Figure 3.3 depicts a circuit for $n = 2$ and $r = 4$ showing the sub-circuit in detail. Each JamDetector consists of three types of gates, which are defined in the following on the field $\mathbb{F}$:

$$G_1(y) = \begin{cases} \mathbb{0}, & \text{if } y = \mathbb{0} \\ 1 & \text{otherwise} \end{cases}$$

$$G_2(y_1, y_2) = \begin{cases} \mathbb{0}, & \text{if } y_1 + y_2 = \mathbb{0} \\ 1, & \text{if } y_1 + y_2 = 1 \\ 2 & \text{otherwise} \end{cases}$$

$$G_3(y) = \begin{cases} \mathbb{0}, & \text{if } y = \mathbb{0}, 1 \\ -1 & \text{otherwise} \end{cases}$$

Each JamDetector contains a perfect binary tree consisting of only G_2 gates over r leaf nodes, consisting of only G_2 gates. This tree is connected from its root gate to an inverted perfect binary tree of only G_3 gates over $r/2$ leaf nodes, consisting of only G_3 gates.

- **Selector gates.** Gates labeled G_S in Figure 3.3 and Figure 3.3 are called *selector gates*. Each of these gates acts like a selector function: if the first input (which is an output of a JAMDETECTOR sub-circuit) is zero, it simply outputs the second input otherwise it outputs zero. The selector gate is defined as follows:

  $$G_S(y_1, y_2) = \begin{cases} y_2, & \text{if } y_1 = \mathbb{0} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

  where y_1 is the output of corresponding G_3 and y_2 is the output of corresponding input gate.

- **Adder.** There is one ADDER sub-circuit, which is a simple sum circuit and consists of r perfect binary trees each of which has n leaf nodes. The j-th binary tree sums up the outputs of all the j-th selector gates from all parties, which correspond to all the j-th slots. In order to avoid sending incorrect data down the tree when a collision happens, each sum gate in ADDER simply outputs zero if both of its inputs are non-zero.

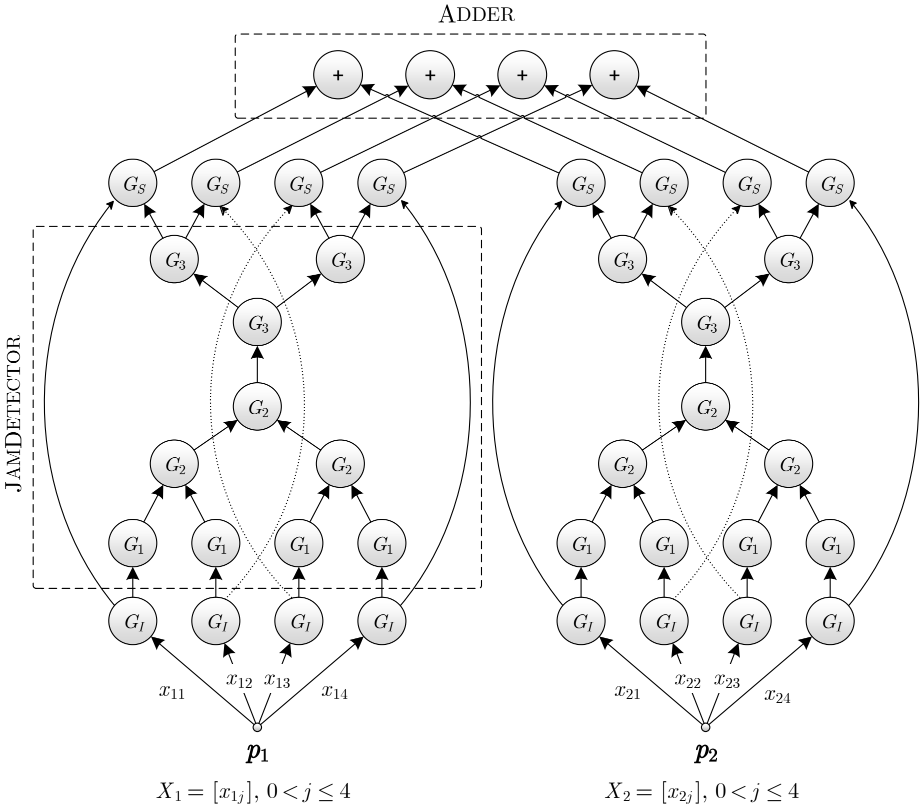


Figure 3.2: Jamming-resistant circuit for $n = 2$ and $r = 4$

Circuit Computation Using our MPC algorithm described in Chapter 2, the inputs of all parties are sent up the circuit simultaneously. The JamDetector sub-circuit filters out any jamming inputs and sends the rest of them up to Adder . The Adder sub-circuit computes the sum of all non-jamming inputs and finally, the result is sent down to every party via the output propagation algorithm described in Section 2. The MPC algorithm ensures that (1) the output of the circuit is computed correctly and is reliably sent to all parties; and (2) no party learns any information about the inputs or outputs of intermediate gates, except what can be learned from their own input and the final output of the circuit.

3.3.1 Security Proof

In this section, we prove Theorem 6.

Lemma 6. *Consider a JamDetector sub-circuit, r input gates (G_I), and r selector gates (G_S) associated with party P_i (see Figure 3.3). During the computation of the circuit, at most one of the selector gates, say G_S^*, outputs a non-zero value. If P_i is honest, then G_S^* is chosen uniformly at random from the r selector gates.*

Proof. Assume P_i has a message m_i to broadcast. First, consider the case where P_i is honest. P_i chooses an input gate uniformly at random and sends m_i to that gate and sends $\mathtt{0}$ to all other input gates. Hence, at most one G_1 gate outputs $\mathtt{1}$ and the rest output $\mathtt{0}$. Thus, all G_2 gates output $\mathtt{0}$ or $\mathtt{1}$, and consequently all G_3 gates output $\mathtt{0}$. Finally, the output of all G_s gates is the same as the output of the corresponding input gate. Hence, at most one G_s gate, say G_S^*, outputs a non-zero value. Obviously, G_S^* is chosen uniformly at random because the corresponding input gate is chosen uniformly at random. Now, consider the case where P_i is dishonest and sends non-zero values to more than one of his input gates (this corresponds to a jamming attack). In this case, more than one G_1 outputs $\mathtt{1}$ and since the sum is greater than one, all G_3 gates output $-\mathtt{1}$ and all G_s gates output $\mathtt{0}$. □

In our protocol, we explained that each party selects one of the r slots uniformly at random. Even if all parties are honest (i.e., no jamming is occurring), *collisions* are always possible meaning that one or more parties may choose the same slot for their non-zero message. Unfortunately, there is no efficient non-interactive method that guarantees all parties select distinct slots. Hence, for simplicity, we ensure that the number of slots is large enough so that the probability of collision remains small. The following lemma gives an upper-bound on the probability of collisions.

Lemma 7. *If $r = kn$, where r is the number of slots and $k > 1$ is a constant, then the probability of collision for one party is less than $1/k$.*

Proof. By Lemma 6, we can ensure that each party sends his input to at most one slot, i.e., jamming has already been prevented. Let P be an arbitrary honest party. The probability that another party chooses exactly the same slot that P chooses is $1/r$. So, the probability of collision for P is at most $(n-1)/r$. Since $r = kn$, the probability of collision is at most $(n-1)/kn < 1/k$. As in [vABH03], in most cases we can set $r = 2n$, which makes the probability of collision for a party less than $1/2$. □

Lemma 8. *Our protocol is fully-anonymous and it ensures that for each honest party P_i, who sends message m_i, all honest parties learn m_i with probability $1 - 1/k$.*

Proof. The MPC protocol guarantees that the adversary only learns the set of outputs and the corresponding slots besides his own input. The input slots were chosen independently and uniformly at randomly by each party. As proved in Lemma 6, every m_j sent by an honest party P_j $(1 \le j \le n)$ appears in at most one of the corresponding selector gates uniformly at random and all other selector gates output $\mathbb{0}$. Since the Adder sub-circuit keeps the ordering of slots, the output slots corresponding to honest parties are also chosen uniformly at random and thus, give the adversary no extra information. Therefore, conditioned on all information that the adversary can learn during the protocol, the probability that m_i is sent by any honest party P_j $(1 \le j \le n)$ is $1/(n-t)$. This means that the communication is sender-anonymous.

Our MPC protocol guarantees that all honest parties learn every output of the circuit with high probability, one of which is m_i. This shows that the communication is receiver-anonymous and thus, the protocol is fully-anonymous. Moreover, based on Lemma 7, the probability that it fails to deliver m_i is less than $1/k$. □

Lemma 9. *The circuit has depth $O(\log n)$ and $O(nr)$ gates, where r is the number of slots.*

Proof. In the circuit of Figure 3.3, for each party a subtree of depth $2\log r + 3$ consisting of $5r - 2$ gates is required ignoring the Adder circuit, which consists of r binary trees each with n leaves. Each tree has depth $\log n + 1$ and consists of $2n - 1$ gates so the Adder sub-circuit has $(2n-1)r$ gates. Therefore, the total number of gates in a circuit for n parties is $7nr - 2n - r$ and the circuit has depth $2\log r + \log n + 4$. □

Lemma 10. *If all honest parties follow our protocol, then with high probability, the protocol sends $\tilde{O}(n)$ bits and performs $\tilde{O}(n)$ computations for sending one anonymous bit. The latency of the protocol is* $\mathsf{polylog}(n)$.

Proof. Our MPC protocol requires each party to send $\tilde{O}(\frac{m}{n})$ bits and perform $\tilde{O}(\frac{m}{n})$ computations to compute any arbitrary function f, where m is the number of gates in the circuit for computing f. However, unlike our MPC protocol that propagates only one message (the result of computation) to every party at the output propagation phase, our protocol transmits n messages at this phase. Therefore, it requires each party to send $\tilde{O}(\frac{n^2+m}{n})$ bits. From Lemma 9, we have $m = O(n^2)$ gates so each party sends $\tilde{O}(n) = \tilde{O}(n)$ bits for sending n anonymous bits. Alternatively, the protocol sends $\tilde{O}(n)$ bits for sending one anonymous bit. With a similar argument, the protocol requires each party to perform $\tilde{O}(n)$ computations. Our MPC algorithm takes $O(d)$ rounds to compute any arbitrary circuit with depth d, where n is the number of parties. Lemma 9 shows that the circuit for n parties has depth $O(\log n)$, since $r = O(n)$. Therefore, the protocol takes $\mathsf{polylog}(n)$ rounds to send an anonymous bit. □

3.4 Anonymous Broadcast via Multi-Party Shuffling

3.4.1 Introduction

Shuffling a sequence of values is a fundamental tool for randomized algorithms; applications include fault-tolerant algorithms, cryptography, and coding theory. In *secure multi-party shuffling (MPS)* problem, a group of parties each holding an input value want to randomly permute their inputs while ensuring no party can map any of the outputs to any of the input holders better than can be done with a uniform random guess. An MPS protocol is a useful primitive for achieving privacy and robustness in many applications such as anonymous communication [Cha81], location-based services [GG03], electronic voting [Nef01], secure auctions [FA00], and general multi-party computation [BGT13].

Despite many applications of MPS, we are not aware of any technique that can be used to achieve a scalable and secure MPS protocol. We believe this is of increasing importance with the growth of modern networks. Moreover, most protocols lack load-balancing – a crucial requirement for protocols running in large networks. With the rise of sophisticated cyber-attacks, it is now essential to provide provable guarantees against strong adversaries. Also, relying on trusted parties has become a major security issue in today’s world.

In this section, we address these concerns by proposing a scalable and load-balanced protocol for MPS that is unconditionally-secure against malicious attacks and does not rely on trusted parties.

Our Contribution. We first propose a formal definition of security for MPS. Our definition is different from the standard definition of security for *multi-party computation (MPC)* [BGW88], where a group of parties each holding a *private input* want to compute a known function over their inputs, without revealing any more information about their inputs than what is revealed by the output of the function. Instead of focusing on inputs privacy, we base our definition on the secrecy of the output permutation.

Next, we propose an unconditionally-secure MPS protocol that scales polylogarithmically with the number of parties, tolerates malicious faults, and is load-balanced. Simulations of our protocol suggest that it compares favorably with the current state of the art in terms of communication cost, computation cost, and the number of communication rounds.

In our protocol, we achieve sublinear per-party communication complexity by requiring each party to only communicate with polylogarithmic-size groups of parties rather than with *all* parties. This approach, however, introduces important technical challenges to our model; the most important one is to guarantee the adversary cannot break the security of our protocol via coalitions of corrupted parties in more than one group, when we share the same secret information with the parties in these groups. Some prior work solve this by relaxing the load-balancing requirement [BGT13], the resiliency bound [ZMS14], or practical efficiency [DKMS12]. We propose a novel technique called *share renewal* without relaxing any of these requirements.

When a protocol is concurrently executed alongside other protocols, one requires to ensure this composition preserves the security of the protocol.[1] Since our goal is to design *modular* MPS protocols that can be flexibly used with other protocols, we show security of our protocol under the universal composability framework as described by Canetti [Can01].

Our Model. Consider n parties $P_1, ..., P_n$ in a fully-connected synchronous network with private and authenticated channels. We assume the parties have no access to any trusted party and/or to any reliable broadcast channel. We consider a *malicious adversary* who corrupts at most $t < n$ of the parties and can see (and analyze) the entire traffic in the network, but cannot see the content of messages transmitted between uncorrupted parties

[1] An adversary attacking several protocols that run concurrently can cause more harm than by attacking *stand-alone* executions of these protocols [Can01].

since we assume private links. The corrupted parties not only can gossip their 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 finally assume that the adversary is *static* meaning that it has to select the set of corrupted parties at the start of the protocol.

Problem Statement. Let $\mathbb{F}$ be a finite field, and $\pi : \{1, ..., n\} \rightarrow \{1, ..., n\}$ denote a *permutation*; a one-to-one and onto function that maps a sequence of n elements $(x_1, ..., x_n) \in \mathbb{F}^n$ to another sequence $(x_{\pi(1)}, ..., x_{\pi(n)}) \in \mathbb{F}^n$. For $i \in \{1, ..., n\}$, every party P_i holds an input $x_i \in \mathbb{F}$. A *multi-party shuffling (MPS)* protocol allows these parties to agree on a permutation π of the sequence $(x_1, ..., x_n)$. We consider two variants of this problem. In the first variant, which we call *single-output MPS*, each party P_i is required to receive only one of the shuffled inputs $x_{\pi(i)}$. In the second variant, which we call *all-output MPS*, each party receives the entire output sequence $(x_{\pi(1)}, ..., x_{\pi(n)})$. We now define our notion of security.

Definition 1. *An MPS protocol is said to be t-*secure *if and only if, in the presence of a malicious adversary corrupting up to $t < n$ of the parties, the protocol ensures*

- Unlinkability: *the adversary can guess π correctly with probability at most $\frac{1}{(n-t)!}$. We refer to the set of possible permutations from which the adversary tries to guess the secret permutation π as the* unlinkability set.
- Correctness: *each party is guaranteed that the output it receives is one of the inputs (for single-output MPS) or contains all (and only all) the inputs (for all-output shuffle).*
- Output delivery: *corrupted parties cannot prevent honest parties from receiving their output.*

In this section, we consider a relaxed version of Definition 1. This allows us to achieve the highest level of efficiency in our protocol in exchange of a very small increase in the success probability of the adversary.

Definition 2. *We say an MPS protocol is* almost t-secure *if and only if in the presence of a malicious adversary corrupting up to $t < n$ of the parties, the protocol guarantees correctness and output delivery, and that the adversary can guess π correctly with probability at most $\frac{1}{(n-t)!}(1 + \delta)$, where $\delta = o(1)$ is called the* deviation factor.

3.4.1.1 Our Results

We prove the following main theorem in Section 3.4.4.

Theorem 2. *There exists a universally-composable MPS protocol such that with probability* $1 - O(n^{-3})$*, it guarantees the following properties:*

- *The protocol is almost* t*-secure against a computationally-unbounded malicious adversary with static corruptions, where* $t < (1/3 - \epsilon)n$*, for some positive constant* ϵ*.*
- *The deviation factor is* $O(2^{-2^{k\sqrt{\log n}}})$*, for some constant* $k > 1$.
- *Each party sends* $\tilde{O}(1)$ *bits and computes* $\tilde{O}(1)$ *operations.*[2]
- *The protocol terminates after* $O(\log n)$ *rounds of communication.*

In Section 3.4.3.4, we also construct a computationally-secure variant of Theorem 10 to observe (via simulations) how much cryptographic techniques can influence practical efficiency of our protocol. This protocol provides the same guarantees as Theorem 10 except for a polynomially time-bounded adversary. We provide our simulation results in Section 3.4.5.

3.4.1.2 Related Work

Shuffling in the multi-party setting has already been studied, primarily in the context of *mix-nets*. As first defined by Chaum [Cha81], a mix-net consists of a chain of servers (called *mixes*) that randomly reorder a sequence of messages in a way that the correlation with the corresponding input messages remains hidden. To ensure honest behavior in the malicious setting, a *verifiable shuffling* [Nef01, AW07] technique is often used, where each mix is asked to prove correctness of its shuffles without leaking how the shuffle was performed.

Unfortunately, Mix-nets and verifiable shuffling techniques rely on cryptographic assumptions. Moreover, mix-nets require semi-trusted servers and are known to be vulnerable to *traffic-analysis* attacks [PW86]. In traffic analysis, a global adversary maps messages to their senders and recipients by monitoring the traffic exchanged between parties. Protocols such as [RS93, BFT04] attempt to solve this with provable guarantees. However, they are either complicated and scale linearly with the number of parties [RS93], or are not secure against malicious attacks and an adversary monitoring all communication channels [BFT04].

[2] The symbol $\tilde{O}$ is used as a variant of the big-O notation that ignores the logarithmic factors. Thus, $f(n) = \tilde{O}\left(g(n)\right)$ means $f(n) = O\left(g(n) \log^k g(n)\right)$, for some k.

Chaum [Cha88] uses MPC to introduce the *dining cryptographers network (DC-net)* for achieving unlinkability [3] between inputs and outputs; a crucial requirement for both anonymous communication and MPS. He uses a simple MPC technique to design an unconditionally-secure anonymous broadcast protocol called . When a party P wants to broadcast a message M anonymously, all parties participate in a multi-party sum with input zero except P who participates with its input M. As a result of MPC, all parties learn the sum of the inputs (*i.e.*, M) without any party being able to trace the output to P. The DC-net eliminates the two limitations of Mix-nets: cryptographic assumptions and traffic-analysis vulnerability.

