Millions of Millionaires: Multiparty Computation in Large Networks

Mahdi Zamani
University of New Mexico
zamani@cs.unm.edu

Mahnush Movahedi
University of New Mexico
movahedi@cs.unm.edu

Jared Saia
University of New Mexico
saia@cs.unm.edu

Abstract

We describe a general Multi-Party Computation (MPC) protocol for arithmetic circuits that is secure against a static malicious adversary corrupting up to a 1/7 fraction of the parties. The protocol requires each party to send an average of $O\big(\frac{m}{n}\log^3 n\big)$ bits, and compute $O\big(\frac{m}{n}\log^4 n\big)$ operations in a network of size n, where m is the size of circuit. This is achieved by increasing latency from constant to $O(d)$, where d is the depth of the circuit. Our protocol has a setup phase that is independent of the circuit and relies on Threshold Fully Homomorphic Encryption (TFHE). The setup requires each party to send $\tilde{O}(\kappa^2)$ messages and compute $\tilde{O}(\kappa^2)$ operations, where κ is the security parameter. We provide results from microbenchmarks conducted over a sorting network showing that our protocol may be practical for deployment in large networks. For example, we consider a network of size 2^{25} (over 33 million) where each party has an input item of size 20 bytes. To securely sort the items, our protocol requires each party on average to send 5 kilobytes per item sorted.

Keywords. Multiparty Computation; Byzantine fault tolerance; Threshold cryptography; Secret sharing.

1 Introduction

In secure Multi-Party Computation (MPC), a set of n parties, each having a secret value (input), want to compute a common function represented as a circuit over their inputs, without revealing the inputs to each other. This problem dates back to Yao [44] who proposed an algorithm for *two-party computation* in the presence of a *passive* adversary that can only read the internal state of the corrupted parties and their communication. Goldreich et al. [29] proposed the first multi-party computation protocol that is secure against an *active* adversary who can make the corrupted parties deviate from the protocol in any arbitrary fashion. In the last three decades, a large body of work has been devoted to designing MPC protocols for the active adversarial setting. Unfortunately, most of these protocols are inefficient for the case where the number of parties is large.

In this paper, we specifically address the problem of MPC when the number of parties is large. 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?

1.1 Our Results

We provide a general multiparty computation protocol with a communication complexity that is polylogarithmic in the number of parties and linear in the size of circuit. We prove the following main theorem (in Section B) under *τ-Strong Diffie-Hellman (τ-SDH)* and *τ-polynomial Diffie Hellman (τ-polyDH)* hardness assumptions defined in Section 3, where $\tau \le (1/3 - \epsilon)N$.

Theorem 1. *Let f be any deterministic function over n inputs in $\mathbb{Z}_p$ for prime $p = \mathsf{poly}(n)$, and $\mathscr{C}$ be a circuit with m gates and depth d that computes f. There exists an n-party protocol that securely computes f with the following properties:*

- *The protocol can tolerate up to $t < (1/7 - \epsilon)n$ malicious parties.*
- *Each party sends $O\left(\frac{m}{n}\log^3 n\right)$ messages and computes $O\left(\frac{m}{n}\log^4 n\right)$ operations.*
- *The protocol has a setup phase that is secure in the CRS model and requires each party to send $\tilde{O}(\kappa^2)$ messages and compute $\tilde{O}(\kappa^2)$ operations.*
- *Each message has $O(\log p)$ bits.*
- *The protocol has $O(d)$ rounds of communication.*

We also provide results from microbenchmarks for the problem of secure Multi-Party Sorting (MPS). MPS is useful in many applications such as anonymous communication [38, 9], privacy-preserving statistical analysis [22] (e.g., top-k queries [14]), auctions [45], and distributed intrusion detection [32]. It is often important for these applications to be run among *many* parties. For example, MPS is a critical component of communications algorithms that could enable the creation of large anonymous microblogging services without requiring trusted authorities (e.g., an anonymous Twitter). Our microbenchmarks consider many large network sizes. For example, we consider a network of size 2^{25} (over 33 million) where each party has an input item of size 20 bytes. To securely sort the items, our protocol requires each party on average to send 5 kilobytes per item sorted.

1.2 Technical Overview

A naive approach to solving MPC is for each party to divide its secret input into n pieces called *shares*, which reveal nothing about the input. Each party then sends one share to each of the other parties. In the next step, each party computes the circuit gate by gate over the shares and broadcasts the result. Finally, each party computes the circuit output using the pieces it receives at the end of the previous step.

Unfortunately, this algorithm does not always work correctly. Suppose $\mathscr{C}$ is an arithmetic circuit. If $\mathscr{C}$ has only addition gates, the algorithm is correct. However, if $\mathscr{C}$ also contains multiplication gates, there would likely be a problem. This is because most secret sharing schemes are not *multiplicatively homomorphic* meaning that the product of two shares is not necessarily a valid share of the product of the corresponding secrets. In the case where the secret-sharing technique is not multiplicatively homomorphic, certain techniques are required to build a valid share of the product. We are not aware of any such technique that can perform this, without requiring further rounds of communication and for any number of multiplication gates.

One approach for solving MPC is to use *Fully Homomorphic Encryption (FHE)*, which was recently made practical by Gentry. [25]. In this approach, each party first encrypts its input under an FHE scheme. The parties then evaluate the desired function on the encrypted data, and finally perform a distributed decryption on the final encrypted data to get the results. Due to the homomorphic properties of FHE, the decrypted value is the correct evaluation of the function over the inputs. FHE-based MPC protocols are usually optimal in terms of round complexity, however their computation cost is usually expensive. Current FHE schemes are very slow and can only evaluate circuits of small depth.

Our goal is to reduce both communication complexity[1] and computation complexity, and to this end we make a trade-off between these complexities and latency. We are inspired by the unconditionally-secure MPC algorithm of Dani et al. [21]. However, we make extensive use of cryptographic tools in order to reduce costs in practice. We reduce the costs further by performing local communications in polylogarithmic-size groups of parties called *quorums*, where the number of adversary-controlled parties in each quorum is at most a certain fraction. The quorums are created in a one-time setup phase that is secure in the *Common Reference String (CRS)* model[2]. The setup phase uses the quorum building algorithm of Santoni et al. [13] and the fully homomorphic encryption scheme of Brakerski et al. [12] to generate a number of parameters required for our protocol. Our online phase combines the circuit randomization technique of Beaver [4] and the efficient verifiable secret sharing scheme of Kate et al. [33] to perform fast computations on secret-shared values.

In our protocol, each gate of the circuit is assigned a quorum Q and the parties in Q are responsible for computing the function associated with that gate. Then, they send the result of this computation to any quorums associated with gates that need this result as input. Let Q' be one such quorum. It is necessary to securely send the output from Q to Q' without revealing any information to any individual party (or to any coalition of adversarial parties). This is a technically challenging problem.

Dani et al. [21] handle this problem by masking the result in Q and unmasking the result in Q'. This method is expensive in practice since parties in Q' need to reconstruct the masks jointly for each input. Boyle et al. [11] handle this problem by sending all the inputs to only one quorum which does all of the computation. This results in large computation and communication costs for parties in that quorum.

We introduce a new solution for this problem. We let each party in Q hold shares of the inputs to the gate associated with Q. Using homomorphic property of these shares, each party can run the gate function on their input to evaluate a share of the output. It is essential that parties in each quorum have a method to send the shares of the result to Q'. These shares cannot be the same shares because this would leak information to the adversary after multiple steps. Thus, in our algorithm, parties in Q jointly generate a *fresh* sharing of the output of the gate for Q'. Performing this fresh sharing correctly is one of the main technical challenges of this paper; it is

[1]Communication complexity is the number of bits transferred in the network.

[2]An algorithm is secure in the CRS model if it assumes all parties have access to a common random string taken from a predetermined distribution.

described as algorithm Reshare in Section 4.

