Secure Multi-Party Shuffling[⋆]

Mahnush Movahedi, Jared Saia, and Mahdi Zamani

Department of Computer Science, University of New Mexico
{movahedi, saia, zamani}@cs.unm.edu

Abstract. In secure multi-party shuffling, multiple parties, each holding an input, want to agree on a random permutation of their inputs while keeping the permutation secret. This problem is important as a primitive in many privacy-preserving applications such as anonymous communication, location-based services, and electronic voting. Known techniques for solving this problem suffer from poor scalability, load-balancing issues, trusted party assumptions, and/or weak security guarantees.
In this paper, we propose an unconditionally-secure protocol for multi-party shuffling that scales well with the number of parties and is load-balanced. In particular, we require each party to send only a polylogarithmic number of bits and perform a polylogarithmic number of operations while incurring only a logarithmic round complexity. We show security under universal composability against up to about $n/3$ fully-malicious parties. 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.

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 paper, 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.

[⋆] This is the extended version of the paper published in the proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Montserrat, Spain, 2015.

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 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 $\pi : \{1, ..., n\} \to \{1, ..., n\}$ 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.*

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

In this paper, 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*.*

1.1 Our Results

We prove the following main theorem in Section 4.

Theorem 1. *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, we also construct a computationally-secure variant of Theorem 1 to observe (via simulations) how much cryptographic techniques can influence practical efficiency of our protocol. This protocol provides the same guarantees as Theorem 1 except for a polynomially time-bounded adversary. We provide our simulation results in Section 5.

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].

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 commu-

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

[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 1. Comparison of MPS techniques

Protocol	**Adversarial Power**	**Malicious Adversary?**	**Fraction of Parties Controlled**	**Fraction of Links Monitored**	**MPS Security**	**Latency**	**Bandwidth**	**Easy to Implement?**
Chaum [**Cha81**]	Computational	No	$O(1)^\dagger$	See note‡	See note§	polylog(n)	polylog(n)	Yes
Rackoff and Simon [**RS93**]	Computational	No	$O(1)^\dagger$	All	Statistical$^\pounds$	polylog(n)	$\tilde{O}(n)$	No
Berman ***et al.*** [**BFT04**]	Computational	No	$O(1)^\dagger$	$O(1)$	Statistical$^\pounds$	polylog(n)	polylog(n)	Yes
Boyle ***et al.*** [**BGT13**]	Computational	Yes	$1/3-\epsilon$	All	Almost	polylog(n)	$\tilde{O}(n)$	No
Dani ***et al.*** [**DKMS12**]	Unconditional	Yes	$1/3-\epsilon$	All	Almost	$O(\log n)$	$\tilde{O}(\sqrt{n})$	No
This paper	Unconditional	Yes	$1/3-\epsilon$	All	Almost	$O(\log n)$	$\tilde{O}(1)$	Yes

†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.
$^\pounds$Measures the statistical distance between the distribution generated by the system and the uniform distribution [RS93].

nication 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 1, we compare our main protocol with several other ones that can be used to 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. In this table, by bandwidth we mean the communication complexity per shuffled message delivered.

2 Preliminaries

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

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 2 ([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 3 ([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]

[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.

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 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 2 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 4 ([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 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 1, 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 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 $\mathcal{C}$ denote the probabilistic sorting network of [LP90] and $m = \Theta(n \log n)$ be the number of gates in $\mathcal{C}$.

For all $j \in [m]$, we assign the j-th gate of $\mathcal{C}$ 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 $\mathcal{C}$ are $Q_1, ..., Q_{\lceil n/2 \rceil}$.[6] When used to receive inputs of $\mathcal{M}$, we refer to

[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.

