Recent Results in Scalable Multi-Party Computation

Jared Saia and Mahdi Zamani

Dept. of Computer Science, University of New Mexico, Albuquerque, NM, USA 87131
{saia,zamani}@cs.unm.edu

Abstract. Secure multi-party computation (MPC) allows multiple parties to compute a known function over inputs held by each party, without any party having to reveal its private input. Unfortunately, traditional MPC algorithms do not scale well to large numbers of parties. In this paper, we describe several recent MPC algorithms that are designed to handle large networks. All of these algorithms rely on recent techniques from the Byzantine agreement literature on forming and using *quorums*. Informally, a quorum is a small set of parties, most of which are trustworthy. We describe the advantages and disadvantages of these scalable algorithms, and we propose new ideas for improving practicality of current techniques. Finally, we conduct simulations to measure bandwidth cost for several current MPC algorithms.

1 Introduction

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

In this paper, we focus on *scalable* MPC algorithms, which are designed to be resource-efficient (*e.g.*, in terms of bandwidth, computation, and latency) for large networks. Scalable MPC is of importance for many applications over 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? How can we perform data mining over data spread over large numbers of machines?

Although much theoretical progress has been made in the MPC literature to achieve scalability (*e.g.*, [DIK$^+$08,DIK10,AJLA$^+$12,BGT13,DKMS14,BCP14a]), practical progress is slower. In particular, most known schemes suffer from either poor or unknown communication and computation costs in practice.

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

1.1 Problem Statement

In the MPC problem, a set of n parties, each holding a private input, jointly evaluate a function f over their inputs while ensuring,

1. Upon termination of the protocol, all parties have learned the correct output of f; and
2. No party learns any information about other parties' inputs other than what is revealed from the output.

We assume the identities of the n parties are common knowledge, and there is a private and authenticated communication channel between every pair of parties. We consider two communication models. In the *synchronous* model, there is an upper bound, known to all parties, on the length of time that a message can take to be sent through a channel. In the *asynchronous* model, there is no such upper bound.

Usually, a certain fraction of the parties are controlled by a *Byzantine*[1] adversary. These parties can deviate arbitrarily from the protocol. In particular, they can send incorrect messages, stop sending any messages, share information amongst themselves, and so forth. Their goal is to thwart the protocol by either obtaining information about the private inputs, or causing the output of the function to be computed incorrectly. We say the adversary is *semi-honest* if the adversary-controlled parties are curious to learn about other parties' secret information, but they strictly follow the protocol. We say that the parties controlled by the adversary are *malicious* (or *Byzantine* or *dishonest*). The remaining parties are called *semi-honest* (or simply, *honest*).

The adversary is either *computationally-bounded* or *computationally-unbounded.* The former is typically limited to only *probabilistic polynomial-time (PPT)* algorithms, and the latter has no computational limitations. The adversary is either assumed to be *static* or *adaptive.* A static adversary is limited to selecting the set of dishonest parties at the start of the protocol, while an adaptive adversary does not have this limitation.

1.2 Measures of Effectiveness

The following metrics are typically used to measure the effectiveness of MPC protocols.

- **Resource costs.** These include *communication cost* (number of messages sent and size of each message), *computation cost*, and *latency* (number of rounds of communication). We remark that load-balancing may be important for all of these resources.
- **Fault Tolerance.** These metrics measure to what degree a protocol can tolerate adversarial attack. They include: the number of nodes that an adversary can take over (without sacrificing correctness); the type(s) of faults, *i.e.*,

[1] Also known as *active* or *malicious.*

Byzantine, crash faults, randomly, or adversarially distributed; the number of bits in messages that can be corrupted by an adversary; and the amount of churn that the protocol can tolerate

1.3 MPC and Byzantine Agreement

In the Byzantine setting, the MPC problem is tightly related to the problem of *Byzantine agreement (BA)*, where a group of n parties each holding an input value want to agree on a common value. In a celebrated result, Pease, Shostak, and Lamport [PSL80] proved that perfectly-secure BA can be achieved as long as less than one third fraction of the parties is corrupted. There are several interesting connections between BA and MPC:

1. BA can be seen as MPC for the simplest type of function: a function that must return a bit equal to the input bit of at least one honest party. However, BA is simpler that MPC in that it is not necessary to maintain privacy of inputs.
2. MPC protocols strongly rely on the use of a *broadcast channel* which is typically realized using a BA protocol.[2] Most MPC results so far assume the existence of a broadcast channel. Unfortunately, this requirement is highly problematic in settings, where the number of parties is large.
3. Several recent MPC schemes [BGT13,DKMS14,BCP14a,ZMS14] crucially build upon the notion of *quorums*[3] for achieving scalability. A quorum is a polylogarithmic set of parties, where the number of corrupted parties in each quorum is guaranteed not to exceed a certain fraction. King *et al.* [KLST11] show how to use BA to efficiently create a collection of quorums.

Paper organization. The rest of this paper is organized as follows. In Section 2, we review related work with a focus on MPC for many parties (*i.e.*, scalable MPC). In Section 3, we describe key open problems in scalable MPC. In Section 4, we describe algorithmic tools used in current scalable MPC algorithms. Section 5 describes recent quorum-based results for scalable MPC, and defines and analyzes new techniques for improving these results. Finally, we conclude in Section 6.

2 Related Work

Due to the large body of work, we do not attempt a comprehensive review of the MPC literature here, but rather focus on work that is relevant to scalable MPC.