Although the original DC-net allows only one participant to broadcast at a time, there are variants such as [vABH03] that implement all-to-all anonymous broadcast and thereby enable multi-party shuffling of the inputs. Unfortunately, DC-nets suffer from collision and jamming attacks. Although several work address these issues [vABH03, GJ04, CGWF13], they either do not scale well with the number of parties [vABH03, GJ04] or require a few highly-available servers [CGWF13].

MPS is closely related to *data-oblivious* protocols [GO96]. A protocol is data-oblivious if its control flow is independent of input data. Such a protocol can be used to anonymize access patterns or prevent an adversary from taking over a certain fraction of protocol inputs. Customized shuffling techniques are designed in the context of oblivious RAMs [GO96], oblivious database manipulation [LWZ11], oblivious sorting [Zha11, Goo11, HKI$^+$12], and evaluation of sublinear functions [BGT13].A multi-party sorting protocol such as that of [Zha11, HKI$^+$12] can be used to perform MPS. Although these protocols scale well with n, they scale poorly with the number of parties.

Rackoff and Simon [RS93] show that if all parties send at each time step, then the traffic-analysis problem can be solved using MPC. This means that a general MPC scheme such as [BGW88] that can securely compute any functionality (including shuffling), can be used to design an MPS protocol with traffic-analysis resistance. Although much theoretical progress has been made in the MPC literature to achieve polylogarithmic overhead [BGT13, DKMS12], there is a lack of practical solutions, especially for large number of parties. Moreover, most of these techniques cannot be easily implemented due to a lack of detailed protocol specifications.

In Table 3.1, we compare our main protocol with several other ones that can be used to

[3] Pfitzmann and Hansen [PK01] define "Unlinkability of two or more items means that within this system, these items are no more and no less related than they are related concerning the *a priori* knowledge." This means the probability of those items being related stays the same before and after the run within the system.

Table 3.1: Comparison of MPS techniques

Protocol	Adversary's Computational Power	Malicious Adversary?	Fraction of Parties Controlled	Fraction of Links Monitored	MPS Security	Latency	Bandwidth
[Cha81]	Bounded	No	$O(1)^\dagger$	See note‡	See note§	polylog(n)	polylog(n)
[RS93]	Bounded	No	$O(1)^\dagger$	All	Statistical	polylog(n)	$\tilde{O}(n)$
[BFT04]	Bounded	No	$O(1)^\dagger$	$O(1)$	Statistical	polylog(n)	polylog(n)
[BGT13]	Bounded	Yes	$1/3 - \epsilon$	All	Almost	polylog(n)	$\tilde{O}(n)$
[DKMS12]	Unbounded	Yes	$1/3 - \epsilon$	All	Almost	$O(\log n)$	$\tilde{O}(\sqrt{n})$
Our result	Unbounded	Yes	$1/3 - \epsilon$	All	Almost	$O(\log n)$	$\tilde{O}(1)$

†This protocol assumes the rest of parties are trusted.
‡[PW86] shows traffic-analysis attacks on this protocol if all links are monitored by the adversary.
§Originally supposed to generate perfect shuffles but known attacks reduce shuffle security.
Measures the statistical distance between the distribution generated by the system and the uniform distribution [RS93].

solve the MPS problem. To make a fair comparison with the MPC protocols of [BGT13, DKMS12], we use their techniques to compute our own shuffling functionality described in Section 3.4.3. In this table, by bandwidth we mean the communication complexity per shuffled message delivered.

3.4.2 Preliminaries

We now define our notation and describe the tools used throughout this section.

Notation. For prime p, let $\mathbb{F}_p$ denote a finite field with p elements. We say an event occurs *with high probability*, if it occurs with probability $1 - 1/n^c$, for some positive constant c.

Verifiable Secret Sharing. A *secret sharing* protocol allows a party (called *the dealer*) to share a secret among n parties such that any set of t or less parties cannot gain any information about the secret, but any set of at least $t + 1$ parties can reconstruct it. A *verifiable secret sharing (VSS)* protocol is a secret sharing protocol with the additional property that after the sharing phase, a corrupted dealer is either disqualified or the honest parties can reconstruct the secret, even if shares sent by corrupted parties are spurious. In our protocol, we use the VSS scheme of Ben-Or *et al.* [BGW88]. We refer to the sharing protocol of this scheme as VSS-Share and to its reconstruction protocol as VSS-Reconst.

The VSS scheme of [BGW88] is based on Shamir's secret sharing [Sha79]. In this scheme, the dealer shares a secret s among n parties by choosing a random polynomial $f(x)$ of degree t such that $f(0) = s$. For all $i \in [n]$, the dealer sends $f(i)$ to the i-th party. Since at least $t + 1$ points are required to reconstruct $f(x)$, no coalition of t or less parties can reconstruct s. A secret sharing scheme is *linear* if given two shares a_i and b_i of secrets a and b, $c_i = a_i + b_i$ is a valid share of $c = a + b$.

Theorem 3 ([BGW88])**.** *There exists a synchronous linear VSS scheme for $t < n/3$ that is unconditionally-secure against a static malicious adversary.*

Quorum Building. King *et al.* [KLST11] give a protocol that can be used to bring all parties to agreement on a collection of n quorums. A *quorum* is a set of $N = O(\log n)$ parties, where it is guaranteed that at most a fixed fraction of the parties in the set are corrupted. In general, one can use any BA algorithm (such as [BGH13]) to build a set of quorums in the way described in [KLST11].

Theorem 4 ([KLST11, BGH13])**.** *There exists an unconditionally-secure protocol that brings all honest parties to agreement on n quorums with probability $1 - O(n^{-3})$. The protocol has $\tilde{O}(n)$ amortized communication and computation complexity over the number of parties, and it can tolerate up to $(1/3 - \epsilon)n$ corrupted parties, for any positive ϵ. Each quorum is guaranteed to have $T < N/3$ corrupted parties.*

We refer to this protocol as Build-Quorums. Several recent MPC schemes [BGT13, ZMS14] make use of quorums to achieve scalability. We are particularly inspired by Dani *et al.* [DKMS12].

Sorting Networks. A *sorting network* is a network of *comparators*. Each comparator is a gate with two input wires and two output wires. When two values enter a comparator, it outputs the lower value on the top output wire, and the higher value on the bottom output wire. Ajtai *et al.* [AKS83b] describe an asymptotically-optimal $O(\log n)$ depth sorting network. However, this network is not practical due to large constants hidden in the depth complexity. Leighton and Plaxton [LP90] propose a *probabilistic sorting circuit* with depth $7.44 \log n$ that sorts a randomly chosen input permutation with very high probability meaning that it sorts all but $\sigma \cdot n!$ of the $n!$ possible input permutations, where $\sigma = 1/2^{2^{\kappa\sqrt{\log n}}}$, for some constant $\kappa > 0$.[4]

Secure Comparison. Given two linearly secret-shared values a, b, Damgård *et al.* [DFK+06] propose an efficient protocol for computing a new secret-shared value $\rho = (a \leq b)$ meaning

[4] This gives a Monte Carlo guarantee: for $(1 - \sigma)n!$ of input permutations, the circuit sorts correctly, but for the rest $\sigma n!$ permutations, it simply fails and gives no guarantees.

that ρ is 1 if $a \leq b$ and 0 otherwise. Their protocol is unconditionally secure, has $O(1)$ rounds, and requires $O(\ell)$ invocations of a secure multiplication protocol, where ℓ is the bit-length of elements to be compared. We denote this protocol by Compare. For multiplication of secret-shared values, we use the protocol of Ben-Or *et al.* [BGW88] with the simplifications of Gennaro *et al.* [GRR98]. By plugging the VSS of Theorem 3 into the protocol of [GRR98], we achieve an unconditionally-secure multiplication protocol, which we denote by Multiply.

Secure Broadcast In the malicious setting, when parties have only access to secure pairwise channels, a protocol is required to ensure secure (reliable) broadcast. Such a broadcast protocol guarantees all parties receive the same message even if the broadcaster (dealer) is corrupted and sends different messages to different parties. It is known that a Byzantine agreement protocol can be used to perform secure broadcasts. Braud-Santoni *et al.* [BGH13] describe the following result. In our proofs, we refer to this algorithm by BA.

Theorem 5 ([BGH13])**.** *There exists an unconditionally-secure protocol for performing secure broadcasts among n parties. The protocol has $\tilde{O}(n)$ amortized communication and computation complexity, and it can tolerate up to $(1/3 - \epsilon)n$ corrupted parties, for any positive ϵ.*

The algorithm of [BGH13] achieves this result by relaxing the load-balancing requirement. If concerned with load-balancing, one can instead use the load-balanced Byzantine agreement of King *et al.* [KLST11] with $O(\sqrt{n})$ blowup.

3.4.3 Our Protocol

We now describe our MPS protocol. Consider two finite fields $\mathbb{F}_p$ and $\mathbb{F}_q$ of prime orders p and q respectively. The high-level idea is as follows: for each party P_i holding an input $x_i \in \mathbb{F}_p$, a uniform and independent random value $r_i \in \mathbb{F}_q$ is chosen to form an input pair (r_i, x_i), where $i \in [n]$. Then, the sequence of pairs $((r_1, x_1), ..., (r_n, x_n))$ is sorted according to the first elements of the pairs. We show that, for sufficiently large prime q, this algorithm randomly shuffles the sequence of inputs $(x_1, ..., x_n)$ with high probability.

To compute this functionality securely, we construct the circuit shown in Figure 3.3, which we denote by $\mathcal{M}$. This circuit consists of the probabilistic sorting circuit of [LP90] augmented by n input gates; the functionality of each gate is computed by a quorum. $\mathcal{M}$ is created jointly by all parties before the protocol starts during an input-independent setup

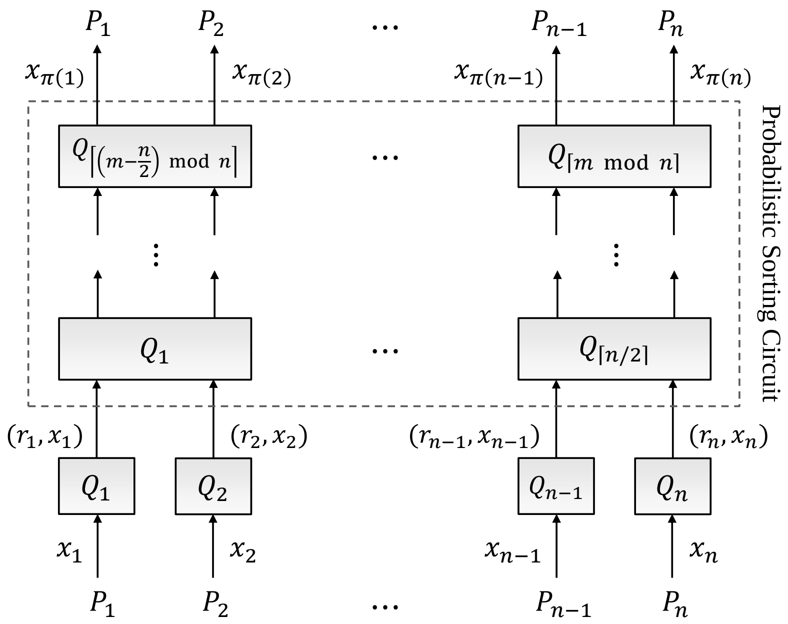


Figure 3.3: MPS circuit

phase. Then, it is jointly evaluated by all parties possibly many times to shuffle many input sequences[5].

The circuit $\mathcal{M}$ is constructed in the following way. First, we create n quorums $Q_1, ..., Q_n$ each with $N = O(\log n)$ parties. We assign each party P_i to Q_i, for all $i \in [n]$. This quorum is responsible for receiving P_i's input x_i and choosing a random value r_i on behalf of P_i. Now, let C denote the probabilistic sorting network of [LP90] and $m = \Theta(n \log n)$ be the number of gates in C.

For all $j \in [m]$, we assign the j-th gate of C to $Q_{(j \bmod n)}$. This quorum is later used for secure evaluation of the gate's functionality. For simplicity of notation, we assume the quorums associated with the output gates of C are $Q_1, ..., Q_{\lceil n/2 \rceil}$.[6] When used to receive inputs of $\mathcal{M}$, we refer to $Q_1, ..., Q_n$ as *input quorums*. When used to send outputs of $\mathcal{M}$ to all parties, we refer to $Q_1, ..., Q_{\lceil n/2 \rceil}$ as *output quorums*.

Creating the probabilistic sorting circuit C requires $O(\log^2 n)$ random bits known to all parties. We generate these bits by asking one of the quorums to agree on a sequence R of $O(\log^2 n)$ random bits, and then send R to all parties via a binary tree of quorums. This randomness is then used by the parties to agree on the structure of C using the random

[5] This setup phase is information-theoretically secure and does not rely on one-time pads. Thus, the same $\mathcal{M}$ can be used *any* number of times for shuffling many input sequences.

[6] Note that a quorum can be *re-used* any number of times for local computations as long as its inputs for each use are secret-shared independently from other uses.

butterfly tournament procedure described in [LP90].

To ensure privacy, every quorum in $\mathcal{M}$ receives and maintains its inputs in a secret-shared format, *i.e.*, each party receives only a share of each input rather than the actual input. Moreover, all computations in these quorums are performed over secret-shared values. 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). As a result, the parties agree on a random polynomial $f(x)$ such that $f(0) = s$, and the i-th party receives $f(i)$ as his verified share of s.

Protocol 3.1 shows our main protocol, where $\mathcal{M}$ is evaluated level-by-level until the final outputs are generated by the output quorums. For all $i \in [n]$, parties in the output quorum Q_i send their shares directly to P_i who reconstructs the corresponding secret $x_{\pi(i)}$, where π denotes the permutation generated by the circuit.

It is left to implement two subprotocols used in Protocol 3.1: Renew-Shares and Ran-Gen. In Section 3.4.3.1, we describe Renew-Shares as a protocol that allows parties of a quorum to securely send a secret-shared value to parties of another quorum. In Section 3.4.3.2, we describe Ran-Gen as a protocol that allows a group of parties to agree on a uniformly random value. We prove the security of Protocol 3.1 (and Theorem 10) in Section 3.4.4. In particular, we show that for sufficiently large $k > 0$ and $q = \Omega(kn^2 \log n)$, this protocol provides almost t-secure MPS with high probability.

3.4.3.1 Share Renewal

Once the computation of each gate is finished, parties in the quorum associated with that gate send the secret-shared result to any quorums associated with gates that need this result as input. Let Q denote a quorum at which the computation of a gate has finished, and let Q' denote a quorum that requires the output of that computation. In order to secret-share the result to Q' without revealing any information to any individual party (or to any coalition of corrupted parties in both Q and Q'), a *fresh sharing* of the result must be distributed in Q'. If s is secret-shared using a polynomial $f(x)$ of degree t, then a fresh sharing of s is a new secret sharing of s defined using another polynomial $g(x)$ of degree t chosen uniformly and independently at random. We refer to the problem of generating a fresh sharing of a secret-shared value among a new set of parties as *share renewal*.

Handling the share renewal problem efficiently and robustly is challenging. Dani *et al.* [DKMS12] solve it by masking the result in Q using a fresh random value and unmasking it in Q'. Although their approach is secure against up to $T < N/3$ corrupted parties in each quorum, they do not provide an explicit construction and simple constructions seem very expensive in terms of both communication and computation costs.[7]

Boyle *et al.* [BGT13] overcome this problem by sending encrypted inputs to only one quorum which does all of the computation using fully-homomorphic encryption. This is not load-balanced, as it incurs a large computation and communication overhead to parties in that quorum. Zamani *et al.* [ZMS14] propose a simple technique for this problem that is, unfortunately, secure only against up to $T < N/6$ corrupted parties in each quorum.

We now describe a novel technique for share renewal that is secure against up to $T < N/3$ corrupted parties in each quorum. Let s denote the output of Q that is secret-shared among parties in Q using a random polynomial $f(x)$ of degree t. Our technique is based on the observation that if every share of s is *reshared* using a fresh random polynomial, then a specific linear combination of the new shares defines a new random polynomial $g(x)$ such that $g(0) = s$. This was first observed by Gennaro *et al.* [GRR98] as a simple method for polynomial randomization and degree-reduction in the multiplication protocol of [BGW88].

Let $g(x) = s + a_1x + ... + a_Tx^T$. Our goal is to calculate the coefficients $a_1, ..., a_T$. Following [GRR98], we write

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{N-1} \\ \vdots & & & \\ 1 & N & \cdots & N^{N-1} \end{bmatrix} \begin{bmatrix} s \\ a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(N) \end{bmatrix},$$

where $a_{T+1}, \cdots, a_N = 0$. The matrix above is an N-by-N Vandermonde matrix that is non-singular and hence is invertible. Let $\begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_N \end{bmatrix}$ be the first row of the inverse matrix. Thus, $s = \lambda_1 f(1) + ... + \lambda_N f(N)$. For all $i \in [N]$, consider a fresh polynomial $h_i(x)$ of degree T, where $h_i(0) = f(i)$. We define $g(x) = \sum_{i=1}^{N} \lambda_i h_i(x)$. Since $g(0) = \lambda_1 f(1) + ... + \lambda_N f(N) = s$, the polynomial $g(x)$ defines a fresh sharing of s. Using this, we define our share renewal protocol Renew-Shares in Protocol 3.2.

In the first step of Renew-Shares, we ask each party to reshare its share $f(i)$ by running a protocol called Reshare. This protocol ensures that every corrupted party shares its correct share $f(i)$ instead of some random or maliciously-chosen value. Asharov and Lindell [AL11] implement a protocol (called *subshare*) that ensures this resharing

[7] Their approach relies on the existence of an unmasking circuit securely evaluated by parties in Q'. Such a circuit must implement an error-correcting technique which requires many multiplication gates.

process is done robustly. We refer to this protocol as Reshare. In Section 3.4.4, we prove Renew-Shares is UC-secure against at most $T < N/3$ corrupted parties in each quorum.

3.4.3.2 Random Generation

We define protocol Gen-Rand using a simple and well-known technique for generating uniformly random secret which is done by adding shares of uniformly random secrets received from all parties. Protocol 3.3 describes the protocol.