1.3 Model

We consider a network of n parties whose identities are common knowledge. We assume there is a private and authenticated communication channel between every pair of parties and the communication is *synchronous*. Our protocol does not require the presence of any trusted third-party, and we do not assume the existence of a reliable broadcast channel. We assume $t < (1/7-\epsilon)n$ of the parties are controlled by a *malicious* adversary, for some fixed, positive constant ϵ. We assume our adversary is *computationally bounded* with respect to a security parameter[3] κ, and is actively trying to prevent the protocol from succeeding by attacking the privacy of the parties, and the integrity of communications, by attempting to corrupt, forge, or drop messages. We say that the parties controlled by the adversary are *malicious* and that the remaining parties are *semi-honest* (or simply, *honest*) meaning that they are curious to learn about other parties' secret information but they strictly follow the protocol. We finally assume that the adversary is *static* meaning that it must select the set of dishonest parties at the start of the protocol.

The rest of this paper is organized as follows. In Section 2 we discuss related work. Preliminaries are given in Section 3. In Section 4, we describe our protocol. The analysis is given in Section B. We conclude and give problems for future work in Section 6.

2 Related Work

The MPC problem dates back to Yao [44]. The first generic solutions presented in [29, 16, 24] are based on cryptographic assumptions. This work was followed by some unconditionally-secure schemes in late 1980s [7, 15, 37, 6, 30, 31, 5]. Unfortunately, these methods all have poor communication scalability that prevents their wide-spread use. 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 $\Omega(mn)$ (see [23, 28, 22]).

Recent years have seen exciting improvements in the cost of MPC when m is much larger than n [17, 19, 18]. For example, Damgard et al. give an algorithm with computation and communication cost that is $\tilde{O}(m)$ plus a polynomial in n [18]. However, the additive polynomial in n is large (e.g., $\Omega(n^6)$) and so these new algorithms are only efficient for relatively small n. Thus, there is still a need for MPC algorithms that are efficient in both n and m.

Boyle et al. [11] describe a synchronous cryptographic protocol to solve MPC using quorums. The algorithm is secure against an adversary that controls up to $(1/3-\epsilon)$ fraction of parties for any fixed positive ϵ. Interestingly, the communication costs are independent of circuit size, which is achieved by using a fully homomorphic encryption scheme. Unfortunately, the protocol is not fully load-balanced as it evaluates the circuit using only one quorum (called *supreme committee*) for performing general MPC. The protocol requires each party to send $\mathsf{polylog}(n)$ messages of size $\tilde{O}(n)$ bits and requires $\mathsf{polylog}(n)$ rounds.

[3]The security parameter controls the success probability of an adversary in breaking the security of our protocol in any way.

Dani et al. [21] describe an algorithm for solving MPC in a similar model we consider in this paper but with unconditional security. The algorithm creates logarithmic-size quorums using the quorum building algorithm of [34]. For each gate in the circuit, a quorum is used to compute the output of gate. The protocol ensures that all parties in the quorum learn the output of gate masked with a value selected uniformly at random. Thus, no party learns any information about the output, but the parties together have enough information to provide the input for computation of the masked output of the next gate. This procedure is repeated to for every level of gates in the circuit. At the top level of the circuit, the output of the function is computed and is sent down to all parties through all-to-all communication between the quorums. This algorithm has communication and computation complexity of $\tilde{O}(m/n + \sqrt{n})$ for each party. Although the algorithm scales well, it has several hidden logarithmic factors that make it inefficient in practice.

3 Preliminaries

In this section, we define standard terms and notation used throughout the paper, and describe the results we use in our protocol.

3.1 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 the paper, we assume g is a generator of the multiplicative group $\mathbb{G}$ of prime order p. We use the notation $\mathcal{C}_a$ to denote an encryption of a plaintext $a \in \mathbb{Z}_p$ ciphered by the encryption algorithm of Section 3.4, meaning that $\mathcal{C}_a = \mathsf{Enc}(a)$.

3.2 Verifiable Secret Sharing (VSS)

For $\eta, \tau \in \mathbb{N}$, where $\tau < \eta$, an (η, τ)*-secret sharing scheme* is a pair of algorithms (Share, Reconstruct) such that a dealer runs Share(s) to share a secret s among η parties, and any subset of $\tau + 1$ or more parties can compute s using Reconstruct, but no subset of τ or less parties can. An (η, τ)*-non-interactive verifiable secret sharing scheme* is a tuple of algorithms (Share, Reconstruct, Commit, Verify) such that (Share, Reconstruct) is an (η, τ)-secret sharing scheme, and

- there is no polynomial-time strategy for picking τ pieces of the secret, such that they can be used to predict the secret with any perceivable advantage, and
- either Reconstruct outputs s in which case Verify outputs true, or honest parties conclude that the dealer is malicious in which case Verify outputs false.

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

By Commit, the dealer commits to the set of shares that is constructed via Share, and by Verify, the parties verify the commitments. In our protocol, we make extensive use of a verifiable secret sharing protocol proposed by Kate et al. [33] called *eVSS*[5]. This protocol uses Shamir's secret sharing scheme [41] along with a commitment scheme that is computationally-hiding under the *Discrete Logarithm (DL) assumption* and computationally-binding under the *τ-Strong Diffie-Hellman (τ-SDH) assumption.*

Definition 1. [DL Assumption] Given $g, g^x \in \mathbb{G}$, computing x is computationally intractable.

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

Definition 3. [τ-polyDH Assumption] Let $a \in \mathbb{Z}_p^*$. Given $g, g^a, g^{(a^2)}, ..., g^{(a^\tau)} \in \mathbb{G}$, finding $\phi(x) \in \mathbb{Z}_p[x]$ and $g^{\phi(a)}$ is computationally intractable.

Boneh and Boyen [10] and Kate et al. [33] show that solving τ-SDH and τ-polyDH problems with $\tau < O(\sqrt[3]{p})$ each requires $\Omega(\sqrt{p/\tau})$ in expected time.

Theorem 2. [eVSS [33]] *There exists a synchronous (η, τ)-non-interactive verifiable secret sharing scheme for $\tau \leq (1/2 - \epsilon)\eta$ secure under the DL, τ-SDH, and τ-polyDH assumptions. In worst case, the protocol requires two broadcasts and four rounds of communication.*

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 [33].

3.2.1 Sharing Phase

For simplicity, we assume algorithm VShare executes Share, Commit, and Verify respectively representing the verifiable sharing phase (Sh) of eVSS [33]. To share a secret $\alpha \in \mathbb{Z}_p$ among η parties, a party (called *dealer*) runs $\mathsf{VShare}(s, \eta, \tau)$ to picks a random polynomial $\phi \in \mathbb{Z}_p[x]$ of degree d such that $\phi(0) = \alpha$, and sends each party P_i a share $\alpha_i = \phi(i)$, for all $i \in [\eta]$.

Definition 4. [Sharing] Let $\phi \in \mathbb{Z}_p[x]$ be a degree d polynomial. A *sharing* of $\alpha \in \mathbb{Z}_p$ among η parties is denoted by $S_\alpha = \langle \alpha_1, ..., \alpha_\eta \rangle$ and is defined as a set of values (*shares*) $\alpha_1, ..., \alpha_\eta \in \mathbb{Z}_p$ such that $\alpha_i = \phi(i)$ is held by the i-th party and $\phi(0) = \alpha$.

Definition 5. [(η, τ)**-Secrecy**] Let τ be a positive integer and $S_\alpha = \langle \alpha_1, ..., \alpha_\eta \rangle$ be a sharing of $\alpha \in \mathbb{Z}_p$ among η parties. We say S_α has (η, τ)-*secrecy* if and only if given a set of at most τ shares X, for every adversary $\mathcal{A}$ the probability $\Pr[\mathcal{A}(X) = \alpha] \leq 1/p$. Informally, this means S_α does not reveal any information about the secret α. We denote such a sharing by $S_\alpha^{(\tau)} = \langle \alpha_1, ..., \alpha_\eta \rangle_\tau$.

Theorem 3. *For positive integers η and τ,* VShare *generates a sharing with (η, τ)-secrecy if and only if the adversary can learn up to τ shares. Also,* VShare *is zero-knowledge for any malicious verifier.*

Proof. We prove the secrecy property in Lemma 5. The zero-knowledge property is proved in [33]. $\square$

[5]stands for efficient VSS.