[2] The standard definition of MPC (as given in Section 1.1) implies Byzantine agreement. Goldwasser and Lindell [GL02] show that a relaxed definition of MPC allows MPC without a broadcast channel (and hence without Byzantine agreement).

[3] Also known as *committees*.

The MPC problem was first described by Yao [Yao82]. He described an algorithm for MPC with two parties in the presence of a semi-honest adversary. Goldreich *et al.* [GMW87] propose the first MPC protocol that is secure against a Byzantine adversary. This work along with [CDG88,GHY88] are all based on cryptographic hardness assumptions and are often regarded as the first generic solutions to MPC. These were followed by several cryptographic improvements [BMR90,GRR98,CFGN96] as well as information theoretically-secure protocols [BGW88,CCD88,Bea91,BCG93] in late 1980s and 1990s. Unfortunately, these methods all have poor communication scalability. In particular, if there are n parties involved in the computation, and the function f is represented by a circuit with m gates, then these algorithms require each party to send a number of messages and perform a number of computations that is $\Omega(mn)$. In 2000s, exciting improvements were made to the cost of MPC, when m (*i.e.*, the circuit size) is much larger than n [DI06,DN07,DIK$^+$08]. For example, Damgard *et al.* [DIK$^+$08] give an algorithm with computation and communication cost that is $\tilde{O}(m)$ plus a polynomial in n. Unfortunately, the additive polynomial in these algorithms is large (at least $\Omega(n^6)$) making them impractical for large n.

Depending on the model of computation, every function can be represented in terms of some *elementary operations* such as arithmetic operations (*e.g.*, addition, multiplication), Boolean operations (*e.g.*, and, or), RAM instructions (*e.g.*, get-value, set-value), etc. Informally speaking, every MPC protocol specifies how a group of elementary operations can be computed securely. The function is computed securely via composition of these secure operations. From this perspective, we classify the broad range of MPC approaches into two categories: techniques that evaluate circuits (Boolean or arithmetic), and techniques that evaluate RAM programs.

2.1 Circuit-Based Techniques

We subdivide the set of circuit-based methods into three categories based on their main approach for achieving privacy: garbled circuits, secret sharing, and fully homomorphic encryption. Although some protocols such as [DPSZ12,BGT13] may fall into more than one category, most protocols follow only one as their main approach.

Garbled Circuits. The idea of garbled circuits dates back to the two-party MPC proposed by Yao [Yao82].[4] One party is called the *circuit generator* and the other one is called the *circuit evaluator*. For each wire in the circuit, the generator creates a mapping that maps each possible value of that wire to another value (called the *garbled value*). The generator then sends this mapping to the evaluator. The evaluator evaluates the circuit using the mapping to compute the *garbled output*. Next, the generator computes another mapping (called *translation*) that maps all possible garbled outputs to their actual values. In the final round, the generator sends the translation to the evaluator, and the evaluator

[4] The term "garbled circuits" is due to Beaver, Micali, and Rogaway [BMR90].

sends the garbled output to the generator. Both parties can compute the actual output at the same time without learning anything about each other's inputs. This algorithm is only secure in the semi-honest setting.

Yao's original model has been the basis for several secure computation algorithms mostly for the two-party setting with computational hardness assumptions [GMW87,LP07,HEKM11,LP11,KMR11]. In a line of research, Lindell and Pinkas give the first proof of Yao's protocol [LP09] and present a two-party approach based on garbled circuits that uses the cut-and-choose technique to deal with malicious parties [LP07,LP11].

Secret Sharing. In secret sharing, one party (called the *dealer*) distributes a secret amongst a group of parties, each of whom is allocated a *share* of the secret. Each share reveals nothing about the secret to the party possessing it, and the secret can only be reconstructed when a sufficient number of shares are combined together.

Many MPC schemes build upon the notion of secret sharing (most notably, [BGW88,CCD88,Bea91,GRR98,DIK$^+$08,DPSZ12,DKMS14]). Informally speaking, each party secret shares its input among all parties using a secret sharing scheme such as Shamir's scheme [Sha79]. Then, all parties perform some intermediate computation on the received shares and ensure that each part now has a share of the result of the computation. In the final stage, all parties perform a final computation on the intermediate results to find the final result. In the Byzantine setting, each stage of this approach usually requires several rounds of communication used to verify consistency of the shares distributed by each party (using a *Verifiable Secret Sharing (VSS)* scheme like [CGMA85,BGW88]) and to perform complicated operations such as multiplication.

Ben-Or *et al.* [BGW88] show that every functionality can be computed with information-theoretic security in the presence of a semi-honest adversary controlling less than half of the parties, and in the presence of a Byzantine adversary controlling less than a third of the parties. They propose a protocol for securely evaluating an arithmetic circuit that represents the functionality. First, the parties secret-share their inputs with each other using Shamir's scheme [Sha79]. For the Byzantine case, an interactive VSS protocol is proposed using bivariate polynomials. The parties emulate the computation of each gate of the circuit by computing shares of the gate's output from the shares of the gate's inputs.

Given shares of the input wires, an addition gate's output is computed without any interaction simply by asking each party to add their local shares together. Unfortunately, multiplying two polynomials results in a polynomial that has a higher degree and is not completely random. Ben-Or *et al.* [BGW88] emulate a multiplication gate computation by running interactive protocols for degree reduction and polynomial randomization.

The efficiency of [BGW88] was later improved by others in similar and different settings [Bea91,GRR98,DIK$^+$08]. Unfortunately, these protocols still do not scale well with n and incur large communication and computation costs in practice.