3.4.3.3 Remarks

In the following, we discuss alternative approaches that could be used to design different MPS protocols from Protocol 3.1.

All-Output MPS. Protocol 3.1 describes a single-output MPS construction, where each party receives only one element of the output sequence. Although this is useful in many applications such as data-oblivious protocols that often use MPS as an intermediate step, an all-output MPS protocol can be used in some applications such as anonymous broadcast. To achieve all-output MPS, the output delivery step of Protocol 3.1 becomes as follows. For all $i \in [n-1]$, parties in the output quorum $Q_{\lceil i/2 \rceil}$ run VSS-Reconst to reconstruct y_i and y_{i+1} and then send (y_i, y_{i+1}) to all n parties. Each party receiving a set of N pairs from each output quorum, chooses one pair via majority filtering and considers it as the output of that quorum.

Remark on Deterministic Sorting Networks. While the probabilistic sorting network of [LP90] is sufficient for us to achieve an almost t-secure MPS with logarithmic latency (Theorem 10), one can instead use a deterministic sorting network such as those of [AKS83a, Bat68] to achieve t-secure MPS (*i.e.*, uniform shuffling) at the expense of increased latency, communication, and computation costs. We are not aware of a sorting network that can result in better asymptotic and practical costs than the sorting network of [LP90] in terms of latency, communication, and computation costs.

Remark on Permutation Networks. One approach for solving MPS is to securely evaluate a *permutation network* instead of obliviously sorting random values. A permutation network is a network of *swappers*, where each swapper is a gate with two inputs and two outputs; it permutes the inputs randomly with probability $1/2$. A permutation network with n input wires is typically used to generate a random permutation of n values. A network consisting entirely of switches with swapping probability of $1/2$ cannot generate uniform permutations of n values, because for a network with m swappers, there are 2^m different outcomes. Since $n!$ is not a power of 2, some of the possible $n!$ permutations are generated with higher probability than others.

Waksman [Wak68] suggests an $O(n \log n)$ time and memory algorithm for generating unbiased permutations. The idea is to first choose a permutation uniformly at random and then compute a proper setting of swappers that represents the permutation.[8] Unfortunately, it is not clear how this algorithm can be implemented efficiently in a load-balanced multi-party setting. Czumaj *et al.* [CKLK01] propose a permutation network with $O(1/n^2)$ statistical distance from the uniform distribution. To the best of our knowledge, this network provides the smallest distance among known networks with $\mathsf{polylog}(n)$ depth. Still, this result cannot be used to achieve an almost t-secure MPS (as in Definition 2) because in worst case, the adversary can guess the correct permutation with probability $1/n! + O(1/n^2)$ that is $\omega(1/n!)$.

3.4.3.4 Cryptographic Variant of Protocol 3.1

We now describe a computationally-secure variant of Protocol 3.1 using two cryptographic subprotocols: the VSS protocol of [KZG10] (known as *eVSS*) and the multiplication protocol of [GRR98]. Since eVSS generates commitments over elliptic curve groups, it requires smaller message sizes than other cryptographic VSS schemes such as [GRR98].

Theorem 6 ([KZG10])**.** *There exists a constant-round linear VSS scheme for $t < n/2$ secure against a computationally-bounded adversary.*

Theorem 7 ([GRR98])**.** *There exists a constant-round multiplication protocol secure against a computationally-bounded malicious adversary corrupting up to $\frac{n-1}{2}$ parties.*

Theorem 8. *By plugging the VSS of Theorem 6 and the multiplication protocol of Theorem 7 into Protocol 3.1, each party is required to send $\tilde{O}(1)$ messages of size $\ell = O(\kappa + \log n)$ each and compute $\tilde{O}(\ell)$ operations, where κ is the security parameter. The protocol has latency $O(\log n)$.*

In Section 3.4.5, we empirically compare this cryptographic variant with Protocol 3.1.

[8] Here, we assume each swapper has a control bit that when it is set, the swapper always swaps its two inputs, otherwise it keeps their order.

3.4.4 Security Proof

The error probability in Theorem 10 comes entirely from the following steps of Protocol 3.1 failing to output correct results with some probability:

- *Setup:* Protocol Build-Quorums may fail to create good quorums. Theorem 4 shows this failure happens with probability $o(1)$.
- *Random Generation:* It is possible that two or more input quorums choose exactly the same random elements from $\mathbb{F}_q$. In this situation, we say a *collision* happens. Collisions increase the probability that the adversary can correctly guess the secret permutation generated by the protocol. Lemma 10 proves that, for sufficiently large q, failure due to collisions happens with probability $o(1)$.
- *Sorting:* The sorting circuit of [LP90] may fail to sort correctly with probability $o(1)$.

All other components of our protocol are deterministic and thus have no error probability. For simplicity, we assume the three steps above return without failure.[9] However, even assuming the sorting step of Protocol 3.1 returns without failure, the adversary can still take advantage of the *a priori* knowledge that a σ fraction of the input permutations are never sorted by the circuit, to reduce the set of possible input permutations; thus increasing his chance of correctly guessing the secret permutation. In Lemma 9, we show this *a priori* knowledge increases the chance of the adversary in correctly guessing the secret permutation by only a small (*i.e.*, $o(1)$) amount. Hence, Protocol 3.1 achieves an almost t-secure MPS. We prove this lemma in Section 3.4.4.

Lemma 9. *Protocol 3.1 implements an almost t-secure MPS.*

In Lemma 10, we find suitable values for q (the size of the field of random values) such that the probability of collision is bounded by a sufficiently small value. We prove this lemma by a simple application of the Chernoff bound in Section 3.4.4.

Lemma 10. *For some prime q, let $\mathbb{F}_q$ be the field of random elements generated in the Random Generation step of Protocol 3.1. The probability that a collision happens between any two parties is $o(1)$ if $q = \Omega(kn^2 \log n)$, for some $k > 0$.*

We prove the security of Protocol 3.1 in the universal composability framework. To this end, we define a hybrid model based on the modular composition theorem [Can01], and

[9] For simplicity in our proofs, we assume the subprotocol Build-Quorums is run only once, and it does not run concurrently with any other protocols.

argue that, for any adversary that interacts with our protocol, there exists a simulator such that no environment can tell apart whether it is interacting with a run of the hybrid protocol and the adversary, or with a run of the ideal model of our protocol and the simulator. The following lemma is proved in Section 3.4.4.1.

Lemma 11. *Up to the output delivery step, Protocol 3.1 guarantees the following:*

1. *Any set of $t < (1/3 - \epsilon)n$ parties cannot learn any information about the protocol inputs other than what they can jointly learn solely from their set of inputs.*
2. *Any set of $t < (1/3 - \epsilon)n$ parties cannot prevent the protocol from succeeding.*
3. *The security is maintained under universal composability.*

We now prove Theorem 10. Throughout this section, whenever we talk about a protocol that runs among $N = O(\log n)$ parties belonging to a quorum, we denote the set of indices of the corrupted parties in this quorum by I. We start by proving Lemma 9.

Proof. Let $X = (x_1, ..., x_n)$ be the input sequence and $Y = (y_1, ..., y_n)$ be the output sequence generated by Protocol 3.1 such that, for all $i \in [n]$, $y_i = x_{\pi(i)}$, where $\pi : [n] \to [n]$ is the permutation mapping X to Y. To prove Protocol 3.1 is almost t-secure, we show that the adversary can guess π correctly with probability at most $\frac{1}{(n-t)!}(1 + o(1))$. We do this by measuring the information leaked by the protocol about π to an adversary controlling at most t parties.

Before proceeding, we remark that the set of all permutations (each as a function $f : [n] \to [n]$) over n values is always the same regardless of the values themselves. Formally, let A and B denote any two arbitrary sets, S_A denote the set of all permutations of elements in A, and S_B denote the set of all permutations of elements in B. Then, $S_A = S_B$. This is because a permutation is essentially a function mapping every *position* in a sequence to another *position* in that sequence, and the set of all such functions over n values $\{1, ..., n\}$ is always the same. Throughout this proof, we let S denote the set of all possible permutations over n values. Clearly, $|S| = n!$.

Let H denote the unlinkability set which is the set of all permutations from which the adversary tries to guess π, where $|H| \leq |S|$. In fact, the larger H, the smaller the chance of the adversary in breaking the security of the protocol. If the protocol did not leak any information, then $|H| = |S| = n!$. To show this, let X^+ denote the sequence of inputs to the sorting circuit denoted by C. This sequence contains the elements of X augmented by the random values $r_1, ..., r_n$ generated in the Random Generation step of Protocol 3.1; thus

$X^+ = ((r_1, x_1), ..., (r_n, x_n))$. Let $Y^+ = ((s_1, y_1), ..., (s_n, y_n))$ denote the sequence that C outputs. We say an arbitrary sequence $Z^+ = ((t_1, z_1), ..., (t_n, z_n))$ is equal to Y (and denote $Z^+ = Y^+$) if and only if $y_i = z_i$, for all $i \in [n]$.

In fact, $Z^+ = Y^+$ if and only if t_i is the i-th smallest element in $\{r_1, ..., r_n\}$ conditioned on knowing the $i - 1$ smallest elements. Note that although the inputs to C are values chosen uniformly and independently at random from $\mathbb{F}_q$, the set of permutations that each can map the inputs of C to its outputs is still S because there are n input positions and n output positions. Thus, the number of possible permutations mapping X^+ to Y^+ is $n \cdot (n - 1) \cdot ... \cdot 1 = n! = |S|$. Since for every input sequence X the protocol builds exactly one augmented sequence X^+, the number of permutations mapping X to Y that the protocol can generate is also $n!$.

Even though Protocol 3.1 is capable of generating all $n!$ permutations (that exist in S), it leaks some information allowing the adversary to rule out two subsets of permutations from S making H smaller than S. These subsets are as follows:

1. R_1: The largest subset of S that the adversary can obtain by learning t protocol inputs and their order in the output sequence. This is revealed after the output delivery step of Protocol 3.1 by the coalition of t corrupted parties. By fixing t positions in the input sequence, the adversary rules out $|R_1| = n! - (n - t)!$ permutations from S.

2. R_2: A subset of S consisting of a σ fraction of the permutations in S. These are the permutations that cannot be sorted by C. Thus, the adversary rules out $|R_2| = \sigma|S| = \sigma n!$ permutations from S.

In Lemma 12, we show that, in each run, Protocol 3.1 chooses a permutation from S uniformly and independently at random. Intuitively, this means that, from the adversary's point of view, the elements of R_2 are uniformly distributed over S. Formally, let ψ denote the random variable corresponding to the permutation that the protocol randomly chooses from S. Lemma 12 shows that the adversary has no control over the sequence of random values $(r_1, ..., r_n)$. This means that the events $\psi \in R_1$ and $\psi \in R_2$ are statistically independent. Thus,

$$\begin{aligned}
\Pr(\psi \in R_1 \wedge \psi \in R_2) &= \Pr\left(\psi \in (R_1 \cap R_2)\right) \\
&= \Pr(\psi \in R_1) \cdot \Pr(\psi \in R_2) \\
&= \Pr(\psi \in R_1) \cdot \frac{|R_2|}{|S|} \\
&= \sigma \Pr(\psi \in R_1).
\end{aligned}$$

Since $\Pr(\psi \in R_1) = \frac{|R_1|}{|S|}$ and $\Pr\left(\psi \in (R_1 \cap R_2)\right) = \frac{|R_1 \cap R_2|}{S}$,

$$|R_1 \cap R_2| = \sigma |R_1|.$$

In Lemma 11, we prove that, other than t input values and their order in the output sequence, Protocol 3.1 does not reveal any information about the inputs to the adversary. This means that the adversary cannot learn more information about π other than what it learns from R_1 and R_2. Thus,

$$\begin{aligned} |H| &\geq |S| - |R_1 \cup R_2| \\ &= |S| - |R_1| - |R_2| + |R_1 \cap R_2| \\ &= n! - (n! - (n-t)!) - \sigma n! + \sigma\,(n! - (n-t)!) \\ &= (1-\sigma)(n-t)!. \end{aligned}$$

We now show that, from the adversary's point of view, the elements of H are all equally likely to be the secret permutation. In Lemma 12, we show that the Random Generation step of Protocol 3.1 securely generates a uniform and independent sequence $(r_1, ..., r_n)$ that is completely unknown to the adversary. Since the protocol chooses the permutation π from H according to this random sequence, π is also chosen uniformly and independently at random.

Let $\xi \in H$ denote the random variable corresponding to the permutation guessed by the adversary. The probability that the adversary guesses the correct permutation π is

$$\begin{aligned} \Pr(\xi = \pi) = \frac{1}{|H|} &\leq \frac{1}{(1-\sigma)(n-t)!} \\ &= \frac{1}{(n-t)!}\left(1 + \frac{\sigma}{1-\sigma}\right) \\ &= \frac{1}{(n-t)!}\left(1 + \frac{1}{2^{2^{\kappa\sqrt{\log n}}} - 1}\right) \\ &\leq \frac{1}{(n-t)!}\left(1 + 2^{-2^{k\sqrt{\log n}}}\right) \\ &= \frac{1}{(n-t)!}\,(1 + o(1)) \end{aligned}$$

In the third line, we set $\sigma = O(2^{-2^{\kappa\sqrt{\log n}}})$ from [LP90], for any constant $\kappa > 0$. The fourth line is correct for any constant $k > 1$. The last line is correct because $2^{-2^{k\sqrt{\log n}}} = o(1)$. Therefore based on Definition 1, Protocol 3.1 is almost t-secure.

$\square$

Lemma 12. *The Random Generation step of Protocol 3.1 generates a sequence* $(r_1, ..., r_n)$*, where each element is chosen uniformly and independently at random from* $\mathbb{F}_q$*, and the adversary does not learn anything about the sequence.*

Proof. Based on the security of Gen-Rand shown in Lemma 14, each input quorum Q_i agrees on a uniform and independent random value r_i chosen from $\mathbb{F}_q$. Since at most $T < N/3$ of the parties in Q_i are corrupted, and r_i is kept in the secret-shared format, the adversary cannot learn anything about the sequence and/or maliciously set/change the sequence. □

We now prove Lemma 10. Based on the security of Gen-Rand shown in Lemma 14, all elements generated by the input quorums in the random generation step of Protocol 3.1 are chosen uniformly at random and independent of all other random elements generated throughout the protocol. Let P_i and P_j be two parties and $r_i, r_j \in \mathbb{F}_q$ be the random values assigned to them respectively by their corresponding input quorums. The probability that $r_i = r_j$ is $1/q$. Let X_{ij} be the following indicator random variable and Y be a random variable giving the number of collisions happening between any two parties. We write

$$X_{ij} = \begin{cases} 1, & r_i = r_j \\ 0, & \text{otherwise} \end{cases}, \quad Y = \sum_{i,j \in [n]} X_{ij}.$$

Using linearity of expectations,

$$E(Y) = E\Big(\sum_{i,j\in[n]} X_{ij}\Big) = \sum_{i,j\in[n]} E(X_{ij}) = \frac{1}{q}\binom{n}{2} = \frac{n(n-1)}{2q}.$$

We want to find an upper bound on the probability of collisions using the Chernoff bound that is

$$Pr(Y \geq (1+\alpha)E(Y)) \leq e^{-\frac{\alpha^2 E(Y)}{3}}.$$

To ensure that no collision happens with high probability, we need to have $(1+\alpha)E(Y) < 1$ while $e^{-\frac{\alpha^2 E(Y)}{3}} < \frac{1}{n^k}$, for any $k > 0$. Choosing $\alpha < \frac{1}{E(Y)} - 1$ and solving the inequalities for $E(Y)$ we get

$$e^{-\frac{\alpha^2 E(Y)}{3}} < \frac{1}{n^k} \quad \Rightarrow \quad e^{-\frac{\alpha^2 E(Y)}{3}} < e^{-k\log n} \quad \Rightarrow \quad 1 - \frac{\alpha^2 E(Y)}{3} < -k\log n \quad \Rightarrow$$

$$\frac{(\alpha+1)^2 E(Y)}{3} > k\log n \quad \Rightarrow \quad E(Y) < \frac{1}{3k\log n}.$$

Since $E(Y) = \frac{n(n-1)}{2q} < \frac{1}{3k\log n}$, solving this for q gives the bound $q > \frac{3}{2}kn^2\log n$ and $\alpha < 3k\log n - 1$. □

To complete the proof of Theorem 10, we need to show that, up to the output delivery step, Protocol 3.1 does not reveal any information about the inputs to any party. We prove this in Lemma 11 using the universal composability framework [Can01] briefly reviewed in Section 3.4.4.1.

3.4.4.1 Security Under Composition

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 $\mathcal{F}$ 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 $\mathcal{F}$ 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 real protocol and $\mathcal{A}$, or with a run of the ideal model $\mathcal{F}$ 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 use the *modular composition theorem* [Can00]. This theorem states that, in order to prove the security of $\mathcal{P}$, it is sufficient to compare the ideal model to a *hybrid model* (instead of the real model), where the subprotocols are assumed to be ideally computed by a trusted third-party. This hybrid model is usually called the $(\mathcal{P}_1, ..., \mathcal{P}_\ell)$*-hybrid* model because it involves both a real protocol execution and an ideal trusted third-party computing the subprotocols.

3.4.4.2 Proof of Theorem 10

Before proceeding to the proof, we remark that the error probability in Theorem 10 comes entirely from the possibility that Build-Quorums or the threshold counting procedure may fail to output correct results. All other components of our algorithm 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.[10]

As in [Gol04], we refer to the security in the presence of a malicious adversary controlling t parties *t-security*. For every gate $u \in \mathcal{M}$, we let Q_u denote the quorum associated with u, and 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 $|I| < t$.

In the context of perfectly-secure protocols, Kushilevitz *et al.* [KLR10] show Theorem 9, which helps us derive the UC-security of some of our building blocks. 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 9 ([KLR10])**.** *Every protocol that is perfectly-secure in the stand-alone model and has a straight-line black-box simulator is UC-secure.*

Our goal is to prove the UC-security of Protocol 3.1. Following the modular composition theorem, we first define the ideal functionalities shown in Table 3.2 that correspond to the subprotocols used in Protocol 3.1. We then prove that Protocol 3.1 is t-secure in the (F_{BA}, $F_{\mathsf{VSS\text{-}Share}}$, $F_{\mathsf{VSS\text{-}Reconst}}$, F_{Multiply}, F_{Compare}, $F_{\mathsf{Renew\text{-}Shares}}$, $F_{\mathsf{Gen\text{-}Rand}}$)-hybrid model. Finally, we use Theorem 9 to infer the UC-security of Protocol 3.1.