3.2.2 Reconstruction Phase

During the secret reconstruction phase, any $\tau + 1$ or more parties send their accepted shares to all other parties. In the malicious case, it is possible that dishonest parties send spurious shares, i.e., values that are different from values given to them in the sharing phase. In eVSS, this is solved by asking all parties to broadcast a proof (called *witness*) during reconstruction to verify broadcast shares [33]. In our protocol, the reconstruction phase is postponed to after circuit computation. Since the witnesses are generated in the sharing phase at the beginning of the computation, and they do not have necessary homomorphic properties, we cannot use them for consistency checking in our reconstruction phase.

In this paper, we use a different reconstruction method that does not depend on witnesses. The method is proposed by McEliece and Sarwate [36], and is based on the properties of *Reed-Solomon (RS) codes* [39]. RS codes can be used along with Shamir's scheme for detecting and correcting up to $\eta/3 - 1$ errors in shares. Using an RS decoding algorithm, the secret can be successfully recovered. In our protocol, we use the efficient RS decoder of Berlekamp and Welch [8].

Welch-Berlekamp Error Correction. 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 $E(x) = e_0 + e_1 x + ... + e_\varepsilon x^\varepsilon$ of degree ε, and $Q(x) = q_0 + q_1 x + ... + q_k x^k$ of degree $k \leq \varepsilon + \tau - 1$ such that $y_i E(x_i) = Q(x_i)$ for all $i \in [\eta]$. This defines a system of η linear equations with $\varepsilon + k = \eta$ variables $e_0, ..., e_\varepsilon, q_0, ..., q_k$ that can be solved efficiently using Gaussian elimination technique to get the coefficients of $E(x)$ and $Q(x)$. Finally, calculate $P(x) = Q(x)/E(x)$.

3.3 Quorum Formation

Similar to [21], we reduce the amount of communication required for our protocol by creating n groups of $N = O(\log n)$ parties called *quorums*, where at most $T = (1/6 - \epsilon)N - 1$ of the parties in each quorum are dishonest. Scalability is achieved by allowing parties of each quorum to communicate only with members of the same quorum and members of a constant number of other quorums. A *Byzantine Agreement (BA)* algorithm can be used to build a set of quorums [34, 13]. In this paper, we use the BA algorithm of Santoni et al. [13] for quorum formation.

Theorem 4. [13] *There exists an unconditionally-secure protocol that brings all good parties to agreement on n quorums with high probability. The protocol has amortized communication and computation complexity*[6] $\tilde{O}(1)$ *and runs in polylogarithmic time.*

Reliable Broadcast. In the malicious model, when parties have only access to secure pairwise channels, a protocol is required to ensure reliable broadcast. Such a broadcast protocol guarantees all parties receive the same message even if the broadcaster is dishonest and sends different messages to different parties. A BA protocol can be used to perform reliable broadcast. We use

[6]Amortized communication complexity is the total number of bits exchanged divided by the number of parties. Since the protocol is not load-balanced, the amortized complexity is calculated, which is essentially the same as per-party complexity for load-balanced protocols.

the BA algorithm of Santoni et al. [13] to perform broadcasts in our protocol. The following theorem will be used in our protocol.

Theorem 5. [13] *There exists an unconditionally-secure protocol for performing reliable broadcast in a network with n parties connected via secure pairwise channels, where at most a 1/3 fraction of the parties are malicious. The protocol has amortized communication and computation complexity* $\tilde{O}(1)$ *and runs in polylogarithmic time.*

3.4 Fully Homomorphic Encryption (FHE)

An FHE scheme allows to perform non-interactive secure computation, which is very useful in designing communication efficient MPC protocols. Gentry [25] 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 [43, 12, 26], 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. [20], 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) [12] that is based on *ring learning with error (R-LWE) assumption* and provides an effective approach for controlling the noise level of ciphertexts. LWE [40] 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 [35]. In order to make homomorphic computations of BGV faster, we use the ciphertext packing technique of Smart and Vercauteren [42] and the optimizations of Gentry et al. [27].

Theorem 6. [12] *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*[7].

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 paper, we use the TFHE scheme of Asharov et al. [2] that is based on the FHE construction of BGV.

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

Theorem 7. [2] *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 $\leq D$.*

4 Our Protocol

In this section, we present our protocol for general MPC. Let f be a deterministic function computed by an arithmetic circuit $\mathscr{C}$ of depth d and depth $|\mathscr{C}|$. Let $G_1, ..., G_{|\mathscr{C}|}$ denoted the gates of $\mathscr{C}$, where instead of only $+$ or $\times$, each gate $G_\ell \in \mathscr{C}$ for $\ell \in [|\mathscr{C}|]$ can compute an arithmetic circuit $\mathscr{C}_{G_\ell}$ that has at most two inputs and at most two outputs[8]. In $\mathscr{C}$, every gate with indegree zero is called an *input gate*, and every gate with outdegree zero is called an *output gate.* Let $G_1, ..., G_n$ be the input gates. Consider n parties $P_1, P_2, ..., P_n$ with inputs $x_1, ..., x_n \in \mathbb{Z}_p$, who want to jointly evaluate $\mathscr{C}$ over their inputs. Let denote the number of gates in $\mathscr{C}$. Our main algorithm proceeds as follows.

Algorithm 1 Main

1. **Setup:**
   (a) *Quorum Building*: Parties run the quorum formation algorithm of Theorem 4 to agree on n quorums $Q_1, ..., Q_n$. For all $\ell \in [|\mathscr{C}|]$, 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 Broadcast:** For all $i \in [n]$, party P_i runs $\mathsf{VShare}(x_i, N, \tau)$ in Q_i associated with input gate G_i, where $\tau = (1/3 - \epsilon)N$.

3. **Circuit Computation:** The circuit $\mathscr{C}$ is evaluated level-by-level starting from the input gates. For each gate G in $\mathscr{C}$, the associated quorum Q_G computes $\mathscr{C}_G$ in the following way. Let $\langle \alpha_1, ..., \alpha_N \rangle_\tau$ and $\langle \beta_1, ..., \beta_N \rangle_\tau$ be the sharings associated with the inputs of G. For each gate g in $\mathscr{C}_G$, 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 $\mathscr{C}_G$ with output value γ_i , then P_i runs $\mathsf{Reshare}(\gamma_i, Q_{G'})$ 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 $\mathsf{Output}(z)$.

[8]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 $\times$ (called a *multiplication gate*).

In the rest of this section, we define various algorithms used in Main. 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 B.

4.1 Setup

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 $\mathscr{C}_G$ corresponding to a gate G of $\mathscr{C}$, 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.

As described in algorithm Main, inputs of G are securely shared in Q using VShare, 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 [20]. 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 InitTriple

Usage. Each party P_i jointly computes a triple (u_i, v_i, w_i), where $\langle u_1, ..., u_N \rangle_\tau$, $\langle v_1, ..., v_N \rangle_\tau$, and $\langle w_1, ..., w_N \rangle_\tau$ are sharings of u, v, and w respectively, where $u, v \in \mathbb{Z}_p$ are chosen uniformly at random, $w = u \cdot v$, and $\tau = (1/3 - \epsilon)N$.

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 $\{(\mathcal{C}_{a_j}, \mathcal{C}_{b_j})\}_{j=1}^{N}$ be the set of pairs P_i receives from the previous step[9]. P_i computes
$$\mathcal{C}_u = \sum_{j=1}^{N} \mathcal{C}_{a_j},\ \mathcal{C}_v = \sum_{j=1}^{N} \mathcal{C}_{b_j}, \text{ and } \mathcal{C}_w = \mathcal{C}_u \cdot \mathcal{C}_v.$$
Parties runs $\mathsf{CipherShare}(\mathcal{C}_u)$, $\mathsf{CipherShare}(\mathcal{C}_v)$, and $\mathsf{CipherShare}(\mathcal{C}_w)$ to generate three sharings $\langle \mathcal{C}_u \rangle_\tau$, $\langle \mathcal{C}_v \rangle_\tau$, and $\langle \mathcal{C}_w \rangle_\tau$.
3. For all $i \in [N]$, party P_i runs $u_i = \mathsf{DecPrivate}(\mathcal{C}_{u_i})$, $v_i = \mathsf{DecPrivate}(\mathcal{C}_{v_i})$, and $w_i = \mathsf{DecPrivate}(\mathcal{C}_{w_i})$.