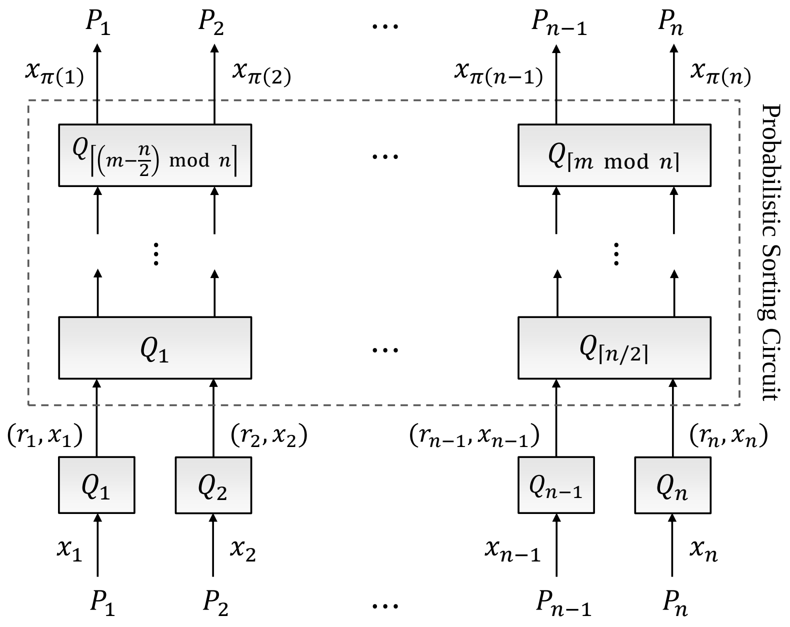

Fig. 1. MPS circuit

$Q_1, ..., Q_n$ 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 $\mathcal{C}$ 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 $\mathcal{C}$ using the random 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 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 1: Renew-Shares and Ran-Gen. In Section 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 3.2, 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 1 (and Theorem 1) in Section 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.

Protocol 1 Secure Multi-Party Shuffling Scheme

Inputs. For all $i \in [n]$, party P_i holds an input x_i. Let $\mathcal{C}$ 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 $\mathcal{C}$, 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 $q > \frac{3}{2} k n^2 \log n$ 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.** $\mathcal{C}$ is evaluated level-by-level starting from the input gates. For each gate G in $\mathcal{C}$ 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 $\rho = (r \le r')$ be the resulting secret-shared value. The parties compute the output secret-shared pairs (s, y) and (s', y') from
   $$s = \rho \cdot r + (1 - \rho) \cdot r', \qquad y = \rho \cdot x + (1 - \rho) \cdot x'$$
   $$s' = \rho \cdot r' + (1 - \rho) \cdot r, \qquad y' = \rho \cdot x' + (1 - \rho) \cdot x$$
   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.

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 2.

[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.

Protocol 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 $g(j) = \sum_{i=1}^{N} \lambda_i h_i(j)$.

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 process is done robustly. We refer to this protocol as Reshare. In Section 4, we prove Renew-Shares is UC-secure against at most $T < N/3$ corrupted parties in each quorum.

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 describes the protocol.

Protocol 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 $r_j = \sum_{k=1}^{N} \rho_{kj}$.

3.3 Remarks

In the following, we discuss alternative approaches that could be used to design different MPS protocols from Protocol 1.

All-Output MPS. Protocol 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 1 becomes as follows. For all $i \in [n-1]$, parties in the output quorum $Q_{\lceil i/2 \rceil}$ run $\mathsf{VSS\text{-}Reconst}$ 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 1), 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 Cryptographic Variant of Protocol 1

We now describe a computationally-secure variant of Protocol 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 5 ([KZG10]). *There exists a constant-round linear VSS scheme for $t < n/2$ secure against a computationally-bounded adversary.*

Theorem 6 ([GRR98]). *There exists a constant-round multiplication protocol secure against a computationally-bounded malicious adversary corrupting up to $\frac{n-1}{2}$ parties.*

Theorem 7. *By plugging the VSS of Theorem 5 and the multiplication protocol of Theorem 6 into Protocol 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 $O(\log n)$.*

[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.

In Section 5, we empirically compare this cryptographic variant with Protocol 1.

4 Security Proofs

The error probability in Theorem 1 comes entirely from the following steps of Protocol 1 failing to output correct results with some probability:

- *Setup:* Protocol Build-Quorums may fail to create good quorums. Theorem 3 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 2 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 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 1, 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 1 achieves an almost t-secure MPS. We prove this lemma in Section 4.

Lemma 1. *Protocol 1 implements an almost t-secure MPS.*

In Lemma 2, 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 4.

Lemma 2. *For some prime q, let $\mathbb{F}_q$ be the field of random elements generated in the Random Generation step of Protocol 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 1 in the universal composability framework. To this end, we define a hybrid model based on the modular composition theorem [Can01], and 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 4.1.