In order to prove the t-security of Protocol 3.1 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 9 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

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

Table 3.2: Ideal functionalities

Functionality	**Implemented by**	**Refer to**
F_{BA}	Protocol BA	Theorem 5
$F_{\mathsf{VSS\text{-}Share}}$	Protocol VSS-Share	Theorem 3
$F_{\mathsf{VSS\text{-}Reconst}}$	Protocol VSS-Reconst	Theorem 3
F_{Multiply}	Protocol Multiply	Protocol 6.17 of [AL11]
F_{Reshare}	Protocol Reshare	Protocol 6.8 of [AL11]
F_{Compare}	Protocol Compare	Comparison protocol of [DFK+06]
$F_{\mathsf{Renew\text{-}Shares}}$	Protocol Renew-Shares	Protocol 3.2
$F_{\mathsf{Gen\text{-}Rand}}$	Protocol Gen-Rand	Protocol 3.3

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.[11] The view of the adversary is then defined as the combined view of all corrupted parties. Finally, we argue that the view of the adversary from the execution of the hybrid model is indistinguishable from its view of the simulation.

Lemma 13. *Subprotocols BA, VSS-Share, VSS-Reconst, Reshare, Multiply, and Compare are UC-secure.*

Proof. Lindell *et al.* [LLR06] show that any BA protocol in the standard model (such as the protocols of [BGH13, KLST11]) is secure under concurrent general composability. Using Theorem 9, since the security proofs of VSS-Share, VSS-Reconst, Reshare, and Multiply given in [AL11] use straight-line black-box simulators, these protocols are UC-secure. Finally, Compare is shown to be UC-secure in [DFK+06]. □

The ideal functionality $F_{\mathsf{Gen\text{-}Rand}}$ is given in Protocol 3.4. At least $N - T$ of the inputs $\rho_1, \ldots, \rho_N$ are sent by uncorrupted parties and thus are chosen uniformly and independently at random from $\mathbb{F}_q$. Hence, $r = \sum_{i=1}^{N} \rho_i$ is also a uniform and independent random element of $\mathbb{F}_q$.

Lemma 14. *The protocol Gen-Rand is UC-secure.*

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

Proof. We prove the t-security of Gen-Rand in the $F_{\text{VSS-Share}}$-hybrid model which is similar to Protocol 3.3 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 3.5.

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. Following Theorem 9, since our simulator is straight-line and black-box, Gen-Rand is UC-secure. □

Lemma 15. *The protocol Renew-Shares is UC-secure.*

Proof. The corresponding ideal functionality $F_{\text{Renew-Shares}}$ is shown in Protocol 3.6. In this functionality, we denote the ideal functionality of Reshare by F_{Reshare} which is equal to $F_{VSS}^{subshare}$ defined in [AL11]. Using this functionality, a set of parties can verifiably secret-share values that are themselves shares. If a corrupted party P_i provides an invalid secret-sharing of its share s_i, or it remains quiet (in which case $F_{\text{Renew-Shares}}$ sets $s_i = \perp$), $F_{VSS}^{subshare}$ defines a new sharing that represents s_i and uses it in place of the invalid (or missing) sharing. See [AL11] for more details.

We prove in the $(F_{\text{VSS-Share}}, F_{\text{Reshare}})$-hybrid model which is similar to Protocol 3.2 except that every call to VSS-Share and Reshare are replaced with calls to the ideal functionalities $F_{\text{VSS-Share}}$ and F_{Reshare} respectively. The corresponding simulator $\mathcal{S}_{\text{Renew-Shares}}$ is given in Protocol 3.7.

Let Q and Q' be two quorums involved in the share renewal procedure, where parties in Q want to send a secret-share value s to parties in Q'. Consider a corrupted party P. First, if $P \notin (Q \cup Q')$, then elements of VIEW_P are independent of the shares Q sends to Q'. Moreover, elements of VIEW_P are independent of the output of Q' since Q' also renews its outputs.

Second, if $P \in (Q \cup Q')$, VIEW_P consists of at most two secret-shares of s defined using two independently random polynomials. The view of a corrupted party P in Q' only contains *subshares* (*i.e.*, shares of shares) of s that reveal nothing about the original shares. Using these subshares, P can reconstruct only one share of the secret over a new random polynomial that is independent of the shares of the parties in Q. The adversary can obtain at most $N/3$ shares of any secret-shared value during F_{Reshare}; $N/3$ shares of s come from corrupted parties in Q and $N/3$ shares come from s being secret-shared in Q' using another random polynomial. Since F_{Reshare} generates a new random polynomial, the first set of $N/3$

shares are independent of the second set of $N/3$ shares. Since least $N/3 + 1$ shares are required for reconstructing the secret, the two views are indistinguishable. □

We are now ready to prove Lemma 11.

Proof. We prove the security of Protocol 3.1 in the (F_{BA}, $F_{\mathsf{VSS\text{-}Share}}$, $F_{\mathsf{VSS\text{-}Reconst}}$, F_{Multiply}, F_{Compare}, $F_{\mathsf{Renew\text{-}Shares}}$, $F_{\mathsf{Gen\text{-}Rand}}$)-hybrid model. We proved the security of all subprotocols in Lemma 13, Lemma 14, and Lemma 15. The last step of the proof is to show that Protocol 3.1 is secure in the (F_{BA}, $F_{\mathsf{VSS\text{-}Share}}$, $F_{\mathsf{VSS\text{-}Reconst}}$, F_{Multiply}, F_{Compare}, $F_{\mathsf{Renew\text{-}Shares}}$, $F_{\mathsf{Gen\text{-}Rand}}$)-hybrid model. This is done by an induction over all gates of the sorting circuit. The view of the adversary VIEW_I from simulation is simply constructed by collecting all shares held by corrupted parties in the quorums associated with every gate of the circuit. Based on Theorem 9, since we have proved the t-security of Protocol 3.1 using a straight-line black-box simulator, the protocol is UC-secure. □

3.4.4.3 Cost Analysis

We first compute the cost of each step of the protocol separately and then compute the total costs. Let $v_1(n)$ and $v_2(n)$ denote the communication and computation complexity of VSS-Share respectively when it is invoked among n parties. For our unconditional Protocol 3.1, we assume VSS-Share implements the sharing protocol of Theorem 3, and for our cryptographic MPS, we assume VSS-Share implements the sharing protocol of VSS of Theorem 6. Both of these VSS protocols take constant rounds of communication.

- *Setup.* The communication and computation costs are equal to those costs of the quorum building algorithm of Theorem 4 which is $\tilde{O}(1)$ for each party. This protocol takes constant rounds of communication.
- *Input Sharing.* This step invokes VSS-Sharing n times among $N = O(\log n)$ parties. So, this step sends $O(n \cdot v_1(N))$ bits and performs $O(n \cdot v_2(N))$ operations. Since the VSS scheme is constant-round, this step also takes constant rounds of communication.
- *Random Generation.* Gen-Rand sends $O(N \cdot v_1(N))$ messages, performs $O(N \cdot v_2(N))$, and has constant rounds.
- *Sorting.* The sorting network [LP90] has $O(n \log n)$ comparators and depth $O(\log n)$. So, the communication cost of this step is equal to the communication and computation cost of running $O(n \log n)$ instantiations of Compare and Renew-Shares. Compare requires $O(\log q)$ invocation of Multiply (see [DFK+06]) which sends $O(N^4 \cdot v_1(N))$

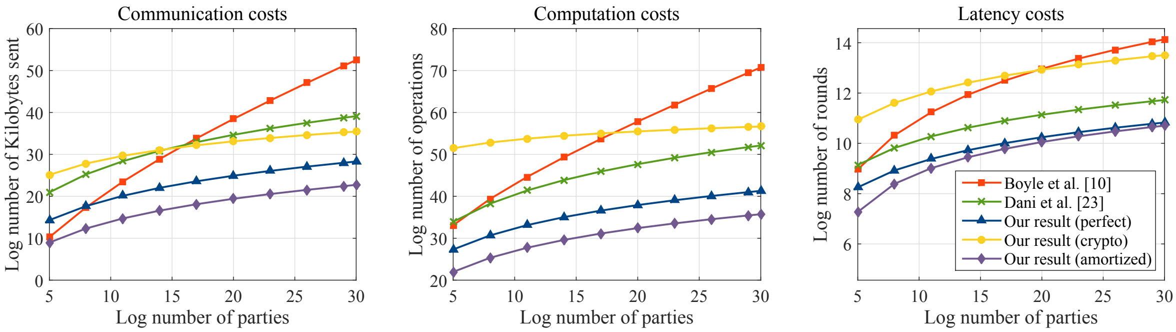


Figure 3.4: Communication cost (left), computation cost (middle), and the number of communication rounds (right)

messages and computes $O(N^4 \cdot \nu_2(N))$ operations. Renew-Shares also sends $O(N \cdot \nu_1(N) + N^3)$ messages and computes $O(N \cdot \nu_2(N) + N^3)$ operations. Hence, the sorting step sends $O(n \log n \cdot \log q \cdot N^4 \cdot \nu_1(N))$ messages computes $O(n \log n \cdot \log q \cdot N^4 \cdot \nu_2(N))$. Since the sorting circuit has depth $O(\log n)$, and Compare takes constant rounds, this steps takes $O(\log n)$ rounds of communication.

- *Output Delivery.* For each output quorum, the costs of output delivery is equal to the communication and computation costs of sending N elements to one of the parties. The party then locally reconstructs the output by running VSS-Reconst over at most N shares. Thus, the total communication and computation complexity of this step is $O(n \cdot N)$. This step takes only one round of communication.

- *Total.* Since $q = \Omega(kn^2 \log n)$, for a constant k, we consider $q = O(n^3)$ and $\log q = O(\log n)$. Using the VSS of Theorem 6, we get $\nu_1(N) = \nu_2(N) = N^2 = O(\log^2 n)$. Thus, our cryptographic MPS protocol sends $O(n \log^8 n)$ messages of size $\ell = O(\kappa + \log n)$ each and computes $O(n\ell \log^8 n)$ operations. This proves Theorem 8. For the costs of Theorem 10 (*i.e.*, our unconditional result), since $\nu_1(N) = \nu_2(N) = O(\mathsf{poly}(N))$, Protocol 3.1 sends $\tilde{O}(n)$ bits and computes $\tilde{O}(n)$ operations. In both cases, the output delivery step costs $O(n \log n)$ field elements. Finally, in both cases, the protocol requires $O(\log n)$ rounds of communication.

3.4.5 Simulation Results

To study the feasibility of our scheme and compare it to previous work, we simulated a proof-of-concept prototype of our protocol (and the cryptographic variant described in Section 3.4.3.4) along with two others that are based on a similar model to ours. These protocols are due to Dani *et al.* [DKMS12] and Boyle *et al.* [BGT13]. To the best of our knowledge, these protocols are the most efficient in terms of communication cost, computation cost, and the number of rounds for large networks. Since the protocols of [DKMS12] and [BGT13] are general MPC algorithms, we use them for computing our (single-output) shuffling functionality described in Section 3.4.3. We are interested in evaluating our protocols for large networks; thus, our choice of protocols for this section is based on their scalability for large values of n.

The prototypes are written in C#, using .NET Framework 4.5, NTL 6.1, GMP 6.0, PBC[12] 0.5.14, and PolyCommit[13] libraries. We ran the simulations on an Intel Xeon E5 machine running Windows 8.1. We simulated our cryptographic protocol for inputs chosen from the field of integers with a 160-bit prime; this ensures about 80 bits of security. We set the parameters of our protocols in such a way that we ensure the probability of error for the quorum building algorithm of [BGH13] is smaller than 10^{-5}. For the sorting circuit, we set $k = 2$ to get $\sigma < 10^{-8}$ for all values of n in the experiment. Clearly, for larger values of n, the error becomes superpolynomially smaller, *e.g.*, for $n = 2^{25}$, we get $\sigma < 10^{-300}$. For all protocols evaluated in this section, we assume cheating (by corrupted parties) happens in every round of the protocols. This is done by having $t = \lfloor n/3 \rfloor$ of the parties send random message in every round of the protocols.

Figure 3.4 illustrates the simulation results obtained for various network sizes between 2^5 and 2^{30} (*i.e.*, between 32 and about 1 billion). To get a system-independent estimation of the computation costs, we implemented a wrapper that counts the number of processor instructions evaluated during the execution of each protocol. We repeat each experiment five times and report the average for each network size. To better compare the protocols, the vertical and horizontal axis of all plots are scaled logarithmically.

In Figure 3.4, we report results from three different versions of our protocols. The first plot (marked with triangles) refers to our unconditionally-secure protocol (Protocol 3.1). The second plot (marked with circles) represents the cryptographic variant of Protocol 3.1 described in Section 3.4.3.4. The third plot (marked with diamonds) shows the cost of our unconditionally-secure protocol with amortized (averaged) setup cost. To obtain this plot, we run the setup phase of Protocol 3.1 once and then use the setup data to run the online protocol 100 times. The total number of bits sent was then divided by 100 to get the average cost.

We observe that our protocol performs significantly better than the prior work. For

[12] http://crypto.stanford.edu/pbc

[13] https://crysp.uwaterloo.ca/software

example, for $n = 2^{15}$ (about 33 thousand parties), our amortized protocol requires each party to send about 128MB of data, while the protocols of [BGT13] and [DKMS12] each send more than one terabyte of data per party. For the computation cost, our amortized protocol requires each party to perform about one billion operations, while the other protocols require each party to perform more than 10^{13} operations. Finally, our amortized protocol requires about 500 rounds of communication, while the protocols of [BGT13] and [DKMS12] require about 1500 and 4100 rounds of communication respectively.

3.5 Application: Privacy-Preserving Location-Based Services

In this section, we describe the problem of privacy-preserving location-based services (LBS), discuss its current challenges, and explain reasons why we believe our multi-party shuffling algorithm can be used to alleviate these challenges. We start by motivating the importance of privacy-preserving LBS.

With the rise of smartphones and tablets that are equipped with various positioning systems (like GPS) and that share locational data over the Internet constantly, user privacy is now becoming a more challenging problem than ever. Mobile users frequently ask location-based services to find points of interest near them, to receive information about traffic along their route, and to receive customized advertising. Such data can be used by others (e.g., companies and governments) for precise surveillance and compromising user privacy. Documents provided by Edward Snowden in 2013 show that NSA is collecting data about the locations of millions of cell phones by tapping into mobile networks [GS13]. Such data can be used to track the movements of individuals precisely and to find relationships between people by correlating various patterns in the locational data.

So far, most privacy-preserving LBS have assumed the existence of a trusted third party [GG03, GL05, MCA06, BLPW08], which is often used for anonymizing location data. Unfortunately, finding such trusted parties in practice is usually very hard or even impossible. Due to the huge number of mobile users interested in sharing their location data frequently, such centralized parties should be powerful servers that are usually owned by large companies and governments. Moreover, centralized parties may be subject to governmental control, and may be banned or forced to disclose sensitive location information. In this section, we describe our protocol for the problem of secure location sharing.

LBS Privacy Approaches. Main approaches for achieving location privacy can be divided into two categories that provide either *location confidentiality* or *location anonymity*. Location confidentiality is to hide[14] user's location and to process locational queries over hidden data. Such data are never revealed to any authority although it might be possible to link the user's identity to its hidden query. Location anonymity, on the other hand, is to hide the connection between a locational query and the user who issues the query. In anonymity terminology, this is often called *unlinkability*, which means that a particular message is not linkable to any sender (or recipient), and that to a particular sender (recipient), no message is linkable [PK01].

While location confidentiality provides a great level of privacy, processing queries over hidden (e.g., encrypted) data is usually hard and expensive. For example, homomorphic encryption [Gen09] and private information retrieval [CKGS98] can be used for privacy-preserving query processing but known such techniques are still computationally intensive. On the other hand, if the number of users having queries is large, which is often the case for LBS, then the queries can be anonymized efficiently to provide a strong level of privacy.

Resistance to Traffic-Analysis. One challenging problem with most anonymity-based location services is resistance against *traffic-analysis*. A global adversary can sniff traffic exchanged between the user and the service provider to link a query containing location information to the user who has issued that query. Such a powerful adversary was assumed to be unrealistic in the past but it might be realistic today if the service provider is controlled or compromised by a state-level surveillance authority [FF14]. Unfortunately, most anonymity-providing protocols like Casper [MCA06], Privé [GKS07], and Tor [DMS04] are not secure against traffic analysis attacks.

***k*-Anonymous LBS.** Several LBS are built upon a relaxed notion of anonymity called *k-anonymity* [GG03, MCA06, GKS07, GL05], where the adversary is assumed to be unable to identify the actual sender/receiver of a locational query from a set of k parties (called *anonymity set*). Even though *k-anonymity* often increases efficiency significantly, choosing small k's can result in severe privacy problems. For example, attackers often have background knowledge and it is shown that small anonymity sets are likely to leak privacy when attackers have such knowledge [MKGV07]. We argue that this is often the case for location anonymity as queries issued by people in different locations usually contain side information about their location. For example, a person located in New Mexico is more

[14] By hiding, we mean using techniques like encryption, steganography, and secret sharing for obscuring data.

likely to search for a restaurant serving chili stew than a person in Vermont. Thus, we believe an algorithm is needed that scales well with the size of anonymity set, k.

Location Cloaking. In a seminal work, Gruteser and Grunwald [GG03] introduced two key ideas for achieving location k-anonymity called *spatial cloaking* and *temporal cloaking*. In spatial cloaking, a client's location (e.g., a two-dimensional point) is converted into a spatial area such that there are k mobile clients located in the area. While a spatially-cloaked location can be calculated efficiently (e.g., using the techniques of [GG03, GL05, MCA06, GKS07]), the inaccuracy associated with the cloaked location often results in sending unnecessary information back to the user. In temporal cloaking, location anonymity is achieved by delaying the user's query until k clients also issue their queries. One drawback of this method is increased latency, especially when k is large. On the other hand, with the fast growth of the number of mobile users sharing their location information, this latency problem is becoming less important.