Dani *et al.* [DKMS12] propose an MPC protocol for evaluating arithmetic circuits in large networks. The protocol is unconditionally-secure against a Byzantine adversary corrupting less than $(1/3 - \epsilon)n$ of the parties, for some positive constant ϵ. The protocol creates a set of quorums using the quorum building algorithm of [KLST11]. For each gate in the circuit, a quorum is used to compute the output of the gate using the MPC of [BGW88] among parties of the quorum. The protocol ensures that all parties in the quorum learn the output of gate masked with a uniformly random value which is secret-shared among all parties of the quorum. 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 for every level of the circuit. At the top level, the output is computed and sent down to all parties through all-to-all communication between the quorums. Assuming a circuit of depth d with m gates, this protocol requires each party to send (and compute) $\tilde{O}(m/n + \sqrt{n})$ bits (operations) with latency $O(d + \mathsf{polylog}(n))$.

This protocol was later modified in [DKMS14] in order to support asynchronous communication incurring the same asymptotic costs but tolerating less than $(1/8 - \epsilon)n$ malicious parties. In the new model, the adversary has control over the latency of the communication channels and can arbitrarily delay messages sent over them. However, all messages sent by the parties are assumed to be eventually delivered but with indefinite delays.

The main challenge in this model is that the parties require a distributed mechanism to learn when sufficient number of inputs are received in order to start the computation over those inputs. To this end, Dani *et al.* [DKMS14] propose to count the number of *ready inputs* using a distributed data structure called τ-*counter*, where $\tau = n - t$ is the threshold on the number inputs to be received before the circuit is evaluated, and $t < n/8$.

Fully Homomorphic Encryption. A *fully homomorphic encryption (FHE)* scheme allows to perform secure computation over encrypted data without decrypting it. Gentry [Gen09] proposed the first FHE scheme based on the hardness of lattice problems. Since then, many techniques have been proposed to improve the efficiency of FHE [vDGHV10,BGV12,GHS12]. Unfortunately, current techniques are still very slow and can only evaluate small circuits. This restriction is primarily due to noise management techniques (such as bootstrapping [Gen09]) used to deal with a noise term in ciphertexts that increases slightly with homomorphic addition and exponentially with homomorphic multiplication.

In particular, if the circuit has a sufficiently small multiplicative depth, then it is possible to use current FHE schemes in practice without using the expensive noise management techniques. Such a scheme is often called *somewhat homomorphic encryption (SHE)* [vDGHV10], which requires significantly less amount of computation than an FHE with noise management.

Damgard *et al.* [DPSZ12] propose a Byzantine-resilient MPC scheme using SHE in an offline phase to compute Beaver multiplication triples [Bea91]. These triples are later used in the online phase to compute multiplication gates efficiently. One drawback of this scheme is that when cheating happens in the network, the protocol cannot guarantee termination. Malicious parties can take advantage of this to prevent the protocol from termination.[5]

Asharov *et al.* [AJLA+12] describe a constant-round MPC scheme using a *threshold FHE (TFHE)* technique that provides Byzantine-resilience and circuit-independent communication cost. All parties first encrypt their inputs under the FHE scheme of Brakerski *et al.* [BGV12] and send the encrypted values to all other parties. Then, each party evaluates the desired function over the encrypted inputs via homomorphism, and eventually participates in a distributed decryption protocol to decrypt the output. Although providing constant rounds of communication, this scheme does not scale well with the number of parties and the circuit size due to all-to-all communication overhead (*i.e.*, $\Omega(n^2)$) and high computation overhead of the FHE of [BGV12] for large-depth circuits. To overcome the high computation cost, the authors propose to outsource circuit computation to a powerful party (*e.g.*, the "cloud"). While this is helpful for the semi-honest setting, it requires expensive zero-knowledge proofs to enforce honest behavior in the Byzantine setting.

Boyle *et al.* [BGT13] describe a synchronous MPC protocol for evaluating arithmetic circuits. The protocol is computationally-secure against an adversary corrupting up to $(1/3 - \epsilon)$ fraction of parties, for some fixed positive ϵ. As network size scales, it becomes infeasible to require each party communicate with all other parties. To this end, the protocol of [BGT13] uses quorums to achieve sublinear ($\mathsf{polylog}(n)$) *communication locality* which is defined as the total number of point-to-point communication channels that each party uses in the protocol. Interestingly, the communication costs are independent of circuit size. This is achieved by evaluating the circuit over encrypted values using an FHE 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.

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

2.2 RAM-Based Techniques

Most MPC constructions model algorithms as circuits. Unfortunately, a circuit can at best model the *worst-case* running time of an algorithm because a circuit

[5] In general, if the majority of parties are malicious, then the termination of MPC (*i.e.*, output delivery) cannot be guaranteed.

can only be created by unrolling loops to their worst-case runtime [GKP+13]. Moreover, circuits incur at least a linear computation complexity in the total size of the input, while a sublinear overhead is crucial for achieving scalability in most large-scale applications. In addition, most algorithms have already been described in terms of instructions (programs) to a *random access memory (RAM)* machine[6] [CR72], not circuits. These all bring the following question to the mind: Is it possible to securely evaluate RAM programs instead of circuits? Luckily, the answer is "yes". Goldreich and Ostrovsky [GO96] show that by constructing a RAM with secure access, one can evaluate arbitrary RAM programs privately. Such a RAM is often called an *Oblivious RAM (ORAM)*. This is typically considered in a setting, where a group of parties (clients) want to access a data storage (RAM) held by another party (a server).