[9]Throughout the paper, 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 3 CipherShare

Usage. Initially, all parties in Q hold a common ciphertext $\mathcal{C}_u$. Using the algorithm, parties jointly convert $\mathcal{C}_u$ into a sharing $\langle \mathcal{C}_{u_1}, ..., \mathcal{C}_{u_N} \rangle_\tau$, where u_i is an eVSS share of $u \in \mathbb{Z}_p$ and $\tau = (1/3 - \epsilon)N$.

CipherShare($\mathcal{C}_u$):

For all $i \in [N]$,

1. Party P_i runs GenRand to jointly generate a sharing $\langle r_1, ..., r_N \rangle_{(\tau-1)}$ of a uniform random value $r \in \mathbb{Z}_p$.
2. P_i computes $\mathcal{C}_{u_i} = \mathcal{C}_u + \mathsf{Enc}(i \cdot r_i)$. The party sends its share $\mathcal{C}_{u_i}$ to all parties in Q via one-to-one communication.
3. Let $\mathcal{C}_{u_1}, ..., \mathcal{C}_{u_N}$ 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 τ. For all $j \in [N]$, if $\phi'(j) \neq \mathcal{C}_{u_j}$, then P_i concludes that P_j is dishonest, and ignores its share $\mathcal{C}_{u_j}$.

We now prove the correctness of CipherShare. From the correctness of GenRand, 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 $\tau - 1$ 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 $\phi_2(x) = x \cdot \phi_1(x)$. 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 $\phi_3(x) = w + \phi_2(x)$. 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 eVSS share of w. Thus, using the homomorphic properties of Enc,

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

Step 3 is correct because the Welch-Berlekamp algorithm (Section 3.2) can be represented as an arithmetic circuit and thus, can be computed over cipher inputs using the homomorphic properties of the TFHE scheme. □

We define GenRand using a simple and well-known technique for generating uniformly random secrets (as also used by Beaver [4]), which is done by adding shares of uniformly random secrets received from all parties.

Algorithm 4 GenRand

Usage. Parties jointly generate a sharing $\langle u_1, ..., u_N \rangle_\tau$ of a value $u \in \mathbb{Z}_p^*$ chosen uniformly at random.

GenRand(τ):

For all $i \in [N]$,

1. Party P_i chooses $\rho_i \in \mathbb{Z}_p^*$ uniformly at random, and runs $\mathsf{VShare}(\rho_i, N, \tau)$ to generate a sharing $\langle \rho_{i1}, ..., \rho_{iN} \rangle_\tau$.
2. Let $\rho_{1i}, ..., \rho_{Ni}$ be the shares P_i receives from step 1. P_i computes $u_i = \sum_{j=1}^{N} \rho_{ji}$.

Algorithm 5 DecPrivate

Usage. Initially, all parties in Q hold a common ciphertext $\mathcal{C}_u$. Using this algorithm, parties in Q jointly decrypt $\mathcal{C}_u$ 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($\mathcal{C}_u$):

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

4.2 MPC in Quorums

In this section, we give the two main algorithms that are used by quorums in the online phase to evaluate gates in the circuit. A product gates G associated with quorum Q and two shared inputs $\alpha, \beta \in \mathbb{Z}_p$ is computed as follows. Let P_i be a party in Q, who holds α_i and β_i as the shares of α and β respectively and let (u_i, v_i, w_i) be a multiplication triple generated using InitTriple for P_i. The party then uses the technique proposed by Beaver [4] for computing a share of the product $\gamma = \alpha \cdot \beta$ as described in algorithm Multiply.

Algorithm 6 Multiply

Usage. Initially, parties jointly hold two sharings $\langle \alpha_1, ..., \alpha_N \rangle_\tau$ and $\langle \beta_1, ..., \beta_N \rangle_\tau$ of secret values $\alpha, \beta \in \mathbb{Z}_p$ respectively, where $\tau < (1/3 - \epsilon)N$. For $i \in [N]$, each party P_i also hold a triple (u_i, v_i, w_i) generated during the setup phase of the protocol. The algorithm computes a new sharing $\langle \gamma_1, ..., \gamma_N \rangle_\tau$ of $\gamma \in \mathbb{Z}_p$ such that $\gamma = \alpha \cdot \beta$.

Multiply(α_i, β_i):

For all $i \in [N]$, party P_i computes $\varepsilon_i = \alpha_i + u_i$ and $\delta_i = \beta_i + v_i$ and runs Reconst(ε_i) and Reconst(δ_i) to learn ε and δ. Party P_i computes and returns $\gamma_i = w_i - \delta\alpha_i - \varepsilon\beta_i + \varepsilon\delta$.

Clearly, ε and δ can be safely revealed to all parties so that each party can compute $\varepsilon\delta$ locally. Correctness and security of Multiply are proved by Beaver in [4]. Once the computation in Q is finished, the parties in Q send the result to any quorums associated with gates that need this result as input. Let Q' be one such quorum. It is necessary to securely send the output from Q to Q' without revealing any information to any individual party or to any coalition of adversarial parties. We refer to this problem as *resharing*, which is depicted in Figure 1 (in the figure, G refers to the gate associated with Q and F_G refers to the functionality of G). Algorithm Reshare describes a simple technique reshare the output of Q in Q'.

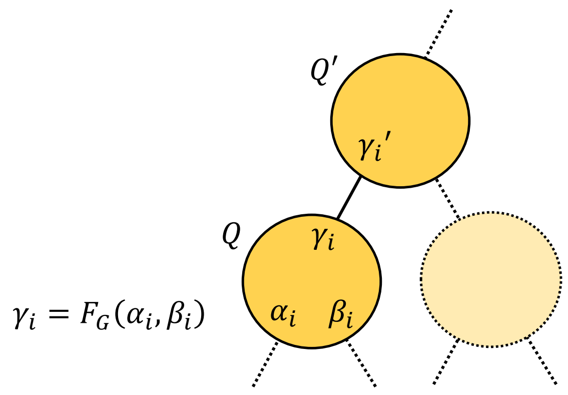


Figure 1: Resharing output of Q in Q'.

Algorithm 7 Reshare

Usage. Let Q be the quorum associated with gate G in circuit $\mathscr{C}$, and Q' be the quorum associated with a parent of G in $\mathscr{C}$. Initially, parties in Q hold a sharing $S_\gamma = \langle \gamma_1, ..., \gamma_N \rangle_\tau$ of a secret $\gamma \in \mathbb{Z}_p$ over $\phi \in \mathbb{Z}_p[x]$, where $\mathsf{deg}(\phi) = (1/3 - \epsilon)N$ and $\tau = (1/3 - \epsilon)N - 1$. Using this algorithm, parties in Q jointly generate a *fresh* sharing of γ in Q'. More formally, they generate a new sharing $S_{\gamma'} = \langle \gamma'_1, ..., \gamma'_N \rangle_\tau$ of a value $\gamma' \in \mathbb{Z}_p$ in Q' such that $\gamma' = \gamma$.