3.5.1 Previous Work

Using anonymity for location privacy was first proposed by Kong and Hong [KH03]. They propose an anonymous routing protocol called ANDOR that targets mobile ad-hoc networks. They address two closely-related unlinkability problems, namely route anonymity and location privacy. Based on a route pseudonymity approach, ANODR prevents the adversary from exposing local wireless transmitters' identities and tracing network packet flows. For location privacy, their protocol ensures that the adversary cannot discover the real identities of local transmitters.

Gruteser and Grunwald [GG03] show that location data introduces new and potentially more severe privacy risks than network addresses pose in conventional services. Moreover, they analyzed the technical feasibility of k-anonymous location-based services and showed the privacy risks can be reduced through it. They propose an algorithm and service model that that can be used by a centralized location broker service and guarantee k-anonymous location information.

Zhong et al. [ZGH07] propose three protocols, Louis, Lester and Pierre, to achieve location privacy for a service that alerts a person of his nearby friend. Location privacy guarantees that users of the service can learn a friend's location if and only if the friend is actually nearby. Their approach was based on secure two-party computation and all three protocols exploit homomorphic encryption. The Louis protocol requires a semi-trusted third party that does not learn any location information. The Lester and Pierre protocols do not need a third party. The Lester protocol has the drawback that a user might be able to learn a friend's location even if the friend is in an area that is no longer considered nearby by the friend. The Pierre protocol does not have this disadvantage at the cost of not being able to tell the user the precise distance to a nearby friend.

The Casper framework of Mokbel et al. [MCA06] consists of two main components called *location anonymizer* and *privacy-aware query processor*. The location anonymizer is a trusted third party that acts as a middle layer between mobile users and the location-based database server. It receives the location information, blurs the information into cloaked spatial areas, and sends the cloaked spatial areas to the location-based database server. The authors also design a privacy-aware query processor that helps database server to deal with anonymous queries and cloaked spatial areas rather than the exact location information. Unfortunately, the cloaked locations in Casper are usually very large and the algorithm lacks a mechanism to dynamically determine the size of cloaking regions.

Ghinita et al. [GKS07] propose a decentralized model that helps mobile users self-organize into a fault-tolerant overlay network in order to run privacy-preserving anonymous location-based queries. Their protocol develops a rectangular area enclosing k users called *k-Anonymous Spatial Region (k-ASR)* in order to guarantee query anonymity even if the attacker knows the locations of all users. k-ASRs are built in a decentralized fashion, therefore the bottleneck of a centralized server is avoided. The empirical results confirm that the system achieves efficient and scalable anonymization and load-balancing with low maintenance overhead, while being fault-tolerant. On the other hand, the protocol does not provide provable security guarantees and can only tolerate non-adversarial fail-stop faults.

Gedik and Liu [GL05] develop a k-anonymizer that is run by a trusted server. Their algorithm enables each mobile client to specify the minimum level of anonymity it desires and the maximum temporal and spatial tolerances it is willing to accept when requesting for k-anonymity. The authors propose a location cloaking algorithm called *CliqueCloak*, which combines the ideas of spatial and temporal cloaking. Unfortunately, the algorithm is expensive and shows poor performance for large k as it relies on the ability to locate a clique in a graph to perform location cloaking.

The PrivacyGrid framework of Bamba et al. [BLPW08] is composed of dynamic grid-based spatial cloaking algorithms for providing location k-anonymity and location ℓ-diversity (as defined in [MKGV07]) for mobile environments. The algorithms find the smallest possible cloaking region meeting desired privacy levels by measuring several metrics for k-anonymity and ℓ-diversity. While the algorithms are shown to be highly efficient and to achieve higher anonymization success rate, they rely on a trusted server for location tracking and anonymization service.

3.5.2 LBS via MPS

Consider n parties $P_1, P_2, ..., P_n$ each having a locational query $x_i \in \mathbb{Z}_p$, for a prime p and all $i \in [n]$. The parties want to anonymously send their queries to a location-based server and receive the results back. Our MPS protocol can be used to shuffle the locational queries among the parties. Since MPS guarantees the user inputs cannot be tracked to the corresponding users, this allows a traffic analysis resistant algorithm for LBS. In order to adapt the MPS algorithm described before in this chapter for LBS, we need to replace the Output Delivery stage of our MPS algorithm with the following:

- **Query Processing.** Let $(\langle s_i \rangle, \langle y_i \rangle)$ and $(\langle s_{i+1} \rangle, \langle y_{i+1} \rangle)$ be the pairs of shared values each output quorum Q'_i receives. Parties in Q'_i run $z_i = \mathsf{Reconst}(\langle y_i \rangle)$ and $z_{i+1} = \mathsf{Reconst}(\langle y_{i+1} \rangle)$ and send z_i and z_{i+1} to the LBS. For all the z_i and z_{i+1} messages received from parties of Q'_i, the location-based server picks two via majority filtering. Then, the server processes the queries $z_1, ..., z_n$, and sends the results to all parties.

3.6 Conclusion

In this chapter, we first described a Byzantine-resistant protocol for fully-anonymous broadcast in large-scale networks that has a total communication complexity of $\tilde{O}(n)$ bits per anonymous bit and a polylogarithmic latency. To the best of our knowledge, this is the first result that ensures provable anonymity against an unbounded adversary that sends asymptotically less than $O(n^2)$ bits per anonymous bit.

Although the $\tilde{O}(n)$ bit complexity of our protocol scales very well comparing to the state-of-the-art, the actual bandwidth cost is still very high. This is mainly due to the large constants in the communication complexity of the scalable MPC algorithm and the large number of gates in our circuit. It may be possible to decrease the message cost significantly in practice by using threshold cryptography to speed up Byzantine agreement [YKGK10], which is used repeatedly in the MPC algorithm for simulating a reliable broadcast channel.

In this chapter, we also described a multi-party shuffling protocol that is fully decentralized and tolerates up to $t < (1/3 - \epsilon)n$ malicious faults. Moreover, our protocol is load-balanced and can tolerate traffic-analysis attacks. The amount of information sent and the number of computations performed by each party scales polylogarithmically with the number of parties. Scalability is achieved by performing local communications and computations in groups of logarithmic size.

Several open problems remain. First, can we decrease the number of rounds of our protocol using a smaller-depth sorting circuit? For example, since our protocol sorts uniform random numbers, it seems possible to use a smaller depth non-comparison-based sorting circuit like bucket sort. Second, can we improve performance even further by detecting and blacklisting parties that exhibit adversarial behavior? Finally, can we adopt our results to the asynchronous model of communication? We believe that this is possible for a suitably chosen upper bound on the fraction of faulty parties.

Algorithm 3.1 Secure Multi-Party Shuffling Scheme

Inputs. For all $i \in [n]$, party P_i holds an input x_i. Let C denote the probabilistic sorting network of [LP90] and d denote its depth.

Goal. Parties jointly compute a random shuffle of their inputs.

The protocol:

1. **Setup.**
   a) Parties run Build-Quorums to agree on n quorums $Q_1, ..., Q_n$.
   b) Parties in Q_1 run Gen-Rand and VSS-Reconst repeatedly to generate a sequence R of $\Theta(\log^2 n)$ random bits.
   c) Parties in Q_1 send R to all other quorums in the following way. For all $i \in \{2, .., n\}$, parties in Q_i receive R from $Q_{\lfloor i/2 \rfloor}$, and then send it to all parties in Q_{2i} and Q_{2i+1}.
   d) For all $i \in [n]$ and $j \in [m]$, parties assign Q_i to P_i and $Q_{(j \bmod n)}$ to the j-th gate of C, and connect the gates based on the random butterfly tournament described in [LP90] and the random sequence R.
2. **Input Sharing.** Party P_i VSS-shares his input x_i with Q_i.
3. **Random Generation.** Parties in input quorum Q_i perform the following steps:
   a) Run Gen-Rand to generate a random secret-shared value $r_i \in \mathbb{F}_q$, where $q > \frac{3}{2}kn^2 \log n$ for any $k > 0$.
   b) Run Renew-Shares to send the secret-shared pair (r_i, x_i) to $Q_{\lceil i/2 \rceil}$.
4. **Sorting.** C is evaluated level-by-level starting from the input gates. For each gate G in C and quorum Q assigned to G, parties in Q perform the following steps:
   a) **Comparison.** Let (r, x) and (r', x') be the secret-shared inputs of G. The parties run Compare to securely compare the secret-shared values r, r'. Let $\rho = (r \leq r')$ be the resulting secret-shared value. The parties compute the output secret-shared pairs (s, y) and (s', y') from

   $$\begin{aligned} s &= \rho \cdot r + (1 - \rho) \cdot r', & y &= \rho \cdot x + (1 - \rho) \cdot x' \\ s' &= \rho \cdot r' + (1 - \rho) \cdot r, & y' &= \rho \cdot x' + (1 - \rho) \cdot x \end{aligned}$$

   For every addition of secret-shared values a, b, parties locally compute $a + b$. For every multiplication, they run Multiply.
   b) **Output Resharing.** Parties run Renew-Shares to send secret-shared values s, y, s', y' to the parent quorum.
5. **Output Delivery.** For all $i \in [n - 1]$, let (s_i, y_i) and (s_{i+1}, y_{i+1}) be the pairs of secret-shared values the output quorum $Q_{\lceil i/2 \rceil}$ computes in the previous step.
   a) Each party in this quorum sends his share of y_i to party P_i and his share of y_{i+1} to party P_{i+1}.
   b) Parties P_i and P_{i+1} run VSS-Reconst to reconstruct y_i and y_{i+1} respectively.

Algorithm 3.2 Renew-Shares

Inputs. A set of parties $P_1, ..., P_N$ jointly hold a secret-shared value s, *i.e.*, a polynomial $f(x)$ of degree $T < N/3$ is defined such that $f(0) = s$, and for all $i \in [N]$, P_i holds $f(i)$.

Goal. Generate a fresh sharing of s among another group of parties $P'_1, ..., P'_N$. This means that the protocol must calculate a polynomial $g(x)$ of degree T uniformly and independently at random such that $g(0) = s$, and for all $j \in [N]$, P'_j holds $g(j)$.

The protocol:

1. Each party P_i runs Reshare to VSS-share $f(i)$ among $P'_1, ..., P'_N$ using a random polynomial $h_i(x)$ of degree T such that $h_i(0) = f(i)$.

2. Each party P'_j locally computes its share of s from $g(j) = \sum_{i=1}^{N} \lambda_i h_i(j)$.

Algorithm 3.3 Gen-Rand

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

The protocol:

1. For all $i \in [N]$, party P_i chooses $\rho_i \in \mathbb{F}_q$ uniformly at random and VSS-shares it among all N parties.

2. For all $j \in [N]$, let $\rho_{1j}, ..., \rho_{Nj}$ be the shares P_j receives from the previous step. P_j computes $r_j = \sum_{k=1}^{N} \rho_{kj}$.

Algorithm 3.4 $F_{\textsf{Gen-Rand}}$

Goal. For a gate $u \in \mathcal{M}$, 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 $F_{\textsf{VSS-Share}}$ to send a share r_i of r to P_i.

Algorithm 3.5 $\mathcal{S}_{\textsf{Gen-Rand}}$

Inputs. For a gate $u \in \mathcal{M}$, 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 $F_{\textsf{VSS-Share}}$ with dummy input 0. Let $s^i_1, ..., s^i_N$ denote the outputs.
2. For every $P_j \in I_u$,
   a) Run $F_{\textsf{VSS-Share}}$ with input ρ_j. Let $\rho^j_1, ..., \rho^j_N$ denote the outputs. For every $k \in [N]$, add ρ^k_j to the view of P_j.
   b) Compute $r_j = \sum_{k=1}^{N} \rho^k_j$ and add r_j to the view of P_j.

Algorithm 3.6 $F_{\textsf{Renew-Shares}}$

Goal. Given a secret s shared among a group of parties $P_1, ..., P_N$, generate a fresh sharing of s among another group of parties $P'_1, ..., P'_N$.

Functionality:

1. Receive inputs $s_1, ..., s_N$ from $P_1, ..., P_N$ respectively. For every $i \in [N]$, if P_i does not send an input, then define $s_i = \perp$.
2. For every $i \in [N]$, invoke $F_{\textsf{Reshare}}$ to generate a sharing of s_i over a polynomial $h_i(x)$ of degree T such that $h_i(0) = s_i$.
3. For every $j \in [N]$, compute $g(j) = \sum_{i=1}^{N} \lambda_j h_i(j)$ and send $g(j)$ to P'_j.

Algorithm 3.7 $\mathcal{S}_{\textsf{Renew-Shares}}$

Inputs. The inputs $\{s_j\}_{j \in I}$ and outputs $\{g(j)\}_{j \in I}$ of the corrupted parties.

Simulation:

1. For every $i \notin I$, call $F_{\textsf{Reshare}}$ with dummy input 0. Let $s^i_1, ..., s^i_N$ denote the outputs.
2. For each $j \in I$,
   a) Run $F_{\textsf{Reshare}}$ with input s_j. Let $s^j_1, ..., s^j_N$ denote the outputs. For every $k \in [N]$, add s^k_j to the view of P_j.
   b) Compute $g(j) = \sum_{k=1}^{N} \lambda_j h_k(j)$ and add $g(j)$ to the view of P_j.

Chapter 4

Blocking-Resistant Tor Bridge Distribution

4.1 Introduction

Today, mass surveillance and censorship severely threaten democracy and freedom of speech. Governments around the world control the Internet to protect their domestic political, social, financial, and security interests. This makes anonymity a crucial tool for preserving privacy of individuals in cyberspace. Tor [DMS04] is the most popular anonymous communication network with more than 2.5 million users per day [Tor15a]. Tor relays the Internet traffic via more than 6,000 volunteer nodes called *relays* spread across the world. Tor users can connect to the network and have their Internet data routed through the network before reaching any server, thus the servers are not able to distinguish between Tor users or locate them.

Since the list of all relays is available publicly, state-sponsored organizations can enforce Internet service providers (ISPs) to block access to all of them making Tor unavailable in territories ruled by the state. When access to the Tor network is blocked, Tor users have the option to use *bridges* – volunteer relay nodes that are not all listed in Tor's public directory [DM06]. Bridges serve only as entry points into the rest of the Tor network, and their addresses are carefully distributed to the users, with the hope that they cannot be all learned by censorship authorities. Tor users behind censorship firewalls must find bridges through email, uncensored web sites, etc.

Although currently the bridges are distributed among the users based on different strategies such as CAPTCHA-enabled email-based distribution and IP-based distribution [DM06], studies indicate that censors are using sophisticated mechanisms along with a large coalition of corrupt users (scanners) to obtain and block many bridge addresses rendering Tor unavailable for many users [Din11b, WL12, EFW+15].

In this chapter, we study the problem of bridge distribution in Tor, where a set of bridges is distributed among n users, t of whom are controlled by a censor, in such a way that all honest users can access a bridge that is not blocked by the censor. A solution to this problem would allow us to make the Tor network provably available for all users. Unfortunately, state of the art techniques for bridge distribution either cannot guarantee that all honest users can access Tor [WLBH13, MML12] or only work when t is known in advance [Mah10].

In contrast, we describe an algorithm that ensures Tor is available to all honest users with high probability without requiring any *a priori* knowledge about t. It is often hard in practice to estimate the number of corrupt users due to the sophisticated nature of Internet censorship in many countries such as China [Oni12, EFW+15]. Moreover, censors can easily implement strategies to prevent the circumvention mechanism from correctly estimating the number of corrupt users.

Inspired by the *resource competitive analysis* approach of Gilbert *et al.* [GSKY12] and Bender *et al.* [BFM+15], our algorithms provide the following guarantee: if the adversary pays t amount of resource cost, then the resource cost of our algorithms is some function of t. This allows us to achieve near-optimal resource costs with strong robustness to disruptions caused by the censor.

Our main strategy for preventing the censor from blocking a large fraction of the bridges is to use randomization. This is because the colluding censor cannot predict the behavior of the randomized process in distributing a set of bridges to the users, and thus it cannot arrange its corrupt users in such a way that prevents some of the honest users from connecting to the Tor network. Moreover, our algorithm can adaptively adjust the number of bridges required based on the number of bridges compromised so far and guarantees that the number of rounds until all honest users can connect to Tor is bounded by $O(\log t)$. Our algorithms can be run independently from Tor; the Tor network remains intact to focus on its main purpose of providing anonymity.

In Section 4.1.1, we describe our communication and adversarial model. In Section 4.1.2, we state our main theorem. We review the previous results and related work in Section 4.2. In Section 4.3, we describe our algorithms for reliable bridge distribution; we start from a simple algorithm and improve it as we continue.

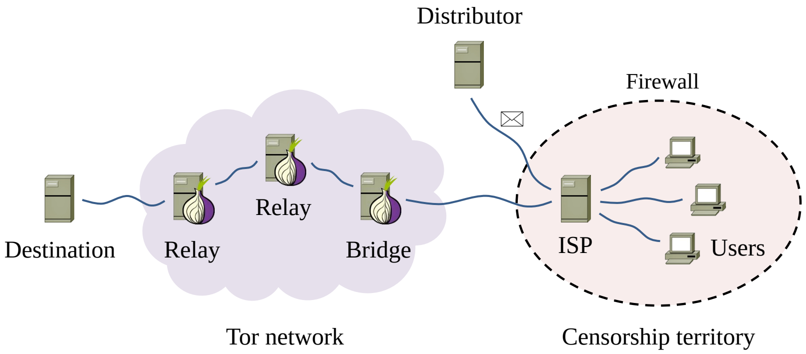


Figure 4.1: Our model

4.1.1 Our Model

Consider n users (clients) connected to the Internet via an ISP inside a censorship territory and a trusted server called the *bridge distributor* (or simply the *distributor*) connected to the Internet via an ISP outside the territory. The distributor has information about a set of m Tor bridges that are also located outside the territory. Each user wants to obtain information about a subset of the bridges at least one of which can be used to connect to Tor. The user sends its request to the distributor using a rate-limited channel[1] such as email and the distributor sends a set of bridges back to the user via the same method of contact.