To build an ORAM, content encryption alone is not sufficient because the party holding the data (or an eavesdropper) can obtain critical information about the queries by analyzing the access patterns, even though the data is encrypted. Therefore, techniques are required to hide the access patterns to the data storage, meaning that no party is able to distinguish between any subsets of the data requests. More precisely, the following information must remain private: (1) the locations of accessed data items, (2) the order of data requests, (3) the number of requests to the same location, and (4) the type of access (*e.g.*, get-value, set-value).

Goldreich and Ostrovsky [GO96] propose a two-party technique for securely outsourcing data to a remote storage. The client's access pattern to the remote storage is hidden by continuously shuffling and re-encrypting data as they are accessed. The authors show that any program in the standard RAM model can be compiled into a program for an ORAM using an *ORAM simulator* with an overhead that is polylogarithmic in the size of the memory. Although asymptotically-efficient, the algorithm of [GO96] is not practical due to large constant factors.

Several techniques have been proposed to improve the overhead of ORAM protocols in general [PR10,SCSL11,SvDS+13] and for secure two-party computation [GKK+12,GGH+13,LO13]. Damgard *et al.* [DMN11] propose the first ORAM algorithm in the multi-party setting. Unfortunately, their algorithm requires each party to communicate and maintain information of size equivalent to all parties' inputs. Boyle *et al.* [BCP14a] describe a scalable technique for secure computation of RAM programs in large networks by performing local communications in quorums of parties. For securely evaluating a RAM program Π, their protocol incurs a total communication and computation of $\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$ while requiring $\tilde{O}(|x| + Space(\Pi)/n)$ memory per party, where $Time(\Pi)$ and $Space(\Pi)$ are time and space complexity of Π respectively, and $|x|$ denotes the input size.

[6] A RAM machine has a *lookup functionality* for accessing memory locations that takes $O(1)$ operations. Given an array A of N values and an index $x \in \{1, ..., N\}$, the lookup functionality returns $A[x]$.

3 Open Problems

In this section, we describe several open problems in the domain of scalable MPC. These problems are roughly ordered from easiest to hardest. We describe some partial progress on the first two problems later in this paper.

Share Renewal. In the protocol of [DKMS14], 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. Inspired by [HJKY95], we refer to this problem as *share renewal*, because it involves generating a fresh sharing of a secret-shared value among a new set of parties.

Dani *et al.* [DKMS14] handle this problem by masking the result in Q and unmasking the result in Q'. Unfortunately, they do not provide an explicit construction of their method, and simple constructions are very expensive in terms of communication and computation costs [ZMS14]. Boyle *et al.* [BGT13] overcome this problem by sending their encrypted inputs to only one quorum which does all of the computation using FHE. This results in large computation and communication costs for parties in that quorum. In Section 5, we give some ideas for solving this problem efficiently.

Secure Multiplication. Consider an arithmetic circuit representing the desired function to be computed. Using a linear secret sharing scheme (such as [Sha79]), addition gates can be computed with no communication by simply adding the two input shares. On the other hand, known secret sharing schemes are not *multiplicatively homomorphic* meaning that the product of two shares is not necessarily a valid and secure share of the product of the corresponding secrets. Designing an efficient technique for secure multiplication is an important building block for secret-sharing-based MPC. We are not aware of a perfectly-secure technique for secure multiplication that requires only constant rounds of communication.

Byzantine-Resilient Threshold Decryption. Consider n parties that have jointly encrypted a message using some encryption scheme. In threshold decryption, for some parameter $x < n$, it is required that any subset of x parties can decrypt the message, while any subset of strictly less than x parties learn nothing about the encrypted message. Threshold decryption of a (fully) homomorphic encryption can be used as a primitive for constructing efficient MPC protocols.

Unfortunately, known techniques for Byzantine-resilient threshold decryption (such as [CDN01,AJLA$^+$12]) suffer from large communication overhead, due to zero-knowledge proofs used for ensuring honest behavior. A key open problem is to reduce this communication overhead.

Las Vegas MPC. Recently, several randomized MPC algorithms have been proposed (such as [BGT13,DKMS14,CCG$^+$14]) with Monte Carlo guarantees.

In particular, the output is correct with high probability[7]. Alternatively, one may try to design a Las Vegas MPC algorithm. For this type of algorithm, the output must be correct with probability 1, but the latency can be a random variable. It is not clear that a quorum-based approach will be effective for solving this open problem.

Oblivious Parallel RAM. While parallelism has been extensively used in various computer architectures for accelerating computations, most ORAM models of computation (see Section 2.2) are not parallel. In the *Parallel RAM (PRAM)* model of computation, several parties, running in parallel, want to access the same shared external memory. This separates the program into two parts: control flow and shared memory. The goal is to parallelize control flow via oblivious access to the share memory in order to reduce computational time and latency.

In general, any PRAM program can be converted into a RAM program to be then evaluated by a standard ORAM. Unfortunately, this transformation incurs a large computational overhead. Boyle *et al.* [BCP14b] take a first step towards addressing this problem by describing an oblivious PRAM scheme that compiles any PRAM program into an *Oblivious PRAM (OPRAM)* program. This compiler incurs polylogarithmic overhead in the number of parties and the memory size. The algorithm is based on the ORAM construction of Shi *et al.* [SCSL11] which requires cryptographic assumptions. It remains unknown if one can design a perfectly-secure oblivious PRAM with resource costs that scale well with the network size and are independent of memory size.