Reshare(γ_i, Q'):

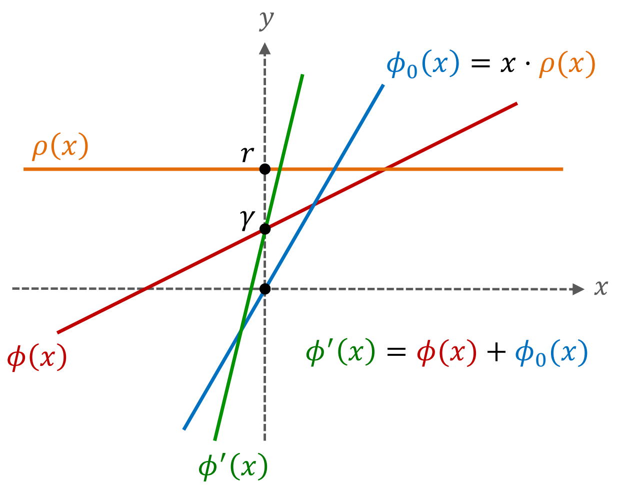
For all $i \in [N]$,

1. Party $P_i \in Q$ runs GenRand to jointly generate a sharing $\langle r_1, ..., r_N \rangle_\tau$ of a uniform random value $r \in \mathbb{Z}_p$ over a polynomial $\rho \in \mathbb{Z}_p[x]$, where $\mathsf{deg}(\rho) = \mathsf{deg}(\phi) - 1$.
2. $P_i \in Q$ computes $\gamma'_i = \gamma_i + i \cdot r_i$, and sends γ'_i to party $P_i \in Q'$.

We now prove the correctness of Reshare. Figure 2 shows a sketch of the proof. From the correctness of GenRand, $r_i \in \mathbb{Z}_p$ is a share of a global random value $r \in \mathbb{Z}_p$. Let $\phi_o(x) = x \cdot \rho(x)$. Clearly, $\mathsf{deg}(\phi_o) = \mathsf{deg}(\rho) + 1 = \mathsf{deg}(\phi)$. Now, define $\phi'(x) = \phi(x) + \phi_o(x)$. Since $\phi(0) = \gamma$ and $\phi_o(0) = 0$, we have $\phi'(0) = \gamma$. Hence, each party P_i can locally compute a new share of γ denoted by γ'_i from $\gamma'_i = \gamma_i + i \cdot r_i$, where $\gamma'_i = \phi'(i)$, $\gamma_i = \phi(i)$, and $i \cdot r_i = \phi_o(i)$. We prove the security of Reshare in Appendix B.5. □

4.3 Output Propagation

The last step of the protocol is to reconstruct the output values in output gates. This can be done by polynomial interpolation and error-correcting techniques to recover possible spurious shares sent by dishonest parties. Algorithm Reconst (Section A) implements this idea. Once the outputs are reconstructed, the next step is to send them to all parties in the network. Initially, all parties in the quorums associated with output gates hold the corresponding output values. The parties send their values to all parties of quorum Q_1. All parties in Q_1 then send the value to other quorums via a complete binary tree of quorums, rooted at Q_1. Algorithm Output (Section A) implements this idea.



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

Figure 2: Resharing technique

5 Microbenchmarks

We now describe the microbenchmarks for our protocol. We use our protocol to solve the problem of secure Multi-Party Sorting (MPS) over a large number network.

MPS can be performed efficiently using sorting networks. A *sorting network* is a network of *comparators*. Each comparator has 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. [1] proposes an asymptotically-optimal (depth $O(\log n)$) sorting network called *AKS*. Unfortunately, the AKS network is not practical due to large constants hidden in the depth complexity. Batcher [3] proposes an efficient and simple sorting network with depth $1/2 \log n(1 + \log n) = O(\log^2 n)$. In our simulations, we use Batcher's sorting network over n inputs. Each input is provided by a different party. Each comparator gate in the network uses the following simple arithmetic comparator formula.

Lemma 1. [Comparator Formula] *Let $F : \mathbb{Z}_p^2 \to \mathbb{Z}_p^2$ be a comparator function over two inputs $x_1, x_2 \in \mathbb{Z}_p$, i.e.,*

$$F(x_1, x_2) = \begin{cases} (x_1, x_2), & \text{if } x_1 \geq x_2 \\ (x_2, x_1), & \text{otherwise} \end{cases}$$

Then the following arithmetic formula over $\mathbb{Z}_p$ computes $(y_1, y_2) = F(x_1, x_2)$,

$$\begin{aligned} x_3 &= x_1 \cdot x_2^{-1},\ b = x_3 \cdot x_3^{-1} \\ y_1 &= b \cdot x_1 + (1 - b) \cdot x_2 \\ y_2 &= b \cdot x_2 + (1 - b) \cdot x_1. \end{aligned} \tag{1}$$

Proof. If $x_1 \geq x_2$, then $x_3 = x_1 \cdot x_2^{-1} \geq 1$, and $b = 1$. Thus, $y_1 = x_1$ and $y_2 = x_2$. Second, if $x_1 < x_2$, then $x_3 = x_1 \cdot x_2^{-1} = 0$, and $b = 0$. Thus, $y_1 = x_2$ and $y_2 = x_1$.

□

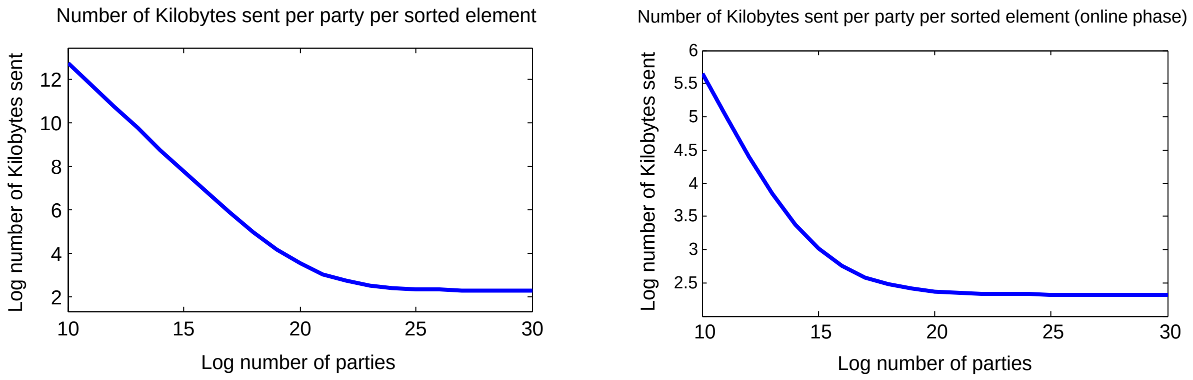


Figure 3: Communication cost for multiparty sorting

The circuit is computed in $\mathbb{Z}_p$ for a 160-bit prime p, with about 80-bit security. We set parameters to ensure error probability of at most 10^{-5} for the quorum formation algorithm. We ran the setup protocol once and then used the setup information to sort 100 vectors of random values, i.e., the online protocol was repeated 100 times with the same setup parameters. Consider n parties each with an arbitrary input from $\mathbb{Z}_p$. Let s be the total number of bits sent in the setup phase, and c be the total number of bits sent in the online phase for sorting 100 vectors. Let a_n be the average number of bits sent by each party for each sorted element received, in a network of size n. This is calculated from $a_n = \big(s + c\big)/100n^2$.

We repeated the experiment for network sizes ranging from $n = 2^{10}$ to $n = 2^{30}$. Figure 3 depicts the log-log plot for a_n as n varies. In both plots, we give the average number of kilobytes sent per party for each sorted element. In the left plot, we give an average that includes the entire setup phase (Section 4.1). In the right plot, we give an average that does not include this the setup phase.

For example, for sorting a vector of 2^{25} elements of $\mathbb{Z}_p$ in a network of size $n = 2^{25}$ (over 33 million parties), each party sends an average of 5 kilobytes[10] per each element of the sorted vector, as the left plot shows. The one-time setup for such a network requires an average of 21 kilobytes of communication per party. After this setup, the communication for the sorting of one vector is an average of 5 kilobytes per party.

6 Conclusion and Open Problems

We have described a MPC protocol that is communication efficient even when the number of parties is very large. Our protocol is robust against a static active adversary in the synchronous communication model. The protocol requires a total communication of $O(m \log^3 n)$ messages and a total computation $O(m \log^4 n)$ operations to evaluate a circuit with m gates and n inputs. There is also an initial setup phase with communication and computation complexity of $\tilde{O}(n\kappa^2)$. We reduce communication and computation costs by performing local communication in polylogarithmic-size groups of parties called quorums.