Among the users, there are $t < n$ *corrupt users* controlled by an adaptive adversary (the *censor*) who can choose the set of corrupt users at the beginning of each round of communication, but is limited to corrupting at most t distinct users combined in all rounds. We refer to the other $n - t$ users as *honest users*. Each corrupt user is willing to obtain information about the bridges in order to block them, but does not have to block a bridge as soon as it finds the bridge; it is allowed to strategically (perhaps by colluding with other corrupt users) to decide when to block a bridge. Moreover, we assume our algorithms do not have any knowledge about t before they begin, and the adversary has no knowledge about the randomness generated by the distributor throughout the algorithms other than what it can learn from the bridges assigned to the corrupt users. Figure 4.1 depicts our model.

4.1.2 Our Results

We prove the following main theorem in Section 4.3.2.

[1] A vital communication mechanism such as Gmail that the censor cannot block due to major economic and political consequences, but is not suitable for interactive communication over the Internet such as web surfing.

Theorem 10. *There exists a bridge distribution algorithm that guarantees the following properties with probability* $1 - 1/n^k$*, for any* $k > 0$*:*

1. *All honest users can connect to Tor after* $\lceil \log t \rceil + 2$ *rounds of communication with the distributor.*
2. *The total number of bridges required is* $O(t)$*.*

The best known algorithm for bridge distribution [Mah10] only works when the number of corrupt users, t, is known in advance and requires at most $k\,(1 + \lceil \log\,(n/k) \rceil)$ bridges. In contrast, our algorithms not only do not require any prior knowledge about t but also use fewer bridges.

4.2 Related Work

The bridge distribution problem can be seen as a special case of the proxy distribution problem, where a set of unblocked servers (proxies) outside the censorship territory are distributed among the users inside the territory. These proxies are used to relay Internet traffic to blocked websites. The proxy distribution problem has been studied by several previous work.

Feamster *et al.* [FBW+03] propose a proxy distribution algorithm that requires every user to solve a cryptographic puzzle to discover a proxy. This way, the algorithm prevents the corrupt users from learning a large number of proxies. Unfortunately, empirical results of [FBW+03] show that a computationally powerful censor can easily block a very large fraction of the proxies.

The Kaleidoscope system of Sovran *et al.* [SLL08] disseminates proxy addresses over a social network whose links correspond to existing real world social relationships among users. Unfortunately, this algorithm assumes the existence of a few internal trusted users who can relay other users' traffic and cannot guarantee its users' access to Tor.

McCoy *et al.* [MML12] propose Proximax; a proxy distribution system that uses social networks such as Facebook as trust networks that can provide a degree of protection against discovery by censors. Proximax estimates each user's effectiveness and chooses the most effective users for advertising proxies with the goals of maximizing the usage of these proxies while minimizing the risk of having them blocked.

Even if bridges are distributed carefully among the users, censors can still block access to the Tor network via deep packet inspection (DPI). The Tor Project has developed a variety of tools under the name *pluggable transports* [Tor15b] that can be used to obfuscate the traffic transmitted between the client and the bridge. This makes it hard for the censor (who monitors the traffic) to distinguish between the legitimate-looking obfuscated traffic and the actual Tor traffic. Although pluggable transports are necessary for preventing bridge blocking via DPI, they cannot prevent blocking via colluding corrupt users. Moreover, recently Wang *et al.* [WDA+15] showed that current obfuscation mechanisms used in Tor can be reliably detected by censors with sufficiently low false-positive rates. In our model, we assume our algorithm runs in parallel with a reliable pluggable transport tool that can prevent DPI blocking.

Mahdian [Mah10] studies the proxy distribution problem when the number of corrupt users, t, is known in advance. He proposes algorithms for both large and small values of t and provides a lower bound for dynamic proxy distribution that is useful only when $t \ll n$.[2] Unfortunately, it is usually hard in practice to reliably estimate the value of t. Mahdian's algorithm for large known t requires at most $k\left(1 + \lceil\log\left(n/k\right)\rceil\right)$ bridges, and his algorithm for small known t uses $O(k^2 \log n / \log\log n)$ bridges.

Wang *et al.* [WLBH13] propose a reputation-based bridge distribution mechanism called rBridge that computes every user's reputation based on the uptime of its assigned bridges and allows the user to replace a blocked bridge by paying some reputation credits. Interestingly, rBridge is the first model to provide user privacy against an honest-but-curious distributor. This is achieved by performing oblivious transfer between the distributor and the users along with commitments and zero-knowledge proofs for achieving unlinkability of transactions.

Our algorithms rely on a technique for testing reachability of bridges from outside the censored territory. Dingledine [Din11a] and Ensafi *et al.* [EKAC14] describe active scanning mechanisms for testing availability of bridges. The details of these methods and their current challenges are out of scope of our work.

4.3 Our Algorithm

In this section, we first construct a bridge distribution algorithm that adaptively increases the number of bridges used with respect to the number of bridges blocked. Before proceeding to our algorithms, we define the standard terms and notation used throughout this chapter.

Notation. We say an event occurs *with high probability*, if it occurs with probability at least $1 - 1/n^k$, for some $k \geq 1$. We denote the set of integers $\{1, ..., n\}$ by $[n]$. We denote a set

[2] For large values of t (*e.g.*, $t = cn$ for $c \in (0, 1)$) the trivial lower bound of $\Omega(t)$ is better than $\Omega\left(\frac{t\log\left(n/t\right)}{\log t + \log\log n}\right)$ of [Mah10].

Algorithm 4.1 Bridge Distribution Scheme

```
1: Initialize parameters: $i \leftarrow 0$; $b_i \leftarrow 1$; $U \leftarrow$ a set of users $\{u_1, ..., u_n\}$
2: while true do
3:     if $b_i \geq 2^i$ then
4:         $i \leftarrow i + 1$
5:         $B_i = \mathsf{Distribute}(U, 2^i)$
6:     end if
7:     $b_i \leftarrow$ number of bridges in $B_i$ that are blocked
8: end while
```

of n users participating in our algorithms by $\{u_1, ..., u_n\}$. We say a bridge is *blocked* when the censor has restricted users' access to this bridge. We refer to the remaining bridges as *unblocked* bridges.

4.3.1 Basic Bridge Distribution Algorithm

Our basic method is a Monte Carlo algorithm that proceeds in rounds (see Algorithm 4.1): in each round, the algorithm randomly distributes a set of bridges among the users and proceeds to the next round once the number of blocked bridges exceeds a threshold that is increased exponentially in each round. In every round, we increase the number of fresh bridges with respect to the number of bridges blocked so far.

We later show that each run of Algorithm 4.1 guarantees that all users can connect to Tor with some constant probability. Thus, if the distributor repeats the algorithm $O(\log n)$ times in parallel, it can guarantee that all users can connect to Tor with high probability.

We refer to a single execution of the while loop in Algorithm 4.1 as an *iteration*. We refer to each increment of the variable i (in line 4) as a *round*. Note that several iterations may correspond to the same round depending on the condition in line 3 of the algorithm. For simplicity, we assume the $O(\log n)$ instances of Algorithm 4.1 run *synchronously* meaning that they start and finish each iteration at the same time.[3] We also assume that each iteration runs *atomically* meaning that the users are allowed to use (or block) the bridges assigned to them (in any round) only at the end of each iteration.

Algorithm 4.2 implements the function Distribute. The algorithm assigns to each user one of the bridges chosen uniformly and independently at random. Each run of Algorithm 4.2 assigns only one bridge to every user in each round. Since the algorithm is

[3] Since the distributor runs the parallel executions locally, it is easy to guarantee that they run synchronously.

repeated several times, possibly over multiple rounds and $O(\log n)$ executions of Algorithm 4.1, every user receives multiple bridges until all honest users are guaranteed to succeed with high probability.

Algorithm 4.2 Function Distribute

Goal: A sequence of m_i bridges is randomly distributed among a set of n users $U = \{u_1, ..., u_n\}$.

```
function Distribute($U, m_i$)
    $B_i \leftarrow$ a sequence of $m_i$ unblocked bridges
    for all $j \in [n]$ do
        Pick an integer $k \in [m_i]$ uniformly at random
        Assign $B_i[k]$ to $u_j$
    end for
    return $B_i$
end function
```

4.3.2 Proof of Algorithm 4.1

We now prove the properties of Algorithm 4.1. For simplicity, we assume a user can connect to Tor in an iteration if and only if at least one unblocked bridge is assigned to it. Although the adversary has a total budget of t corrupt users, only some of the corrupt users might be actively blocking bridges in any given round. From the distributor's perspective, since t is unknown, only those users who have blocked at least one bridge in any round so far are considered corrupt and are counted towards the adversary's total budget. If a corrupt user has only attempted to block bridges that have already been blocked by other corrupt users, then our algorithm obviously cannot identify this user as a corrupt user until the user blocks at least one unblocked bridge in future rounds.

We use the following variables in our proof:

- m_i: the number of bridges distributed in round i.
- b_i: the number of bridges blocked in round i.
- t_i: the number of corrupt users each of whom has blocked at least one bridge in round i.
- B_i: the sequence of bridges distributed in round i.

Lemma 16. *In the i-th round of Algorithm 4.1, if $b_i < 2^i$, then all honest users can connect to Tor with high probability.*

Proof. We first consider the execution of one of the $\lceil c \log n \rceil$ instances of Algorithm 4.1. For each user, Algorithm 4.3 chooses a bridge independently and uniformly at random and assigns it to the user. Without loss of generality, assume the corrupt users are assigned bridges first. For $k = 1, 2, ..., t_i$, let $\{X_k\}$ be a sequence of random variables each representing the bridge assigned to the k-th corrupt user. Also, let Y be a random variable corresponding to the number of *bad* bridges (*i.e.*, the bridges that are assigned to at least one corrupt user) after all t_i corrupt users are assigned bridges. The sequence $\{Z_k = \mathrm{E}[Y|X_1, ..., X_k]\}$ defines a Doob martingale [DP09, Chapter 5], where $Z_0 = \mathrm{E}[Y]$. Since each corrupt user is assigned a fixed bridge with probability $1/m_i$, the probability that the bridge is assigned to at least one corrupt user is $1 - (1 - 1/m_i)^{t_i}$. By symmetry, this probability is the same for all bridges. Thus, by linearity of expectation,

$$\mathrm{E}[Y] = \left(1 - (1 - 1/m_i)^{t_i}\right) m_i \approx (1 - e^{-t_i/m_i})m_i.$$

Since $2^{i-1} \leq b_i < 2^i$, we have $2^{i-1} \leq t_i < 2^i$ because in each round $m_i = 2^i$ bridges are distributed and each corrupt user is assigned exactly one bridge; thus, each corrupt user can block at most one bridge. Hence,

$$(1 - 1/\sqrt{e})m_i \leq \mathrm{E}[Y] < (1 - 1/e)m_i. \tag{4.1}$$

Therefore in expectation, a constant fraction of the bridges become bad in each instance of the algorithm.

Since $|Z_{k+1} - Z_k| \leq 1$, $Z_0 = \mathrm{E}[Y]$, and $Z_{t_i} = Y$, by the Azuma-Hoeffding inequality [DP09, Theorem 5.2],

$$\Pr\left(Y > \mathrm{E}[Y] + \lambda\right) \leq e^{-2\lambda^2/t_i},$$

for any $\lambda > 0$. By setting $\lambda = \sqrt{m_i}$, we have

$$\Pr(Y > \mathrm{E}[Y] + \sqrt{m_i}) \leq e^{-2m_i/t_i} < 1/e^2. \tag{4.2}$$

The last step is because $t_i < m_i$. Therefore, with at most a constant probability, the actual number of bad bridges is larger than its expected value by at most $\sqrt{m_i}$. Therefore, the probability that an honest user is assigned a bad bridge is at most

$$\begin{aligned} \frac{\mathrm{E}[Y] + \sqrt{m_i}}{m_i} &< \frac{(1 - 1/e)m_i + \sqrt{m_i}}{m_i} \\ &= 1 - 1/e + 1/\sqrt{m_i}, \end{aligned} \tag{4.3}$$

where the first step is achieved using (4.1).

Now, let $p_1 = \Pr(Y > \mathrm{E}[Y] + \sqrt{m_i})$, and let p_2 be the probability that an honest user is assigned a bad bridge. From (4.2) and (4.3), we have

$$p_1 < 1/e^2 \quad \text{and} \quad p_2 < 1 - 1/e + 1/\sqrt{m_i}.$$

Thus, the error probability of the algorithm for one user in one round is equal to $p_1 + (1 - p_1)p_2$, which is at most 0.8 for $m \geq 65$.

If the algorithm is repeated $\lceil 15 \log n \rceil$ times in parallel, then the probability that a user is assigned a bad bridge is at most $0.8^{\lceil 15 \log n \rceil} \leq 1/n^3$. By union bound, the probability that any of the n users is assigned a bad bridge in a round is at most $1/n^2$. By Lemma, since the algorithm runs for at most $\lceil \log t \rceil + 2 < n$ rounds, by union bound the total error probability of the algorithm is at most $1/n$. Therefore, all honest users can connect to Tor with high probability. □

Algorithm 4.2 may assign different number of users to each bridge. In the following lemma, we show that each bridge is assigned almost the same number of users as other bridges with high probability.

Lemma 17. *Let X be a random variable representing the maximum number of users assigned to any bridge, and let Y be a random variable representing the minimum number of users assigned to any bridge. We have*

$$\Pr\left(X \geq \frac{e\mu \ln n}{\ln \ln n}\right) \leq \frac{1}{n} \quad \textit{and} \quad \Pr\left(Y \leq \frac{e\mu \ln n}{\ln \ln n}\right) \leq \frac{1}{n},$$

where $\mu = n/m_i$.

Proof. Algorithm 4.3 can be seen as the classic balls-and-bins process: n balls (users) are thrown independently and uniformly at random into m_i bins (bridges). Therefore, the distribution of the number of users assigned to a bridge is approximately Poisson with $\mu = n/m_i$ [MU05, Chapter 5].

Let X_j be the random variable corresponding to the number of users assigned to the j-th bridge, and let $\tilde{X}_j$ be the Poisson random variable approximating X_j. We have $\mu = \mathrm{E}[X_j] = \mathrm{E}[\tilde{X}_j] = n/m_i$. We use the following Chernoff bounds from [MU05, Chapter 5] for Poisson random variables:

$$\Pr(\tilde{X}_j \geq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x > \mu \tag{4.4}$$

$$\Pr(\tilde{X}_j \leq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x < \mu \tag{4.5}$$

We let $x = \mu y$, where $y = ez$ and $z = \frac{\ln n}{\ln \ln n}$. From (4.4), we have

$$\begin{aligned}\Pr(\tilde{X}_j \geq \mu y) &\leq \left(\frac{e^{y-1}}{y^y}\right)^{\mu} \\ &\leq \frac{e^{y-1}}{y^y} \\ &= \frac{1}{e}\left(\frac{1}{z^z}\right)^{e} \\ &\leq \frac{1}{e}\left(\frac{1}{c'n}\right)^{e} \leq \frac{1}{n^2},\end{aligned}$$

for some positive constant c'. The second step is because $y^y > e^{y-1}$ (since $z > 1$) and $\mu > 1$. The last step is from Lemma. □

Lemma 18. *By running Algorithm 4.1, all honest users can connect to Tor with high probability after at most* $\lceil \log t \rceil + 1$ *iterations.*

Proof. Let k denote the smallest number of rounds needed until all users can connect to Tor with high probability. Intuitively, k is bounded, because t, the budget of the adversary, is bounded. Therefore, for all $i \geq k$, we have $b_i < 2^i$, and based on Lemma 16, all users can connect to Tor with high probability. In the following, we find k with respect to t.

The best strategy for the adversary is to maximize k, because this prevents the algorithm from succeeding soon. In each round i, this can be achieved by minimizing the number of bridges blocked (*i.e.*, b_i), while ensuring the algorithm proceeds to the next round. However, the adversary has to block all 2^i bridges distributed in each round to force the algorithm to proceed to the next round. Let ℓ be the smallest integer such that $2^\ell \geq t$. In round ℓ, the adversary has enough budget to cause the algorithm to proceed to round $\ell + 1$. However in round $\ell + 1$, the adversary can block at most $2^\ell < 2^{\ell+1}$ bridges which is insufficient for proceeding to round $\ell + 2$. Therefore, $\ell + 1$ is the last round and $k = \ell + 1$. Since $2^\ell \geq t$, $k = \lceil \log t \rceil + 1$. In other words, if the algorithm is run for at least $\lceil \log t \rceil + 1$ iterations, then with high probability all honest users can connect to Tor.

□

Lemma 19. *The total number of bridges used by Algorithm 4.1 is at most* $(8t - 2)c \log n$.

Proof. In every round $i > 1$, the algorithm distributes a new bridge only to replace a bridge blocked in round $i - 1$. Thus, the total number of bridges used until round i, denoted by N_i, is equal to the number of bridges blocked until round i plus the number of new bridges distributed in round i, which we denote by a_i. Therefore,

$$N_i = a_i + \sum_{j=1}^{i-1} b_j. \tag{4.6}$$

In the i-th round, the algorithm recruits $a_i \leq 2^i$ new bridges, because some of the bridges required for this round might be reused from previous rounds. Since in the i-th round $b_i < 2^i$,

$$N_i < 2^i + \sum_{j=1}^{i-1} 2^j = 2^i + 2^i - 2 = 2^{i+1} - 2.$$

From Lemma 18, it is sufficient to run the algorithm for $\lceil \log t \rceil + 1$ rounds. Therefore,

$$N_i < 2^{\lceil \log t \rceil + 2} - 2 < 8t - 2,$$

i.e., the total number of bridges used by the algorithm is at most $(8t-2)c \log n$. □

4.3.3 Improved Distribution Algorithm

In each run of Distribute, an $\frac{n}{m_i}$-by-m_i matrix is created and each user in U is randomly assigned to one of the elements of the matrix such that all users appear in the matrix and each user appears exactly once. The random assignment of users is done using a random permutation π that maps every integer between 1 and n (corresponding to every element of the matrix) to an integer between 1 and n (corresponding to every user index).

Next, the algorithm recruits a set of w fresh (unblocked) bridges and assigns a unique bridge to all users in each column of the matrix. To improve the efficiency of our algorithm in practice, we assume the bridges that were recruited in previous rounds and remain unblocked are reused for distribution in the next round.

Figure 4.2 shows the matrix generated in each execution of Algorithm 4.2. In this figure, $\pi : [n] \rightarrow [n]$ refers to the random permutation generated in line 3 of the algorithm, and B refers to the sequence of w bridges being assigned to the users in the current run of the algorithm.