Large Inputs. It is also interesting to consider MPC when each party can have very large inputs. Following is a concrete problem in this domain. Let M be a sparse adjacency matrix for some graph, and let the columns of M be partitioned among the parties. This problem motivates the following questions: Can we securely and efficiently compute the shortest path between a given pair of vertices over M? We can also consider other graph-theoretic problems. Even simpler, can we securely compute the dot product of two sparse vectors with resource costs proportional only to the number of non-zero entries in the vectors? We are not aware of any algorithms for even these simple types of problems.

4 Algorithmic Tools

In this section, we describe key algorithmic tools used in scalable MPC protocols.

4.1 Verifiable Secret Sharing

An (n, t)-*secret sharing* scheme, is a protocol in which a dealer who holds a secret value shares the secret among n parties such that any set of t parties cannot gain any information about the secret, but any set of at least $t + 1$ parties

[7] 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.

can reconstructs it. An (n, t)-*verifiable secret sharing (VSS)* scheme is an (n, t)-secret sharing scheme with the additional property that after the sharing phase, a dishonest dealer is either disqualified or the honest parties can reconstruct the secret, even if shares sent by dishonest parties are spurious. Katz *et al.* [KKK08] propose a constant-round VSS protocol based on Shamir's secret sharing [Sha79]. This result is described in Theorem 1.

Theorem 1. [KKK08] *There exists a synchronous linear (n, t)-VSS scheme for $t < n/3$ that is perfectly-secure against a static adversary. The protocol requires one broadcast and three rounds of communication.*

For practicality, one can use the cryptographic VSS of Kate *et al.* [KZG10] called *eVSS*[8], which is based on Shamir's scheme and the hardness of the discrete logarithm problem. Since eVSS generates commitments over elliptic curve groups, it requires smaller message sizes than other DL-based VSS scheme such as [GRR98].

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

4.2 Secure Broadcast

In the Byzantine 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 dishonest and sends different messages to different parties. It is known that a BA protocol can be used to perform secure broadcasts. The BA algorithm of Braud-Santoni *et al.* [BGH13] gives the following result.

Theorem 3. [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 $(1/3 - \epsilon)n$ Byzantine parties, for any positive ϵ.*

The algorithm of [BGH13] achieves this result by relaxing the load-balancing requirements. If concerned with load-balancing, one can instead use the load-balanced BA algorithm of King *et al.* [KSSV06b] with $O(\sqrt{n})$ blowup.

4.3 Quorum Building

As an intermediate result in a Byzantine agreement paper, King *et al.* [KLST11] give a protocol that can be used to bring all parties to agreement on a collection of n quorums. This protocol is based on the almost-everywhere Byzantine agreement[9] of King *et al.* [KSSV06b].

[8] Stands for efficient VSS

[9] King *et al.* [KSSV06b] relax the requirement that all uncorrupted parties reach agreement at the end of the protocol, instead requiring that a $1 - o(1)$ fraction

Theorem 4. [KSSV06b] *Suppose there are n parties, $b < 1/4 - \epsilon$ of which are malicious, for any fixed positive ϵ. Then, there is a polylogarithmic (in n) bounded degree network and a protocol such that,*

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

Their main technique is to divide the parties into groups of polylogarithmic size; each party is assigned to multiple groups. In parallel, each group then uses a leader election algorithm [Fei99] to *elect* a small number of parties from within their group to move on. This step is recursively repeated on the set of elected parties until size of the remaining parties in this set becomes polylogarithmic. At this point, the remaining parties can solve the *semi-random-string agreement problem*[10] (similarly, they can run Byzantine agreement protocol to agree on a bit). Provided the fraction of dishonest parties in the set of remaining parties is less than 1/3 with high probability, these parties succeed in agreeing on a semi-random string. Then, these parties communicate the result value to the rest of the parties.

In general, one can use any BA algorithm to build a set of quorums as described in [KLST11]. Theorem 5 gives a quorum building protocol using the BA algorithm of Braud-Santoni *et al.* [BGH13].

Theorem 5. [BGH13] *There exists an unconditionally-secure protocol that brings all good parties to agreement on n good quorums with high probability. The protocol has $\tilde{O}(n)$ amortized communication and computation complexity*[11]*, and it can tolerate up to $(1/3 - \epsilon)n$ Byzantine parties, for any positive ϵ. If at most $(\alpha - \epsilon)$ fraction of parties are Byzantine, then each quorum is guaranteed to have $T \leq \alpha N$ Byzantine parties.*

5 Quorum-Based MPC

In Table 1, we review recent MPC results that provide sublinear communication locality. All of these results rely on some quorum building technique for creating a set of quorums each with honest majority. In the rest of this section, we describe a few ideas for improving efficiency of the synchronous MPC of [DKMS12]. These techniques are proved in [ZMS14]. We also conduct experiments to show the effectiveness of our techniques when compared with the protocols of [DKMS12] and [BGT13].

of uncorrupted parties reach agreement. They refer to this relaxation as almost-everywhere agreement.

[10] In *semi-random-string agreement problem*, we want to reach a situation where, for any positive constant ϵ, $1/2 + \epsilon$ fraction of parties are good and agree on a single string of length $O(\log n)$ with a constant fraction for random bits.

[11] Amortized communication complexity is the total number of bits exchanged divided by the number of parties.

Table 1. Recent MPC results with sublinear communication locality.