[10]i.e., 267 bits per each bit of the sorted vector

To examine performance of our protocol in practice, we performed microbenchmarks for the problem of multiparty sorting (MPS) in large networks. The results indicate that even for very large networks, it may be possible to jointly and securely sort. In particular, consider the situation where the input and network sizes are 2^{25} (over 33 million), and each input is 20 bytes in size. In such a scenario, our protocol requires each party to send on average 5 kilobytes per item sorted.

Several open problems remain. First, can we improve performance even further by detecting and blacklisting parties that exhibit adversarial behavior? We believe that such an approach could lead to significant practical improvements. Second, 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. Finally, can we adopt our results to a model that is more in line with fully-distributed peer-to-peer networks? In such networks, it is unlikely that each party knows the identities of every other party (which is a standard assumption in the MPC model).

References

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

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer Berlin Heidelberg, 2012.

[3] 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.

[4] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 1991.

[5] Z. Beerliova and M. Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference*, 2006.

[6] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth ACM Symposium on the Theory of Computing (STOC)*, 1993.

[7] M. Ben-Or, S. Goldwasser, and A. 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.

[8] E. Berlekamp and L. Welch. Error correction for algebraic block codes, US Patent 4,633,470, Dec. 1986.

[9] R. Berman, A. Fiat, and A. Ta-Shma. Provable unlinkability against traffic analysis. In A. Juels, editor, *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2004.

[10] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer Berlin Heidelberg, 2004.

[11] E. Boyle, S. Goldwasser, and S. 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.

[12] Z. Brakerski, C. Gentry, and V. 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.

[13] N. Braud-Santoni, R. Guerraoui, and F. 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.

[14] M. Burkhart and X. Dimitropoulos. Fast privacy-preserving top-k queries using secret sharing. In *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pages 1–7, Aug. 2010.

[15] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[16] D. Chaum, I. Damgård, and J. v. d. 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.

[17] I. Damgård and Y. Ishai. Scalable secure multiparty computation. *Advances in Cryptology - CRYPTO 2006*, pages 501–520, 2006.

[18] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology – CRYPTO 2008*, pages 241–261, 2008.

[19] I. Damgård and J. 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.

[20] I. Damgård, V. Pastro, N. P. Smart, and S. 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.

[21] V. Dani, V. King, M. Movahedi, and J. Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In M. Chatterjee, J.-n. Cao, K. Kothapalli, and S. Rajsbaum, editors, *Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2014.

[22] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings of the 2001 Workshop on New Security Paradigms*, NSPW '01, pages 13–22, New York, NY, USA, 2001. ACM.

[23] K. Frikken. Secure multiparty computation. In *Algorithms and theory of computation handbook*, pages 14–14. Chapman & Hall/CRC, 2010.

[24] Z. Galil, S. Haber, and M. 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.

[25] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41^{st} annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[26] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31^{st} Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[27] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012.

[28] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 1998.

[29] O. Goldreich, S. Micali, and A. 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.

[30] M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology – CRYPTO 2001*, pages 101–118. Springer, 2001.

[31] M. Hirt and J. Nielsen. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. *Advances in Cryptology - ASIACRYPT 2005*, pages 79–99, 2005.

[32] K. V. Jónsson, G. Kreitz, and M. Uddin. Secure multi-party sorting and applications. Cryptology ePrint Archive, Report 2011/122, 2011.

[33] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology – ASIACRYPT 2010 - 16^{th} International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[34] V. King, S. Lonergan, J. Saia, and A. Trehan. Load balanced scalable Byzantine agreement through quorum building, with full information. In *International Conference on Distributed Computing and Networking (ICDCN)*, 2011.

[35] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer Berlin Heidelberg, 2010.

[36] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Commun. ACM*, 24(9):583–584, Sept. 1981.

[37] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, STOC '89, pages 73–85, New York, NY, USA, 1989. ACM.

[38] C. Rackoff and D. 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.

[39] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, pages 300–304, 1960.

[40] O. 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.

[41] A. Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613, 1979.

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

[43] M. van Dijk, C. Gentry, S. Halevi, and V. 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.

[44] A. 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.

[45] B. Zhang. Generic constant-round oblivious sorting algorithm for MPC. In X. Boyen and X. Chen, editors, *Provable Security*, volume 6980 of *Lecture Notes in Computer Science*, pages 240–256. Springer Berlin Heidelberg, 2011.

A Output Propagation Algorithms

Algorithm 8 Reconst

Usage. Initially, parties jointly hold a sharing $\langle u_1, ..., u_N \rangle_\tau$ of a secret $u \in \mathbb{Z}_p$, where $\tau = (1/3 - \epsilon)N$. Using this algorithm, parties jointly reconstruct the secret, i.e., all parties learn the value u.

Reconst(u_i):

For all $i \in [N]$,

1. P_i sends its share u_i to all parties via one-to-one communication.
2. Let $u_1, ..., u_N$ be the messages P_i receives from the previous step.
3. P_i computes a polynomial $\phi(x)$ using the Lagrange interpolation polynomial,
$$\phi(x) = \sum_{i=1}^{\tau} u_i \prod_{j=1, j \neq i}^{\tau} (x-j)(i-j)^{-1}$$
4. For all $j \in [N]$, if there exists at least one u_j such that $\phi(j) \neq u_j$, then P_i runs the Welch-Berlekamp algorithm to recover the correct polynomial $\phi'(x)$ of degree τ. For all $j \in [k]$, if $\phi'(j) \neq u_j$, then P_i concludes that P_j is dishonest and must be disqualified.

Algorithm 9 Output

Usage. Let $\{Q_1, ..., Q_n\}$ be the set of all quorums. Initially, all parties in the quorum Q associated with output gate G, hold a value $z \in \mathbb{Z}_p$. Using this algorithm, the parties in Q send z to all parties in the network.

Output(z):

1. Each party in Q sends z to all parties in Q_1.
2. Each party in Q_1 considers the message with majority as z and send z to all parties in Q_2 and Q_3.
3. For all $2 \leq i \leq \lfloor n/2 \rfloor$, each party in quorum Q_i receives z from $Q_{\lfloor i/2 \rfloor}$via majority filtering. Each party in Q_i sends z to all parties in Q_{2i} and Q_{2i+1}.

B Correctness and Security Proofs

B.1 Proof of GenRand

Lemma 2. *The algorithm GenRand generates a sharing $\langle u_1, ..., u_N \rangle_\tau$ such that the distribution of u is uniform over $\mathbb{Z}_p$, and u is independent of all other random numbers generated during the protocol. Moreover, the distribution of each u_i where $i \in [N]$, is uniform over $\mathbb{Z}_p$, and u_i is independent of all other random numbers generated during the protocol.*

Proof. Correctness and security of the algorithm are proved as follows.

Correctness. For all $i \in [N]$, party P_i commits to $\rho_i \in \mathbb{Z}_p$ by running $\mathsf{VShare}(\rho_i, N, \tau)$. If P_i is honest, then ρ_i and its shares are uniform random values, otherwise ρ_i and its shares can be any spurious value. However, once P_i commits to ρ_i, the adversary can change neither ρ_i, its shares nor $\sum_{i=1}^{N} \rho_i$. Let $u = \sum_{i=1}^{N} \rho_i$. The adversary cannot bias the distribution of u and its shares from the uniform distribution because there exists at least one ρ_j such that it is chosen uniformly at random by an honest party and its shares are also chosen uniformly at random by eVSS sharing scheme. [11]. The value u and its shares are independent of other random values generated during the protocol because all honest parties choose their values independently. Finally, since eVSS shares are additively homomorphic, $u_i = \sum_{j=1}^{N} \rho_{ji}$ is a valid share of $u = \sum_{i=1}^{N} \rho_i$.

Security. Although the adversary has access to up to $T \leq \tau$ shares of $\langle u \rangle_\tau$, based on the security of eVSS scheme, it does not have enough information to reconstruct u.