Lemma 3. *Up to the output delivery step, Protocol 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.*

[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.

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 1. 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 1.

Proof. Let $X = (x_1, ..., x_n)$ be the input sequence and $Y = (y_1, ..., y_n)$ be the output sequence generated by Protocol 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 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 $\mathcal{C}$. This sequence contains the elements of X augmented by the random values $r_1, ..., r_n$ generated in the Random Generation step of Protocol 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 $\mathcal{C}$ 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 $\mathcal{C}$ are values chosen uniformly and independently at random from $\mathbb{F}_q$, the set of permutations that each can map the inputs of $\mathcal{C}$ 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 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 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 $\mathcal{C}$. Thus, the adversary rules out $|R_2| = \sigma|S| = \sigma n!$ permutations from S.

In Lemma 4, we show that, in each run, Protocol 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 4 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 statically 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 3, we prove that, other than t input values and their order in the output sequence, Protocol 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,

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

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 4, we show that the Random Generation step of Protocol 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

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

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 1 is almost t-secure.

□

Lemma 4. *The Random Generation step of Protocol 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 6, 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 2. Based on the security of Gen-Rand shown in Lemma 6, all elements generated by the input quorums in the random generation step of Protocol 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 1, we need to show that, up to the output delivery step, Protocol 1 does not reveal any information about the inputs to any party. We prove this in Lemma 3 using the universal composability framework [Can01] briefly reviewed in Section 4.1.

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.

4.2 Proof Sketch

Before proceeding to the proof, we remark that the error probability in Theorem 1 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 8, 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.

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

Theorem 8 ([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 1. Following the modular composition theorem, we first define the ideal functionalities shown in Table 2 that correspond to the subprotocols used in Protocol 1. We then prove that Protocol 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 8 to infer the UC-security of Protocol 1.

In order to prove the t-security of Protocol 1 in the hybrid model, we first show that all of our subprotocols are UC-secure. Similar to the above approach, we first prove t-security of every subprotocol in its corresponding hybrid model using a straight-line black-box simulator and then use Theorem 8 to infer its UC-security. To prove the t-security of a protocol Π, we describe a

Table 2. Ideal functionalities

Functionality	**Implemented by**	**Refer to**
F_{BA}	Protocol BA	Theorem 4
$F_{\mathsf{VSS\text{-}Share}}$	Protocol VSS-Share	Theorem 2
$F_{\mathsf{VSS\text{-}Reconst}}$	Protocol VSS-Reconst	Theorem 2
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 2
$F_{\mathsf{Gen\text{-}Rand}}$	Protocol Gen-Rand	Protocol 3

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.[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 5. *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 8, 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 4. At least $N - T$ of the inputs $\rho_1, ..., \rho_N$ 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 $\mathbb{F}_q$.

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

Protocol 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.

Lemma 6. *The protocol* Gen-Rand *is UC-secure.*

Proof. We prove the t-security of Gen-Rand in the $F_{\textsf{VSS-Share}}$-hybrid model which is similar to Protocol 3 except that every call to VSS-Share is replaced with a call to the ideal functionality $F_{\textsf{VSS-Share}}$. The corresponding simulator $\mathcal{S}_{\textsf{Gen-Rand}}$ is given in Protocol 5.

Protocol 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 $s_1^i, ..., s_N^i$ 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 views of the corrupted parties in the hybrid execution and the simulation are indistinguishable because the only difference between the two views is that $\mathcal{S}_{\textsf{Gen-Rand}}$ generates 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. Following Theorem 8, since our simulator is straight-line and black-box, Gen-Rand is UC-secure. □

Lemma 7. *The protocol* Renew-Shares *is UC-secure.*

Proof. The corresponding ideal functionality $F_{\textsf{Renew-Shares}}$ is shown in Protocol 6. In this functionality, we denote the ideal functionality of Reshare by $F_{\textsf{Reshare}}$ 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 $F_{\textsf{Renew-Shares}}$ 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.

Protocol 6 $F_{\mathsf{Renew\text{-}Shares}}$

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_{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.