Protocol	Security	Resiliency Bound	Adaptive Adversary?	Async?	Assumes Broadcast Channel?	Total Message Complexity	Total Computation Complexity	Latency	Msg Size	Load-Balanced?
[DKMS12]	Perfect	$(1/3-\epsilon)n$	No	No	No	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d+\mathsf{polylog}(n))$	$O(\ell)$	Yes
[BGT13]	Crypto	$(1/3-\epsilon)n$	No	No	No	$\tilde{O}(n)$	$\tilde{\Omega}(n)+\tilde{\Omega}(\kappa m d^3)^{\dagger}$	$\tilde{O}(1)$	$O(n\ell\cdot\mathsf{polylog}(n))$	No
[DKMS14]	Perfect	$(1/8-\epsilon)n$	No	Yes	No	$\tilde{O}(m+n\sqrt{n})$	$\tilde{O}(m+n\sqrt{n})$	$O(d+\mathsf{polylog}(n))$	$O(\ell)$	Yes
[ZMS14]	Perfect	$(1/6-\epsilon)n$	No	No	No	$\tilde{O}(m)$ or $O(m\log^4 n)^{\ddagger}$	$\tilde{O}(m)$ or $O(m\log^5 n)^{\ddagger}$	$O(d)$	$O(\ell)$ or $O(\kappa+\ell)^{\ddagger}$	Yes
[BCP14a]	Perfect	$(1/3-\epsilon)n$	No	No	Yes	$\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$	$\mathsf{poly}(n)+\tilde{O}(Time(\Pi))$	$\tilde{O}(Time(\Pi))$	$O(\ell)$	Yes
[CCG+14]	Crypto	$n/2$	Yes	No	No	$O(n\log^{1+\epsilon} n)^{\S}$ or $O(n\sqrt{n}\log^{1+\epsilon} n)$	$\Omega(n\log^{1+\epsilon} n)^{\S}$ or $\Omega(n\sqrt{n}\log^{1+\epsilon} n)$	$O(\log^{\epsilon'} n)$	$\Omega(\log^{\log n} n)^{\S}$ or $\Omega(\sqrt{n}^{\log n})$	Yes

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

Notes.
†Using the FHE scheme of [BGV12].
‡The latter is achieved using cryptographic assumptions.
§Assuming a symmetric-key infrastructure.

Although the protocol of [DKMS12] asymptotically scales well with n, it is inefficient in practice due to large hidden factors in its cost complexities. Moreover, the paper does not provide an explicit construction of the proposed share renewal technique (see Section 3). Unfortunately, simple constructions seem very expensive in terms of communication and computation costs since parties in the second quorum need to jointly and securely unmask each input.

In [ZMS14], we propose a simple and efficient technique for share renewal. We also implement an efficient secure multiplication protocol to reduce the bandwidth and computation costs of [DKMS12]. In the rest of this section, we sketch the main properties of this new algorithm. Full details of the algorithms are in [ZMS14].

Consider n parties in a fully-connected synchronous network with private channels. Let f be any deterministic function over n inputs in some finite field $\mathbb{F}_p$ for prime $p = \mathsf{poly}(n)$, where f is represented as an arithmetic circuit with m gates and depth d. We prove the following theorem in [ZMS14].

Theorem 6. *There exists an unconditionally-secure n-party protocol that can compute f with high probability, while ensuring,*

- *The protocol tolerates up to $(1/6 - \epsilon)n$ malicious parties.*
- *Each party sends (computes) $\tilde{O}(m/n)$ messages (operations).*
- *Each message is of size $O(\log p)$ bits.*
- *The protocol has $O(d)$ rounds of communication.*

While our techniques are perfectly-secure, one can use the efficient VSS of Theorem 2 to achieve a cryptographic variant of Theorem 6 with better efficiency.

Theorem 7. *There exists an n-party protocol secure against a PPT adversary such that the protocol can compute f with high probability, while ensuring,*

- *The protocol tolerates up to $(1/6 - \epsilon)n$ malicious parties.*
- *Each party sends $O\big(\frac{m}{n}\log^4 n\big)$ messages and computes $O\big(\frac{m}{n}(\kappa+\log p)\log^4 n\big)$ operations, where κ is the security parameter.*
- *Each message is of size $O(\kappa + \log p)$ bits.*
- *The protocol has $O(d)$ rounds of communication.*

In this section, we represent each shared value $s \in \mathbb{F}_p$ by $\langle s \rangle = (s_1, ..., s_n)$ meaning that each party P_i holds a share s_i generated by the VSS scheme of Theorem 1 during its sharing phase. Using the natural component-wise addition of representations, we define $\langle a \rangle + \langle b \rangle = \langle a + b \rangle$. For multiplication, we define $\langle a \rangle \cdot \langle b \rangle = \mathsf{Multiply}(\langle a \rangle, \langle b \rangle)$, where $\mathsf{Multiply}$ is a protocol defined later in this section.

5.1 Share Renewal

Recalling from Section 3, let Q and Q' be the quorums involved in the share renewal process. In [ZMS14], we propose a different approach than [DKMS12] by reducing the share renewal problem to the simpler problem of generating a fresh sharing of the output of Q among parties of Q'. In other words, parties in Q generate a new set of random shares that represents the same secret as the output of Q, and distribute this new sharing among parties of Q'.

The high-level idea is to first generate a random polynomial that passes through the origin, and then add it to the polynomial that corresponds to the shared secret. The result is a new polynomial that represents the same secret but has coefficients that are chosen randomly and completely independent of the coefficients of the original polynomial. Combined with the VSS scheme of Theorem 1 in a group of n parties with $t < n/3$ dishonest parties, this protocol has one round of communication and requires each party to send $O(n)$ field elements.