Lemma 20. *Let Algorithm 4.1 call the modified version of* Distribute *defined in Algorithm 4.3, and $m_i = 2^{i+1}$ in each round i. Then, Algorithm 4.1 guarantees that all honest users can connect to Tor with high probability, and the total number of bridges used is at most* $(6t-1)c \log n$.

Algorithm 4.3 Fully-Load Balanced Distribute

Goal: A sequence of m_i bridges is randomly distributed among a set of n users $U = \{u_1, ..., u_n\}$.

```
function Distribute($(U, m_i)$)
    $B_i \leftarrow$ a sequence of $m_i$ unblocked bridges
    Define a matrix $M = \left[ u_{\pi\left(i+(j-1)\frac{n}{m_i}\right)} \right]_{\frac{n}{m_i} \times m_i}$ such
    that $\pi : [n] \to [n]$ is a random permutation
    for all $j \in [m_i]$ do
        Assign $B_i[j]$ to users in $j$-th column of $M$
    end for
    return $B_i$
end function
```

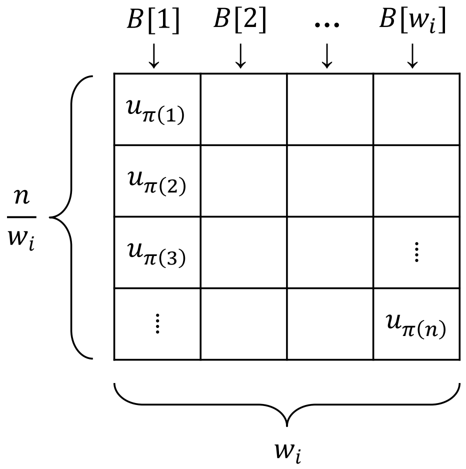


Figure 4.2: Matrices generated by Algorithm 4.2 in round i

Proof. Since the n users are arranged uniformly and independently at random in M, and there are t_i active corrupt users among them, the probability that for a given user u the column that u appears in contains at least one corrupt user is at most

$$1 - \left(\frac{n - t_i}{n}\right)^{n/m_i} = 1 - \left(1 - \frac{t_i}{n}\right)^{n/m_i} \le \frac{t_i}{m_i}.$$

The last inequality is correct based on the Bernoulli's inequality when $t_i \le m_i$. If $b_i < 2^i$, then $t_i < 2^i$ because in each round each corrupt user appears in exactly one column in M, and thus can block at most one bridge. Since in each round $m_i = 2^{i+1} \ge 2t_i$, this probability becomes a constant $\le 1/2$.

Since $\lceil c \log n \rceil$ instances of Algorithm 4.1 run in parallel, the probability that every column that u appears in among all matrices is "bad" (*i.e.*, has at least one active corrupt

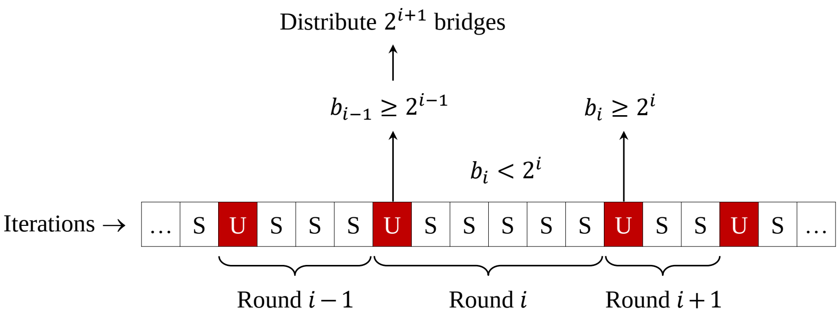


Figure 4.3: Number of bridges distributed in the i-th round of Algorithm 4.1

user) is at most $(1/2)^{c \log n} = 1/n^c$. By applying the union bound, the probability that any of the n users fails to sit in a "good" column (*i.e.*, a column with no active corrupt user) is at most $1/n^{c-1}$. Therefore for any $c > 1$, all honest users can connect to Tor with high probability. □

Lemma 21. *By running Algorithm 4.1, all honest users can connect to Tor with high probability after at most* $\lceil \log t \rceil + 1$ *iterations.*

Proof. Similar to Lemma 18, we let k denote the smallest number of rounds needed until all users can connect to Tor with high probability. k is bounded, because t is bounded. So, for all $i \geq k$, $b_i < 2^i$, and based on Lemma 20, all users can connect to Tor with high probability. We now find k with respect to t.

In each round i, the adversary can maximize k by minimizing the number of bridges blocked (*i.e.*, b_i), while ensuring the algorithm proceeds to the next round. This can be done by blocking only half of the 2^{i+1} bridges distributed in each round and memorizing the rest 2^i bridge addresses to be blocked in future rounds, where $t < 2^i$. Let ℓ be the smallest integer such that $t \leq 2^\ell$. Until round ℓ, the adversary can memorize at most

$$\sum_{j=1}^{\ell-1} 2^j = 2^\ell - 2$$

bridges. In round ℓ, no more bridges can be memorized, because the adversary has to block all of the $t \leq 2^\ell$ bridges it has learned in this round to force the algorithm to proceed to round $\ell + 1$. In round $\ell + 1$, however, the adversary can block at most

$$2^\ell - 2 + 2^\ell = 2^{\ell+1} - 2 < 2^{\ell+1}$$

bridges, which is insufficient for proceeding to round $\ell + 2$. Therefore, $\ell + 1$ is the last round and $k = \ell + 1$. Since $t \leq 2^{\ell}$, $k = \lceil \log t \rceil + 1$. In other words, if the algorithm is run for at least $\lceil \log t \rceil + 1$ iterations, then all honest users succeed with high probability. □

Lemma 22. *The total number of bridges used by Algorithm 4.3 is at most* $(12t - 2)c \log n$*.*

Proof. Similar to Lemma 19, the total number of bridges can be calculated from (4.6). In the i-th round, the algorithm recruits $a_i \leq 2^{i+1}$ new bridges. Since $b_i < 2^i$,

$$N_i < 2^{i+1} + \sum_{j=1}^{i-1} 2^j = 2^{i+1} + 2^i - 2 = 3 \cdot 2^i - 2.$$

From Lemma 21, it is sufficient to run the algorithm for $\lceil \log t \rceil + 1$ rounds. Hence,

$$N_i < 3 \cdot 2^{\lceil \log t \rceil + 1} - 2 < 12t - 2,$$

i.e., the total number of bridges used by the algorithm is at most $(12t - 2)c \log n$. □

In the following lemma, we use martingales to show that the number of bridges used by the algorithm can be reduced by a factor of two when a constant fraction of the users are corrupted, *i.e.*, when $t = \alpha n$, for some constant $\alpha \in [0, 1]$. To achieve this, we let Algorithm 4.1 distribute only $m_i = 2^i$ bridges in each round.

Lemma 23. *Let only a fixed constant fraction of the users be corrupted, and* $m_i = 2^i$ *in each round i of Algorithm 4.1. The algorithm guarantees all honest users can connect to Tor with high probability, and the total number of bridges used is* $(8t - 2)c \log n$*.*

Proof. For $j = 1, 2, ..., m_i$, let $\{X_j\}$ be a sequence of random variables each corresponding to the j-th column of M, where each column consists of $\frac{n}{m_i}$ users chosen uniformly at random without replacement from the set of all users. Also, let Y be a random variable corresponding to the number of columns that have no corrupt users. Since $\mathrm{E}[|Y|] < \infty$, the sequence $\{Z_j = \mathrm{E}[Y | X_1, ..., X_j]\}$ defines a Doob martingale. Since the probability that a given column has no corrupt user is at least $((n - t_i)/n)^{n/m_i}$, by the linearity of expectation,

$$\mathrm{E}[Y] = m_i \cdot \left(1 - \frac{t_i}{n}\right)^{n/m_i}.$$

Since in each round $m_i = 2^i \geq t_i$ and $t = \alpha n$, for some constant $\alpha \in [0, 1]$, we have $\mathrm{E}[Y] = m_i(1 - \alpha)^{1/\alpha}$. This means that, in expectation, a constant fraction of the columns in each matrix (in each round) are good. Since choosing one random column changes the expected number of good columns by at most one, $|Z_{j+1} - Z_j| \leq 1$. Using the Azuma's inequality, we get

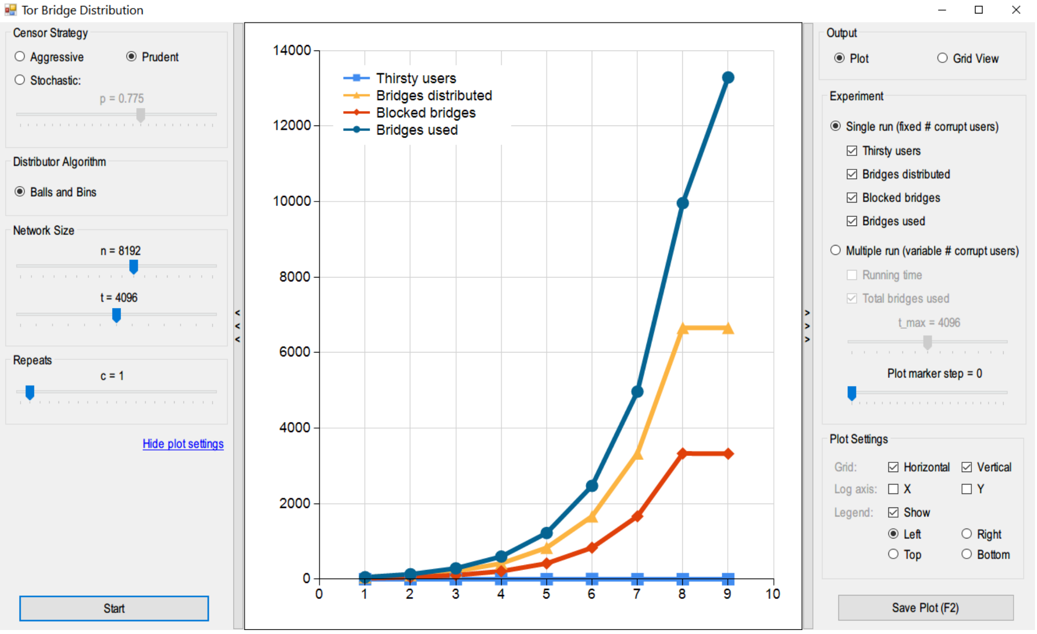
$$\Pr\left(|Y - \mathrm{E}[Y]|\right) \geq \lambda) \leq 2e^{-2\lambda^2/m_i},$$

for any $\lambda > 0$. Therefore, the actual values of Y are highly concentrated around its expected value $\mathrm{E}[Y]$.

The number of bridges distributed in each round of the modified algorithm is $m_i = 2^i$. Thus, using equation (4.6) in Lemma 19, the total number of bridges used by the modified algorithm is at most $(8t - 2)c \log n$. □

4.4 Simulation Results

To evaluate various properties of our algorithms, we implemented a proof-of-concept prototype and tested it in a simulated environment under various adversarial behavior. The prototype is written in C# using .NET Framework 4.5. We ran the simulations on an Intel Core i5-4250U 1.3GHz machine with 4GB of RAM running Windows 10 Pro. We set the parameters of our protocols in such a way that we ensure the failure probability of the distribution algorithm is smaller than 10^{-5}.



Tor Bridge Distribution

Censor Strategy
○ Aggressive ● Prudent
○ Stochastic:
p = 0.775

Distributor Algorithm
● Balls and Bins

Network Size
n = 8192
t = 4096

Repeats
c = 1

Output
● Plot ○ Grid View

Experiment
● Single run (fixed # corrupt users)
☑ Thirsty users
☑ Bridges distributed
☑ Blocked bridges
☑ Bridges used
○ Multiple run (variable # corrupt users)
□ Running time
☑ Total bridges used
t_max = 4096
Plot marker step = 0

Plot Settings
Grid: ☑ Horizontal ☑ Vertical
Log axis: □ X □ Y
Legend: ☑ Show
● Left ○ Right
○ Top ○ Bottom

Figure 4.4: Bridge Distribution Simulation Application

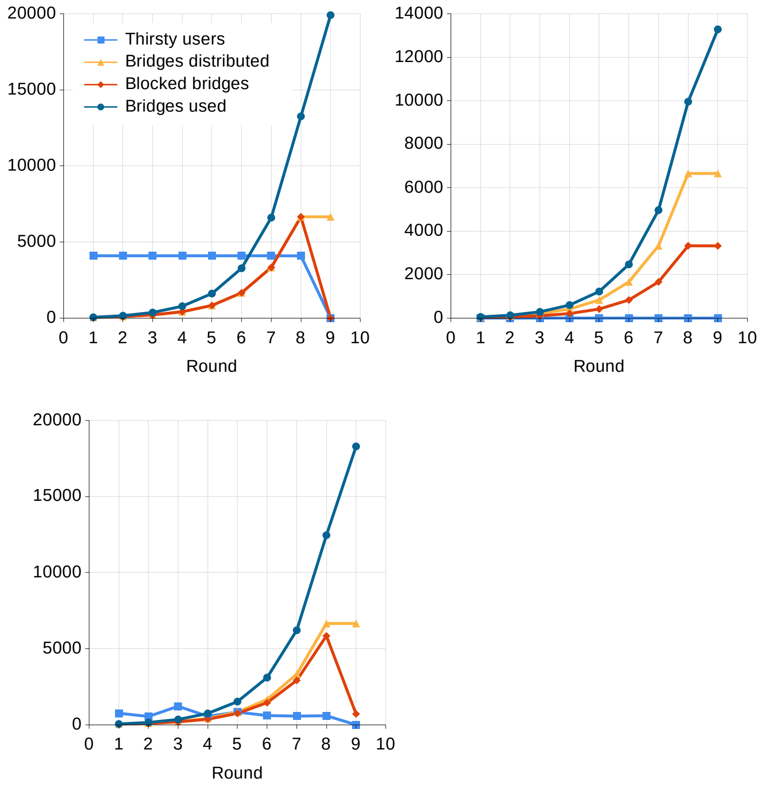


Figure 4.5: Simulation results for $n = 8192$ and $t = 4096$ with aggressive (left) prudent (middle), and stochastic (right) adversary.

4.5 Conclusion

We described bridge distribution algorithms that allow all (honest) users to connect to Tor in the presence of an adversary corrupting an unknown number of users. Our algorithms can adaptively increase the number of bridges according to the behavior of the adversary and use near-optimal number bridges.

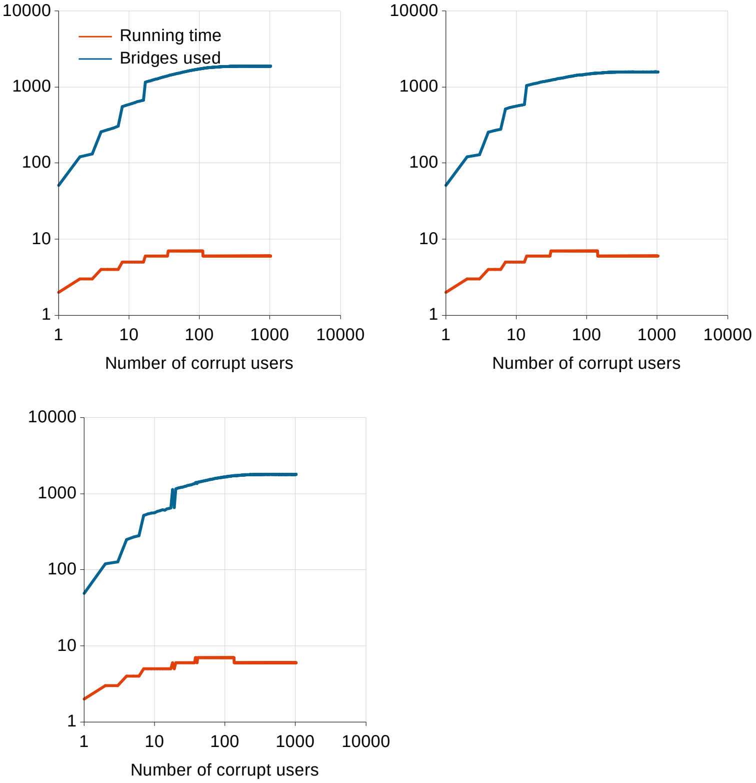


Figure 4.6: Simulation results for $n = 1024$ and various number of corrupt users with aggressive (left) prudent (middle), and stochastic (right) adversary.

Chapter 5

Conclusion and Open Problems

With the increasing level of sophisticated cyber-attacks that target user privacy and operation of information systems, we require new tools and technologies that enable privacy-preserving communications and computations over user data and that tolerate arbitrary failures of system components.

We described efficient algorithms for tracking-resistant anonymous communication in both the Internet and peer-to-peer networks. Moreover, unlike the majority of previous work which rely on centralized trusted servers, most of our algorithms are fully-decentralized and do not require any trusted party.

In the rest of this chapter, we propose a few open problems related to the research done in this dissertation. Most of these open problems are related to possible improvements to our current MPC and ABC protocols in order to achieve practicality. We also initiate a new direction in solving MPC using oblivious RAM model which is a relatively new area with a radical but intuitive approach to secure computation.

5.1 Practical MPC

The following are some promising techniques for improving the efficiency of our MPC protocol.

- *Blacklisting malicious parties.* One technique for increasing efficiency of our protocols is to detect and evict parties that exhibit adversarial behavior over time. Once all malicious parties are blacklisted, the protocol can go into a light-weight mode, where the expensive verification steps are avoided. If new adversarial faults happen while in the lightweight mode, the protocol should be able to maintain privacy, restart the computation, and blacklist new malicious parties.

- *Covert adversarial model.* Addressing an adversarial model like *covert* that is weaker than unconditional but stronger than cryptographic models. Cryptographic techniques are often communication and round-efficient but are computationally expensive, require large key sizes, and sometimes hard to implement. Unconditional methods are often simple to implement but hard to design. They also provide an excessive level of security that is usually unnecessary in practice and increases communication costs. A model in between seems to provide most of these benefits at a small cost.

- *Improving the resiliency bound.* Increasing the fraction of dishonest parties that our algorithms can tolerate. This probably requires new share renewal and secure multiplication techniques. A simple technique seems possible using the multiplication protocol of [GRR98].