Costs. The communication and computation cost of $\mathsf{GenRand}$ is equal to the communication and computation cost of running N different instantiation of $\mathsf{VShare}(\rho_i, N, \tau)$ algorithm. Thus, the communication cost of $\mathsf{GenRand}$ is $O(N^3)$, and the computation cost of $\mathsf{GenRand}$ is $O(N^2 \log p)$, where $\log p$ is the size of $\mathbb{Z}_p$. □

B.2 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 $\left\{(\mathcal{C}_{a_j}, \mathcal{C}_{b_j})\right\}_{j=1}^{N}$. Let $u = \sum_{j=1}^{N} a_j$ and $v = \sum_{j=1}^{N} b_j$. Since $\mathcal{C}_{a_i}$ and $\mathcal{C}_{b_i}$ are homomorphic ciphertexts, $\sum_{j=1}^{N} \mathcal{C}_{a_j} = \mathcal{C}_{\sum_{j=1}^{N} a_j} = \mathcal{C}_u$ and $\sum_{j=1}^{N} \mathcal{C}_{b_j} = \mathcal{C}_{\sum_{j=1}^{N} b_j} = \mathcal{C}_v$. 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 $\mathsf{CipherShare}$ and $\mathsf{DecPrivate}$. Since $\mathsf{DecPrivate}$ is correct, step 3 of the algorithm reveals u_i, v_i, and w_i only to P_i. The $\mathsf{CipherShare}$ algorithm requires each party to calculate a random polynomial on encrypted values. Lemma 3 shows how homomorphic properties of Enc can be applied to an eVSS sharing.

Steps 1 and 2 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 step 3 follows by the security of $\mathsf{DecPrivate}$. Finally, although the adversary has access to up to $T \leq \tau$ shares of each sharing $\langle u \rangle_\tau$, $\langle v \rangle_\tau$, and $\langle w \rangle_\tau$, based on the security of eVSS scheme, it does not have enough information to reconstruct u, v, and w, respectively. The communication cost of $\mathsf{InitTriple}$ can be computed based on the cost of $\mathsf{CipherShare}$ and $\mathsf{DecPrivate}$. Thus, it is equal to $\mathsf{poly}(N)\mathsf{polylog}(N)$. □

Lemma 3. *For any $\tau < (1/3-\epsilon)N$, if $\langle w_1, ..., w_N \rangle_\tau$ is a sharing of $w \in \mathbb{Z}_p$, then $\langle \mathsf{Enc}(w_1), ..., \mathsf{Enc}(w_N) \rangle_\tau$ is a sharing of $\mathsf{Enc}(w)$.*

Proof. Let $\phi(x) \in \mathbb{Z}_p[x]$ be the polynomial representing the sharing $\langle w_1, ..., w_N \rangle_\tau$. This means that $\phi(x)$ passes through the point $(0, w)$, and $\phi(i) = w_i$. We define a new polynomial $\phi'(x) =$

[11]The sum of two or more values from $\mathbb{Z}_p^*$ is always a uniform random value if at least one of them is chosen uniformly at random

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

B.3 Proof of CipherShare

We have already proved the correctness in Section 4.1. Based on the security of GenRand, and the security of theorem 7, $\mathcal{C}_w$ and $\mathcal{C}_{u_i}$, 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 GenRand, 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 $\tau - 1$. $\phi_2(x) = x.\phi_1(x)$ is a new polynomial of degree τ. In this case at most $T + 1 < \tau/3$ 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 GenRand plus N^2 extra messages sent in step 2 of the algorithm. The computation cost of CipherShare is equal to the computation cost of GenRand plus $\mathsf{poly}(N)$ evaluations on encrypted data in step 3. Thus, the computation cost of CipherShare is equal to $\tilde{O}(\kappa + \log p)$. □

B.4 Proofs of DecPrivate and Reconst

Proof. [DecPrivate] Based on the result of [2], DecPrivate is correct and secure, its communication cost is N, and its computation cost is $\tilde{O}(\kappa)$. □

Proof. [Reconst] The correctness and security of Reconst follows from the correctness of Lagrange interpolation and Welch-Berlekamp decoding algorithm [8, 36]. 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 4. □

Lemma 4. [Welch-Berlekamp] *Given a set of η points $S = \big\{(x_1, y_1) \mid x_i, y_i \in \mathbb{F}_p\big\}_{i=1}^{\eta}$ as input, the Welch-Berlekamp algorithm (Section 3.2) can be represented as an arithmetic circuit of multiplicative depth poly(η) with computation cost of $O(\eta^3)$.*

B.5 Proof of Reshare

We have already proved the correctness of Reshare in Section 4.2. Theorem 8 shows the security of Reshare. We first describe a few lemmas that are later used in the proof of Theorem 8.

Lemma 5. *Let $\phi \in \mathbb{Z}_p[x]$ and the adversary know $c \le \mathsf{deg}(\phi)$ points on ϕ. Consider a sharing $S_\alpha = \big\langle \phi(1), ..., \phi(\eta) \big\rangle$ among η parties. S_α has (η, τ)-secrecy if and only if the adversary can learn at most $\tau \le \mathsf{deg}(\phi) - c$ shares via any coalition of malicious parties.*

Proof. In order to uniquely reconstruct a polynomial of degree $d \ge \tau$, at least $d+1 \ge \tau+1$ points are required. However, the adversary has access to at most $c + \tau \le \mathsf{deg}(\phi)$ points on ϕ. Since all elements of $\mathbb{Z}_p$ are equally likely to be the missing point, the adversary has at most $1/p$ chance to guess the correct point and uniquely reconstruct the polynomial. Therefore based on Definition 5, S_α does not reveal anything about $\alpha = \phi(0)$ to the adversary. □

Corollary 1. *If S_α is a sharing with (η, τ)-secrecy, then it also has (η, τ')-secrecy, where $\tau' < \tau$.*

Lemma 6. *If $S_\alpha = \big\langle \alpha_1, ..., \alpha_\eta \big\rangle$ and $S_\beta = \big\langle \beta_1, ..., \beta_\eta \big\rangle$ are sharings with (η, τ_1)-secrecy and (η, τ_2)-secrecy respectively, then $S_\gamma = \big\langle \alpha_1 + \beta_1, ..., \alpha_\eta + \beta_\eta \big\rangle$ has (η, τ_3)-secrecy, where $\tau_3 = \min(\tau_1, \tau_2)$.*

Proof. Let $\phi_\alpha, \phi_\beta, \phi_\gamma \in \mathbb{Z}_p[x]$ be the polynomials associated with S_α, S_β, and S_γ respectively. From Corollary 1, S_α and S_β both have (η, τ_3)-secrecy. Without loss of generality assume that the adversary learns a set of at most τ_3 points $S_{\phi_\alpha} = \big\{\big(x_i, \phi_\alpha(x_i)\big)\big\}_{i=1}^{\tau_3}$ on ϕ_α as well as a set of at most τ_3 points $S_{\phi_\beta} = \big\{\big(x_i, \phi_\beta(x_i)\big)\big\}_{i=1}^{\tau_3}$ on ϕ_β. The only information the adversary is given about ϕ_γ is a set of at most τ_3 points $S_{\phi_\gamma} = \big\{\phi_\gamma(x_i) = \phi_\alpha(x_i) + \phi_\beta(x_i)\big\}_{i=1}^{\tau_3}$ on ϕ_γ. Thus based on Lemma 5, S_γ has (η, τ_3)-secrecy. □

Lemma 7. *Let $\phi_1, \phi_2 \in \mathbb{Z}_p[x]$ be arbitrary 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 $\phi_1(x) = a_1 + ... + a_d x^{d-1} \in \mathbb{Z}_p[x]$ such that $\phi_2(x) = x \cdot \phi_1(x)$. □

Lemma 8. *Let $\phi_1 \in \mathbb{Z}_p[x]$, $\mathsf{deg}(\phi_1) \ge \tau$, and $\phi_2(x) = x \cdot \phi_1(x)$. If $\big\langle \phi_1(1), ..., \phi_1(\eta) \big\rangle$ has (η, τ)-secrecy, then $\big\langle \phi_2(1), ..., \phi_2(\eta) \big\rangle$ also has (η, τ)-secrecy.*

Proof. Clearly, $\mathsf{deg}(\phi_2) \ge \tau + 1$. Let S_τ be a set of at most τ points of ϕ_1 the adversary learns via a coalition of at most τ malicious parties. By Lemma 7, the only information the adversary learns about ϕ_2 is a set of at most $\tau + 1$ points $S_\tau \cup \big\{(0, 0)\big\}$. Hence, by Lemma 5 the adversary cannot reconstruct ϕ_2, i.e., $\big\langle \phi_2(1), ..., \phi_2(\eta) \big\rangle$ has (η, τ)-secrecy. □

Theorem 8. [Security of Reshare] Let Q be a quorum of size N that holds a sharing $S_\gamma = \big\langle \gamma_1, ..., \gamma_N \big\rangle_\tau$ of a secret γ over $\phi \in \mathbb{Z}_p[x]$, where $\mathsf{deg}(\phi) = (1/3 - \epsilon)N$ and $\tau = (1/3 - \epsilon)N - 1$. Also, let $S_{\gamma'} = \big\langle \gamma'_1, ..., \gamma'_N \big\rangle_\tau$ be a new sharing of γ correctly generated by Reshare in a quorum Q' of size N. Algorithm Reshare is secure against an adversary corrupting up to $T = (1/6 - \epsilon)N - 1$ parties in each of Q and Q'.