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

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

Theorem 8. [ZMS14] *Let Q and Q' be two quorums of size N, where Q holds a shared value $\langle s \rangle = (s_1, ..., s_N)$ over a polynomial ϕ of degree $d = N/3$. There exists a protocol that can generate a new shared value $\langle s' \rangle = (s'_1, ..., s'_N)$ in Q' such that $s' = s$. The protocol is secure against a computationally-unbounded Byzantine adversary corrupting less than a $1/6$ fraction of the parties in each quorum.*

5.2 Secure Multiplication

The secure multiplication protocol of [ZMS14] (denoted by $\mathsf{Multiply}$) is based on a well-known technique proposed by Beaver [Bea91]. The technique generates a shared multiplication triple $(\langle u \rangle, \langle v \rangle, \langle w \rangle)$ such that $w = u \cdot v$. The triple is then used to convert multiplications over shared values to additions.

The only difference between $\mathsf{Multiply}$ and Beaver's multiplication method is that Beaver generates shared random elements u and v on polynomials of degree d and multiplies them to get a polynomial of degree $2d$ for w. Then, a degree reduction algorithm is run to reduce the degree from $2d$ to d. Instead, we choose

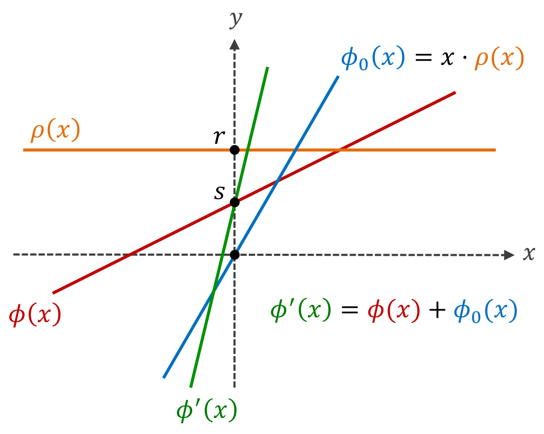
y
$\phi_0(x) = x \cdot \rho(x)$
$\rho(x)$
r
s
x
$\phi(x)$
$\phi'(x) = \phi(x) + \phi_0(x)$
$\phi'(x)$

Fig. 1. Share renewal technique [ZMS14]

polynomials of degree $d/2$ for u and v to get a polynomial of degree d for w. In our protocol, since we require less of 1/6 fraction of the parties be dishonest in each quorum, we can do this without revealing any information to the adversary. We note that the first step of Multiply is independent of the inputs and thus, can be performed in an offline phase to generate a sufficient number of multiplication triples.

Theorem 9. [ZMS14] *Given two secret-shared values $\langle a \rangle$ and $\langle b \rangle$, the protocol Multiply correctly generates a shared value $\langle c \rangle$ such that $c = a \cdot b$. The protocol is perfectly-secure against an adversary corrupting less than a 1/6 fraction of the parties.*

5.3 Simulation Results

To study the effectiveness of our techniques and compare our new MPC scheme to previous work, we simulate our protocol along with the protocols of Dani *et al.* [DKMS12] and Boyle *et al.* [BGT13]. We use these protocols to solve the *secure multi-party sorting (MPS)* problem. MPS is useful in many applications such as anonymous communication [RS93,BFTS04,MSZ14], privacy-preserving statistical analysis [DA01] (*e.g.*, top-k queries [BD10]), auctions [Zha11], and location-based services [ZM14]. 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).

We run our protocol for inputs chosen from $\mathbb{Z}_p$ with a 160-bit prime p for getting about 80 bits of security. We set the parameters of our protocol 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 $\epsilon < 10^{-8}$ 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 $\epsilon < 10^{-300}$. For all protocols evaluated in this section, we assume cheating (by malicious parties) happens in every round of the protocols. This is essential for evaluating various strategies used by these protocols for tolerating active attacks.

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 better compare the protocols, the vertical and horizontal axis of the plot are scaled logarithmically. The x-axis presents the number of parties and the y-axis presents the number of Kilobytes sent by each party for delivering one anonymous bit.

In this figure, we report results from three different versions of our protocol. The first plot (marked with circles) belongs to our unconditionally-secure protocol (Theorem 6) that uses the perfectly-secure VSS scheme of Katz *et al.* [KKK08]. The second plot (marked with stars) represents our computationally-secure protocol (Theorem 7) which uses the cryptographic VSS of Kate *et al.* [KZG10]. The last plot (marked with diamonds) shows the cost of the cryptographic protocol with amortized (averaged) setup cost.

We obtain this by running the setup phase of our protocol once and then using 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 communication cost. To achieve better results, we also generated a sufficient number of random triples in the setup phase. Then, the triples were used by our multiplication subprotocol in the online phase to multiply secret-shared values efficiently.

Our protocols significantly reduce bandwidth costs when compared to the protocols of [DKMS12] and [BGT13]. For example, for $n = 2^{20}$ (about 1 million parties[12]), the amortized protocol requires each party to send about 64KB of data per anonymous bit delivered (about 8MB for our non-amortized version) while the protocols of [DKMS12] and [BGT13] each send more than one Terabytes of data per party and per sorting bit delivered.

6 Conclusion

We described recent MPC algorithms that are efficient, even with many parties. In particular, we reviewed the most important results that achieve scalability via quorums. To draw distinctions between various schemes, we described different approaches used in the literature for solving MPC. We described six open problems whose solutions would improve efficiency of scalable MPC schemes. Additionally, we described constructive techniques to improve efficiency of current quorum-based techniques. A drawback of most MPC results is the lack of empirical studies. We addressed this by implementing and benchmarking a number of recent methods as well as our new techniques.