5.1.1 MPC in the Oblivious RAM Model

The typical approach for implementing an algorithm for MPC is to rewrite the algorithm as an arithmetic circuit or a Boolean circuit and then, execute each gate of the circuit using secure addition or multiplication operations. Cramer et al. [CFIK03] argue that one can represent any formulas and branching program by low-degree randomizing polynomials over some finite ring, which can be evaluated in constant rounds. However, these approaches have their own limitations, e.g., it is not easy to construct the optimal branching program for complex functions. On the other hand, Damgard et al. [DFK+06] show how to perform equality-check, comparison, bit-decomposition, and unbounded fan-in symmetric functions in constant-round over finite fields. Nevertheless, it is not obvious how to efficiently perform oblivious evaluation of functions such as sorting and shuffling in constant rounds by using these elementary protocols.

Most algorithms have already been described in terms of instructions (programs) to a RAM machine. Such programs are usually called *RAM programs.* A RAM machine has a *lookup functionality* for accessing memory locations that takes $O(1)$ operations. Given an array A of N values and an index $x \in \{1, \dots, N\}$, the lookup functionality returns $A[x]$. Goldreich and Ostrovsky [GO96] propose a new technique called *Oblivious RAM (ORAM)* that allows a client to hide its access pattern to a remote storage by continuously shuffling and re-encrypting data as they are accessed. ORAM techniques enjoy simplicity and practical efficiency.

Hiding the access pattern, or making it *oblivious*, means that any equal-length sequence of clients' data requests to the server are equivalent from the point of view of the eavesdropper who might be the server itself. The server must only know the number of queries in the sequence. Achieving this property implies in particular that the following information would be unknown to the server: (1) the locations of accessed data items, (2) the order of data requests, and (3) the number of requests to the same location. In addition, different types of access (e.g., get-value, set-value, insert-new-value) must also be indistinguishable.

We envision a programming model for MPC based on the ORAM model. This separates the program into two parts: *control flow* and *memory*. Memory can be abstracted by either a trusted server or by the well-known and proven *shared memory paradigm*. Control flow can be parallelized in order to reduce computational time and latency. We propose a technical approach that relies on the use of quorums and homomorphic secret sharing as described in Section 2. Information is secret-shared among parties inside a quorum and the control flow and access to variables are handled securely by the quorums. This is performed in a way that ensures no single party ever can access or learn the value of any variable. Instead, the variables are manipulated via quorums that act as single functional units. The state of a variable is manipulated via homomorphic operations applied to these shares.

5.2 Practical Anonymous Broadcast

In this dissertation, we described an anonymous broadcast protocol that securely shuffles input messages by evaluating a sorting network using MPC. In this section, we provide two promising approaches for improving efficiency of this protocol. In both approaches, wee seek better techniques for multi-party shuffling of a set of messages by replacing the sorting network with a possibly smaller-depth and simpler circuit.

5.2.1 Non-Comparison-Based Sorting

One major drawback of our current ABC protocol is its round complexity. One promising way of improving this is to design a sorting circuit from known non-comparison-based sorting algorithms such as bucket sort and radix sort. This seems possible because our multi-party shuffling protocol sorts uniform random numbers. Since the $O(n \log n)$ lower bound on the time complexity of sorting algorithms does not hold for non-comparison-based sorting algorithms, we conjecture that it might be possible to build a sorting circuit without using comparators to achieve a better than $O(\log n)$-depth sorting circuit. For example, Is it possible to use known oblivious sorting algorithms such as [HICT14, HKI+12, Goo11] to improve round complexity?

5.2.2 Shuffling via Expander Graphs

One approach for solving the secure shuffling problem is to evaluate a permutation network using MPC. In this section, we first describe permutation networks and their limitations for the problem of anonymous broadcast. Then, we conjecture that expander graphs can be used for constructing permutation networks that meet our anonymity goals.

Permutation Networks. *A permutation network* is a network of *swappers* (*a.k.a.*, *switches*). Each swapper is a gate with two input wires (variables) x_1, x_2, a switch wire r, and two output wires y_1, y_2. While x_1 and x_2 can be any value, r is either zero or nonzero. Each swapper behaves in the following way. If $r = 0$, then $y_1 = x_1, y_2 = x_2$, otherwise, $y_1 = x_2, y_2 = x_1$. A permutation network may be used for generating different permutations of n values by setting the switches arbitrarily. Typically, a permutation network has $\lceil n/2 \rceil$ input swappers, $\lceil n/2 \rceil$ output swappers, and several intermediate swappers all connected to each other via their wires in some way.

By setting the switches uniformly at random, one obtains a certain probability distribution that is often desired to be *close* to the uniform distribution. A network consisting entirely of switches with swapping probability of $1/2$ cannot generate a uniform distribution because for a circuit with m such gates, there are 2^m possible outcomes. Since $m! \nmid 2^m$, some permutations are more likely to be generated by the network than others (see [Knu98] Section 5.3.4 for a detailed discussion). Typically, the effectiveness of a permutation network is measured by the variation distance between the uniform distribution (denoted by u) and the probability distribution generated by the network (denoted by v) on the sample space Ω of all possible permutations, defined as

$$d(u, v) = 1/2 \sum_{\omega \in \Omega} |u(\omega) - v(\omega)|$$

Czumaj et al. [CKLK01] propose a permutation network for generating almost-uniform permutations with distance $O(1/n^2)$ from uniform distribution. To the best of our knowledge, this network has the smallest distance among other known permutation networks such as [Wak68].

Anonymity via Permutation Networks. One approach for secure multi-party shuffling is to securely evaluate a permutation network using MPC. We believe such a circuit not only can have a smaller depth but also it can result in a higher bandwidth efficiency of our protocol because we believe secure swapping is significantly less expensive than secure comparison in terms of both communication and computation costs. For example, our microbenchmarks show that a secure comparison (using the technique of [NO07]) requires sending roughly 200 times more bits than a secure swapping in a network with one million parties.

In the context of anonymous broadcast, however, we believe the variation distance (defined above) is not necessarily an effective measure of anonymity due to the following informal argument. In anonymity, we are interested to keep the worst-case deviations as small as possible. In other words, we want almost all permutations to happen with a probability close to uniform. Unfortunately, the variation distance does not indicate whether only a few permutations (out of the sample space) have large deviations from the uniform probability or most permutations have small deviations from the uniform probability. Thus, for anonymity, we propose to measure the weakness of a random permutation generator by the *maximum deviation* from the uniform distribution on the sample space Ω of all possible permutations, i.e.,

$$w(u, v) = \max_{\omega \in \Omega} |u(\omega) - v(\omega)|$$

Permutation Networks using Exapnders. Informally, an *expander* is a graph in which the neighborhood of any set of vertices S is large relative to the size of S. This means that every subset of vertices expands rapidly. Ideally, the graph should have few (linear) edges, and in fact be of bounded degree, i.e., a good expander has low degree and high expansion parameters. Due to their rapid mixing properties, expanders have been extensively used in many applications such as pseudorandom generators and sorting networks. Although they have been used for designing sorting networks [AKS83b, Pat90], we are not aware a permutation network based on expanders. We are specifically interested to see if using random walks on an expander we can build a permutation network with a small maximum deviation (as defined above).

5.3 Other Privacy-Preserving Applications

We are interested in studying applications that can be efficiently solved using our MPC and ABC algorithms. One such application is a privacy-preserving location-based service described in Section 3.5. As mentioned earlier, we envision that the problem of privacy-preserving LBS can be efficiently modeled as an MPC problem.

User data collected by companies and government agencies are constantly mined and analyzed. Such data sets are usually very large that drive organizations to outsource their analytics activities to the cloud or large multi-party networks. Unfortunately, this lays out several privacy challenges because untrusted parties can abuse the data and extract sensitive private information from customers. Sorting is one of the most important and well-studied primitives in data analysis. In a multi-party setting with untrusted parties, however, Multi-Party Sorting (MPS) is a new and challenging problem with applications in several problems including privacy-preserving data mining, private information retrieval, and private database operations [JKU11, Zha11]. Unfortunately, known solutions either scale poorly with the number of parties [HKI$^+$12] or have impractical costs [BGT13, DKMS14]. Using our MPC protocol, we can build an efficient MPS algorithm for privacy-preserving data mining of massive data sets such as large matrix factorization.

Bibliography

[Abr74] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974.

[AJLA$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer Berlin Heidelberg, 2012.

[AKS83a] 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.

[AKS83b] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, January 1983.

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, April 2010.

[AL11] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. Cryptology ePrint Archive, Report 2011/136, 2011.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*, page 14. John Wiley Interscience, March 2004.

[AW07] Ben Adida and Douglas Wikström. How to shuffle in public. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, pages 555–574, Berlin, Heidelberg, 2007. Springer-Verlag.

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer Berlin Heidelberg, 2004.

[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.

[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.

[BFM$^+$15] Michael A. Bender, Jeremy T. Fineman, Mahnush Movahedi, Jared Saia, Varsha Dani, Seth Gilbert, Seth Pettie, and Maxwell Young. Resource-competitive algorithms. *ACM SIGACT News*, 46(3):57–71, September 2015.

[BFT04] Ron Berman, Amos Fiat, and Amnon Ta-Shma. Provable unlinkability against traffic analysis. In *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2004.

[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.

[BLPW08] Bhuvan Bamba, Ling Liu, Peter Pesti, and Ting Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 237–246, New York, NY, USA, 2008. ACM.

[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.

[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 Instittute 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.

[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 multiparty computation. Technical report, Cambridge, MA, USA, 1996.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multiparty computation over rings. In *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer Berlin Heidelberg, 2003.

[CGF10] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 340–350, New York, NY, USA, 2010. ACM.

[CGWF13] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Proceedings of the 22nd USENIX Security Symposium*, pages 147–162, Berkeley, CA, USA, 2013.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[CKLK01] Artur Czumaj, Przemka Kanarek, Krzysztof Lorys, and Miroslaw Kutylowski. Switching networks for generating random permutations, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 494–503, New York, NY, USA, 2002. ACM.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, pages 42–51, 1993.

[CWF12] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Dining in the Sunshine: Verifiable Anonymous Communication with Verdict. *ArXiv e-prints*, September 2012.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, JesperBuus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer Berlin Heidelberg, 2006.

[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.

[Din11a] Roger Dingledine. Research problem: Five ways to test bridge reachability. Available at: `https://goo.gl/BTJuZP`, 2011.

[Din11b] Roger Dingledine. Research problems: Ten ways to discover Tor bridges. Available at: `https://goo.gl/CYFfC4`, 2011.

[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.

[DM06] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. Technical report, The Tor Project Inc., 2006.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, Berkeley, CA, USA, 2004.

[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.

[DO00] Shlomi Dolev and Rafail Ostrovsky. Xor-trees for efficient anonymous multicast and reception. *ACM Trans. Inf. Syst. Secur.*, 3(2):63–84, May 2000.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 2009.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[EFW$^+$15] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining how the Great Firewall discovers hidden circumvention servers. In *Internet Measurement Conference (IMC)*. ACM, 2015.

[EKAC14] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R. Crandall. Detecting intentional packet drops on the Internet via TCP/IP side channels. In *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362*, PAM 2014, pages 109–118, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[FA00] Stajano Frank and Ross Anderson. The cocaine auction protocol: On the power of anonymous broadcast. In *Proceedings of the Third International Workshop on Information Hiding*, IH 99, pages 434–447, London, UK, 2000. Springer-Verlag.

[Far88] R.W. Farebrother. *Linear Least Squares Computations*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1988.

[FBW$^+$03] Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger. Thwarting web censorship with untrusted messenger discovery. In Roger Dingledine, editor, *Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer Science*, pages 125–140. Springer Berlin Heidelberg, 2003.

[FF14] Joan Feigenbaum and Bryan Ford. Seeking anonymity in an Internet panopticon. *e-Print arXiv:1312.5307*, March 2014.

[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.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 193–206, New York, NY, USA, 2002. 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.

[GG03] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 31–42, New York, NY, USA, 2003. ACM.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012.

[GHY88] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-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.

[GJ04] Philippe Golle and Ari Juels. Dining cryptographers revisited. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473. Springer Berlin Heidelberg, 2004.

[GKS07] Gabriel Ghinita, Panos Kalnis, and Spiros Skiadopoulos. Prive: Anonymous location-based queries in distributed mobile systems. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 371–380, New York, NY, USA, 2007. ACM.

[GL05] B. Gedik and Ling Liu. Location privacy in mobile systems: A personalized anonymization model. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 620–629, June 2005.

[GMOT12] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Practical oblivious storage. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, pages 13–24, New York, NY, USA, 2012. ACM.

[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.

[Goo11] Michael T. Goodrich. Randomized shellsort: A simple data-oblivious sorting algorithm. *J. ACM*, 58(6):27:1–27:26, December 2011.

[Gro04] Jens Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 90–104. Springer Berlin Heidelberg, 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.

[GS13] Barton Gellman and Ashkan Soltani. NSA tracking cellphone locations worldwide, snowden documents show. The Washington Post, December 2013.

[GSKY12] Seth Gilbert, Jared Saia, Valerie King, and Maxwell Young. Resource-competitive analysis: A new perspective on attack-resistant distributed computing. In *Proceedings of the 8th International Workshop on Foundations of Mobile Computing*, FOMC '12, pages 1:1–1:6, New York, NY, USA, 2012. ACM.

[HICT14] Koki Hamada, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. Cryptology ePrint Archive, Report 2014/121, 2014.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology – CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer Berlin Heidelberg, 1995.

[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.

[JKU11] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. Cryptology ePrint Archive, Report 2011/122, 2011.

[KH03] Jiejun Kong and Xiaoyan Hong. Anodr: Anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, pages 291–302, New York, NY, USA, 2003. ACM.

[KKK08] Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of VSS in point-to-point networks. In *Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 499–510. Springer Berlin Heidelberg, 2008.

[KKSZ13] Josh Karlin, Joud Khoury, Jared Saia, and Mahdi Zamani. Brief announcement: Scalable anonymous communication with Byzantine adversary. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 128–130, New York, NY, USA, 2013. ACM.

[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.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[LBCZ$^+$13] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 303–314, New York, NY, USA, 2013. ACM.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. *J. ACM*, 53(6):881–917, November 2006.

[LP90] Tom Leighton and C. Greg Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, FOCS ’90, pages 264–274, Oct 1990.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer Berlin Heidelberg, 2010.

[LWZ11] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In *Information Security*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer Berlin Heidelberg, 2011.

[Mah10] Mohammad Mahdian. Fighting censorship with algorithms. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms*, volume 6099 of *Lecture Notes in Computer Science*, pages 296–306. Springer Berlin Heidelberg, 2010.

[MCA06] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. The new casper: Query processing for location services without compromising privacy. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB ’06, pages 763–774. VLDB Endowment, 2006.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *ACM Trans. on Knowledge Discovery from Data*, 1(1), March 2007.

[MML12] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. Proximax: Measurement-driven proxy dissemination. In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security*, FC’11, pages 260–267, Berlin, Heidelberg, 2012. Springer-Verlag.

[MS81] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and Reed-Solomon codes. *Commun. ACM*, 24(9):583–584, September 1981.

[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, 2005.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

[NO07] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360. Springer Berlin Heidelberg, 2007.

[Oni12] The Open Net Initiative: China. Available at: https://goo.gl/YLb37p, 2012.

[Pat90] M.S. Paterson. Improved sorting networks witho(logn) depth. *Algorithmica*, 5(1-4):75–92, 1990.

[PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity – a proposal for terminology. In *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2001.

[PW86] Andreas Pfitzmann and Michael Waidner. Networks without user observability – design options. In *Advances in Cryptology – EUROCRYPT '85*, volume 219 of *Lecture Notes in Computer Science*, pages 245–253. Springer Berlin Heidelberg, 1986.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, November 1998.

[RS93] Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 672–681, New York, NY, USA, 1993. ACM.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SLL08] Yair Sovran, Alana Libonati, and Jinyang Li. Pass it on: Social networks stymie censors. In *Proceedings of the 7th International Conference on Peer-to-peer Systems*, IPTPS'08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

[SV11] N.P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011.

[Tor15a] The Tor Project metrics: Direct users connecting between January 1, 2015 and March 31, 2015. Available at: https://goo.gl/mz1vLS, 2015.

[Tor15b] The Tor Project: Pluggable transport. Available at: https://goo.gl/SBGupD, 2015.

[vABH03] Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. k-anonymous message transmission. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 122–130, New York, NY, USA, 2003. ACM.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag.

[Wak68] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, January 1968.

[WDA$^+$15] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings* *of the 22nd ACM Conference on Computer and Communications Security*, CCS '15, 2015.

[WL12] Philipp Winter and Stefan Lindskog. How the great firewall of China is blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet*, Berkeley, CA, 2012.

[WLBH13] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas Hopper. rbridge: User reputation based tor bridge distribution with privacy preservation. In *Network and Distributed System Security Symposium*, NDSS 2013. The Internet Society, 2013.

[WP90] Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, page 690. Springer Berlin Heidelberg, 1990.

[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.

[YF05] Gang Yao and Dengguo Feng. A new k-anonymous message transmission protocol. In *Information Security Applications*, volume 3325 of *Lecture Notes in Computer Science*, pages 388–399. Springer Berlin Heidelberg, 2005.

[YKGK10] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical robust communication in DHTs tolerating a Byzantine adversary. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 263–272, Washington, DC, USA, 2010. IEEE Computer Society.

[ZGH07] Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *Proceedings of the 7th International Conference on Privacy Enhancing Technologies*, PET'07, pages 62–76, Berlin, Heidelberg, 2007. Springer-Verlag.

[Zha11] Bingsheng Zhang. Generic constant-round oblivious sorting algorithm for MPC. In *Provable Security*, volume 6980 of *Lecture Notes in Computer Science*, pages 240–256. Springer Berlin Heidelberg, 2011.

[ZMS14] Mahdi Zamani, Mahnush Movahedi, and Jared Saia. Millions of millionaires: Multiparty computation in large networks. Cryptology ePrint Archive, Report 2014/149, 2014.

[ZSMK13] Mahdi Zamani, Jared Saia, Mahnush Movahedi, and Joud Khoury. Towards provably-secure scalable anonymous broadcast. In *the 3rd USENIX Workshop on Free and Open Communications on the Internet*, FOCI '13, 2013.