Proof. Let $\rho \in \mathbb{Z}_p[x]$ be the polynomial associated with $S_r = \big\langle r_1, ..., r_N \big\rangle$ generated in the first step of Reshare. Since $\mathsf{deg}(\phi) = (1/3 - \epsilon)N = \tau + 1$, based on Lemma 5, S_γ has (N, τ)-secrecy. Since $\mathsf{deg}(\rho) = \mathsf{deg}(\phi) - 1 = \tau$, based on Lemma 5, S_r also has (N, τ)-secrecy meaning that at least $\tau = (1/3 - \epsilon)N - 1$ shares are required to reconstruct r. Since at most $(1/6 - \epsilon)N - 1 < \tau$ parties are malicious in Q, the adversary learns nothing about r. Using Lemma 8 and S_r, the second

step of Reshare constructs a new sharing $\langle o_1, ..., o_N \rangle_\tau$, such that $o_i = i \cdot r_i$, for all $i \in [N]$. Finally, using Lemma 6 the algorithm constructs a new sharing $S_{\gamma'} = \langle \gamma'_1, ..., \gamma'_N \rangle_\tau$ such that $\gamma'_i = \gamma_i + o_i$, for all $i \in [N]$. Since $S_{\gamma'}$ has (N, τ)-secrecy, at least τ shares are required to reconstruct γ'. Since only at most $(1/6 - \epsilon)N - 1 < \tau$ parties are malicious in Q, the adversary learns nothing about γ' (Lemma 5).
Based on the correctness of Reshare, $\gamma' = \gamma$. Via a coalition of at most $2T$ malicious parties in Q and Q', the adversary can learn up to $2T$ shares of γ. Since S_γ has (N, τ)-secrecy, $S_{\gamma'}$ has $(N, \tau - 1)$-secrecy, and $2T = (1/3 - \epsilon)N - 2 = \tau - 2 < \tau$, the last step of the algorithm reveals nothing about γ (Lemma 5).
Based on the correctness and security of GenRand invoked in the first step of Reshare, r and its shares are uniformly random and independent of any other value used in the protocol. For all $i \in [N]$, since r_i is independent of γ_i, we can conclude that γ'_i is independent of γ_i. □

Cost. The communication and computation cost of Reshare is equal to the communication and computation cost of GenRand plus N extra messages/operations. □

B.6 Proof of Output

The correctness follows by induction. Based on the correctness of Reconst algorithm, all honest parties in the quorum associated with the output gate hold the correct value of z. By Lemma 9, honest parties in Q_1 receive z from the output quorum. Consider this as the base case for our proof. Suppose z has been learned by all honest parties in quorum Q_j, for all $j < i$. Consider the parties in Q_i. By induction hypothesis, all honest parties in quorum $\lfloor i/2 \rfloor$ have learned z. Thus, all honest parties in quorum $\lfloor i/2 \rfloor$ send z to all parties in Q_i. By Lemma 9, all honest parties in Q_i learn z via majority filtering. This completes the induction. It is clear that the communication and computation costs of Output is $O(nN^2)$. □

Lemma 9. *If all honest parties in Q sends a common message $z \in \mathbb{Z}_p$ to all parties in quorum Q', then z is received by all honest parties in Q'.*

Proof. The majority of the parties in each quorum are honest. In particular. Thus, the majority of the parties in Q send z to each party in Q'. It follows that each honest party in Q' must receive z from the majority of the parties of Q. □

B.7 Proof of Main

In the following, we prove algorithm Main and our main theorem (Theorem 1).

Setup. The correctness and security follows the proof of Theorems 4, 7, and 2, and the InitTriple algorithm. Since $T = (1/6 - \epsilon)N - 1$ and $\tau = (1/3 - \epsilon)N$, we have $T < \tau$. □

Input Broadcast. The correctness and security follows the proof of VShare(x_i, N, τ) in Q_i since $T < \tau$ because $T = (1/6 - \epsilon)N - 1$ and $\tau = (1/3 - \epsilon)N$. 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 computation phase. □

Circuit Computation. *Correctness.* We prove by induction in the real-ideal model. The invariant is that if the input shares are correct, then the output of each gate is equal to the output of the gate if the it is evaluated by a trusted party in the ideal model, and the result shared between parties correctly. For the base case, note that the invariant is true for input gates. Induction step is based on the correctness of $\gamma_i = \mathsf{Multiply}(\alpha_i, \beta_i)$ and $\gamma_i = \alpha_i + \beta_i$. Moreover, based on the correctness of $\mathsf{Reshare}(\gamma_i)$, the third step only refreshes the sharing, and does not change the value of γ. □

Security. We prove by induction that the adversary cannot obtain any information about the inputs and outputs during the computation of each gate of $\mathscr{C}$. Let Q_1, Q_2, and Q_3 be the quorums involved in computation of a gate G, where Q_1 and Q_2 provide the inputs to Q_3, and Q_3 computes the functionality of G. Consider any party P in the network. Let S be the set of all shares P receives during the protocol. We consider two cases. First, if party $P \notin (Q_1 \cup Q_2 \cup Q_3)$, then elements of S are independent of the shares Q_1 and Q_2 send to Q_3 as input. Moreover, elements of S are independent of Q_3's output before it sends fresh shares of the output to the parent quorums. Hence, S reveals no information about the inputs and outputs of G to dishonest parties.

Second, if party $P \in (Q_1 \cup Q_2 \cup Q_3)$, then the inductive invariant is that the combination of the shares held by dishonest parties in Q_1, Q_2, and Q_3 does not give the adversary any advantage. As the base case, it is clear that the invariant is valid for input gates. Induction step is as follows. Consider the worst case when inputs and outputs are the same (this happens for example when G is an identity gate). In this case, the adversary can obtain at most $3T < (1/3 - \epsilon)N - 3 = \tau - 3$ shares of any secret value during the computation phase. By the security of eVSS, at least $\tau + 1$ shares are required for reconstructing the secret. So, the re-sharing process to feed inputs to Q_3 does not reveal any information to the adversary. Since, eVSS shares are secure under addition and $\mathsf{Multiply}$ is secure, the security of computation phase is proved. □

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 $\langle\gamma\rangle_\tau$, where $i \in [N]$. Since the number of honest parties in the quorum is $N - T = N - (1/6 - \epsilon)N + 1 > \tau$, 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 $\mathsf{InitTriple}$ that are executed for each of the n quorums by their $N = O(\log n)$ parties. TGen communication cost is $O(N^2 \mathsf{polylog}(N) D\kappa^2)$, 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 $\mathsf{InitTriple}$, 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 Broadcast phase is equal to the communication cost of running n different instantiation of $\mathsf{VShare}(\rho_i, N, \tau)$. 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 $\mathsf{Multiply}$ and $\mathsf{Reshare}$, 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 $\mathsf{Reconstruct}$ 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 $O(m \log^4(n))$. $\square$