[12] This is less than 1% of the number of active Twitter users. An intriguing application of our protocol is an anonymous version of Twitter.

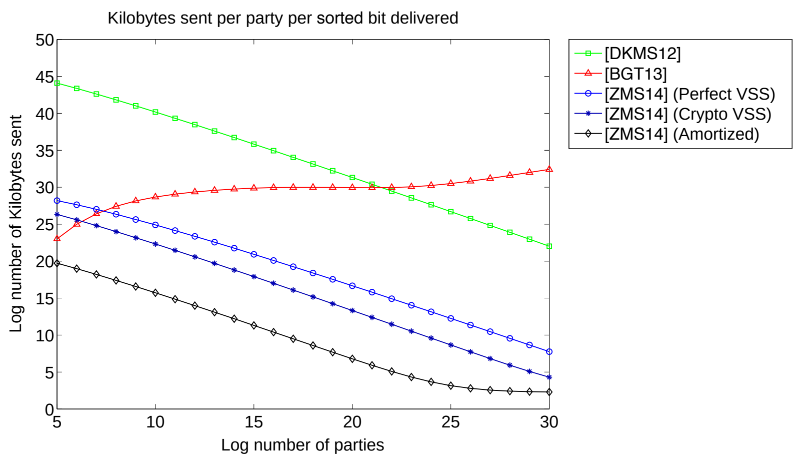

Fig. 2. Bandwidth cost of scalable MPC protocols

7 Acknowledgments

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 by Mahnush Movahedi from University of New Mexico, David Evans from University of Virginia, Elette Boyle from Cornell University, and Aniket Kate from Saarland University.

References

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

BCG93. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth ACM Symposium on the Theory of Computing (STOC)*, 1993.

BCP14a. Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation. Cryptology ePrint Archive, Report 2014/404, 2014.

BCP14b. Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram. Cryptology ePrint Archive, Report 2014/594, 2014.

BD10. 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, August 2010.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 1991.

BFTS04. Ron Berman, Amos Fiat, and Amnon Ta-Shma. Provable unlinkability against traffic analysis. In *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2004.

BGH13. Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast Byzantine agreement. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 57–64, New York, NY, USA, 2013. ACM.

BGT13. Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation: how to run sublinear algorithms in a distributed setting. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, TCC'13, pages 356–376, Berlin, Heidelberg, 2013. Springer-Verlag.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the Twentieth ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

BMR90. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

CCG$^+$14. Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. Optimally resilient and adaptively secure multi-party computation with low communication locality. Cryptology ePrint Archive, Report 2014/615, 2014.

CDG88. David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 87–119, London, UK, UK, 1988. Springer-Verlag.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 280–299, London, UK, UK, 2001. Springer-Verlag.

CFGN96. R. Canetti, U. Friege, O. Goldreich, and M. Naor. Adaptively secure multiparty computation. Technical report, Cambridge, MA, USA, 1996.

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, pages 383–395, Washington, DC, USA, 1985. IEEE Computer Society.

CR72. Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, New York, NY, USA, 1972. ACM.

DA01. Wenliang Du and Mikhail 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.

DI06. I. Damgård and Y. Ishai. Scalable secure multiparty computation. *Advances in Cryptology - CRYPTO 2006*, pages 501–520, 2006.

DIK$^+$08. I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology – CRYPTO '08*, pages 241–261, 2008.

DIK10. Ivan Damgrd, Yuval Ishai, and Mikkel Krigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer Berlin Heidelberg, 2010.

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. Full version: http://arxiv.org/abs/1203.0289.

DKMS14. Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2014.

DMN11. Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC'11, pages 144–163, Berlin, Heidelberg, 2011. Springer-Verlag.

DN07. I. Damgård and J.B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, pages 572–590. Springer-Verlag, 2007.

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

Fei99. Uriel Feige. Noncryptographic selection protocols. In *FOCS*, pages 142–153, 1999.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

GGH$^+$13. Craig Gentry, KennyA. Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation. In *Privacy Enhancing Technologies*, volume 7981 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2013.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

GHY88. Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 135–155, London, UK, UK, 1988. Springer-Verlag.

GKK$^+$12. S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 513–524, New York, NY, USA, 2012. ACM.

GKP$^+$13. Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In *Advances in Cryptology CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer Berlin Heidelberg, 2013.

GL02. Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *Distributed Computing*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg, 2002.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

GO96. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

GRR98. R. Gennaro, M.O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 101–111. ACM, 1998.

HEKM11. Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

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

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

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.

KMR11. Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011.

KSSV06a. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999, Philadelphia, PA, USA, 2006.

KSSV06b. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 87–98, Washington, DC, USA, 2006. IEEE Computer Society.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology – ASIACRYPT 2010 - 16th 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.

LO13. Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 377–396, Berlin, Heidelberg, 2013. Springer-Verlag.

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

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC'11, pages 329–346, Berlin, Heidelberg, 2011. Springer-Verlag.

MSZ14. Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Secure anonymous broadcast. *ArXiv e-prints*, May 2014.

PR10. Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 502–519, Berlin, Heidelberg, 2010. Springer-Verlag.

PSL80. M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

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.

SCSL11. Elaine Shi, T.-H.Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *Advances in Cryptology ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 197–214. Springer Berlin Heidelberg, 2011.

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

SvDS$^+$13. Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

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

Yao82. Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

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.

ZM14. Mahdi Zamani and Mahnush Movahedi. Secure location sharing. In *Proceedings of the 10th ACM International Workshop on Foundations of Mobile Computing*, FOMC '14, pages 1–10, New York, NY, USA, 2014. ACM.

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