We prove in the $(F_{\mathsf{VSS\text{-}Share}}, F_{\mathsf{Reshare}})$-hybrid model which is similar to Protocol 2 except that every call to $\mathsf{VSS\text{-}Share}$ and $\mathsf{Reshare}$ are replaced with calls to the ideal functionalities $F_{\mathsf{VSS\text{-}Share}}$ and F_{Reshare} respectively. The corresponding simulator $\mathcal{S}_{\mathsf{Renew\text{-}Shares}}$ is given in Protocol 7.

Protocol 7 $\mathcal{S}_{\mathsf{Renew\text{-}Shares}}$

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_{Reshare} with dummy input 0. Let $s^i_1, ..., s^i_N$ denote the outputs.
2. For each $j \in I$,
   (a) Run F_{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.

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 3.

Proof. We prove the security of Protocol 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 5, Lemma 6, and Lemma 7. The last step of the proof is to show that Protocol 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 8, since we have proved the t-security of Protocol 1 using a straight-line black-box simulator, the protocol is UC-secure. □

4.3 Cost Analysis

We first compute the cost of each step of the protocol separately and then compute the total costs. Let $\nu_1(n)$ and $\nu_2(n)$ denote the communication and computation complexity of VSS-Share respectively when it is invoked among n parties. For our unconditional Protocol 1, we assume VSS-Share implements the sharing protocol of Theorem 2, and for our cryptographic MPS, we assume VSS-Share implements the sharing protocol of VSS of Theorem 5. 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 3 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 $O(n \cdot \nu_1(N))$ bits and performs $O(n \cdot \nu_2(N))$ operations. Since the VSS scheme is constant-round, this step also takes constant rounds of communication.
- *Random Generation.* Gen-Rand sends $O(N \cdot \nu_1(N))$ messages, performs $O(N \cdot \nu_2(N))$, 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 $O(N^4 \cdot \nu_1(N))$ 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 5, 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 7. For the costs of Theorem 1 (*i.e.*, our unconditional result), since $\nu_1(N) = \nu_2(N) = O(\mathsf{poly}(N))$, Protocol 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.

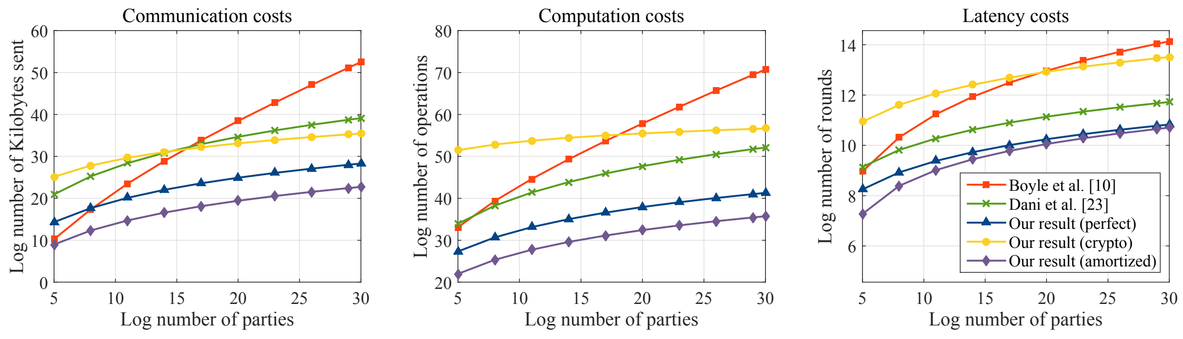

Fig. 2. Communication cost (left), computation cost (middle), and the number of communication rounds (right)

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) 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. 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 2 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 2, we report results from three different versions of our protocols. The first plot (marked with triangles) refers to our unconditionally-secure protocol (Protocol 1). The second plot (marked with circles) represents the cryptographic variant of Protocol 1 described in Section 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 1 once

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

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

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 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.

6 Conclusion

We 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.

Acknowledgment

The authors would like to acknowledge supports from NSF under grants CCF-1320994, CCR-0313160, and CAREER Award 644058. We are also grateful for valuable comments from Ran Canetti (Boston University), Bryan Ford (Yale University), Shafi Goldwasser (MIT), Aniket Kate (Purdue University), Yehuda Lindell (Bar-Ilan University), and anonymous reviewers of this paper.

References

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.