Breaking the $O(nm)$ Bit Barrier: Secure Multiparty Computation with a Static Adversary

Varsha Dani
University of New Mexico

Valerie King
University of Victoria

Mahnush Movahedi
University of New Mexico

Jared Saia
University of New Mexico

Abstract

We describe scalable algorithms for secure multiparty computation (SMPC). We assume a synchronous message passing communication model, but unlike most related work, we do not assume the existence of a broadcast channel. Our main result holds for the case where there are n players, of which a $1/3 - \epsilon$ fraction are controlled by an adversary, for ϵ any positive constant. We describe a SMPC algorithm for this model that requires each player to send $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ messages and perform $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ computations to compute any function f, where m is the size of a circuit to compute f. We also consider a model where all players are selfish but rational. In this model, we describe a Nash equilibrium protocol that solve SMPC and requires each player to send $\tilde{O}(\frac{n+m}{n})$ messages and perform $\tilde{O}(\frac{n+m}{n})$ computations. These results significantly improve over past results for SMPC which require each player to send a number of bits and perform a number of computations that is $\theta(nm)$.

1 Introduction

In 1982, Andrew Yao posed a problem that has significantly impacted the weltanschauung of computer security research [22]. Two millionaires want to determine who is wealthiest; however, neither wants to reveal any additional information about their wealth. Can we design a protocol to allow both millionaires to determine who is wealthiest?

This problem is an example of the celebrated *secure multiparty computation (SMPC)* problem. In this problem, n players each have a private input, and their goal is to compute the value of a n-ary function, f, over its inputs, without revealing any information about the inputs. The problem is complicated by the fact that a hidden subset of the players are controlled by an adversary that actively tries to subvert this goal.

SMPC abstracts numerous important problems in distributed security, and so, not surprisingly, there have been thousands of papers written in the last several decades addressing this problem. However, there is a striking barrier that prevents wide-spread use: current algorithms to solve SMPC are not resource efficient. In particular, if there are n players involved in the computation and the function f can be computed by a circuit with m gates, then most algorithms require each player to send a number of messages and perform a number of computations that is $\Omega(mn)$ (see, for example, [11, 12, 5, 2, 15, 10, 16, 17, 3]).

Recent years have seen exciting improvements in the *amortized* cost of SMPC, where the number of messages and total computation done per player can be significantly better than $\Theta(mn)$ [7, 9, 8]. However, the results for these algorithm hold only in the amortized case where m is much larger than n, and all of them have additional additive terms that are large polynomials in n (e.g. n^6). Thus, there is still a strong need for SMPC algorithms that are efficient in both n and m.

1.1 Formal Problem Statement

We now formally define the SMPC problem. As previously stated, there are n players, and each player i has a private input, x_i. Further there is a n-ary function f that is known to all players. The goal is to to ensure that: 1) all players learn the value $f(x_1, x_2, \ldots, x_n)$; and 2) the inputs remain as private as possible: each player i learns nothing about the private inputs other than what is revealed by $f(x_1, x_2, \ldots, x_n)$ and x_i.

The main complication is the fact that up to a 1/3 fraction of the players are assumed to be controlled by an adversary that is actively trying to prevent the computation of the function. We will say that the players controlled by the adversary are *bad* and that the remaining players are *good*. The adversary is *static*, meaning that it must select the set of bad players at the start of the algorithm. A careful reader may ask: How can we even define the problem if the bad players control their own inputs to the function and thereby can exert control over the output of f?

The answer to this question is given by Figure 1. In the left illustration in this figure, there are 5 players that are trying to compute a function over their 5 private inputs. If there is a trusted external party, as shown in the center of this illustration, the problem would be easy: each player sends their input to this trusted party, the party performs the computation, and then sends the output of f back to all the players. In essence, this is the situation we want to simulate with our SMPC algorithm. The right illustration of Figure 1 shows this goal: the SMPC algorithm simulates the trusted party. In particular, we allow all players, both good and bad, to submit a single input to the SMPC algorithm. The SMPC algorithm then computes the function f based on all of these submitted inputs, and sends the output of f back to all players.

This problem formulation is quite powerful. If f returns the input that is in the majority, then

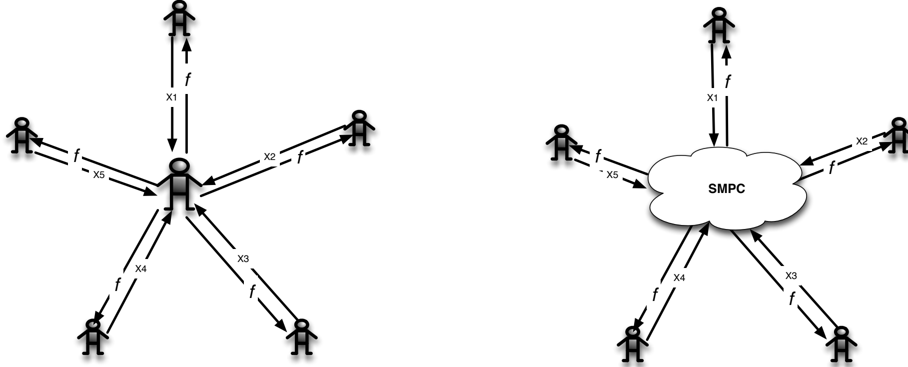

Figure 1: Schematic of SMPC problem

SMPC enables voting. If f returns a tuple containing 1) the index of the highest variable; and 2) the value of the second highest variable, then SMPC enables a simple Vickrey auction, i.e. the highest bidder wins and pays the second highest bid. If f returns the output of a digital signing function, where the private key equals the sum of all player inputs modulo Z_p for some prime p, then SMPC enables group digital signatures, i.e. the entire group can sign a document, but no individual player learns the secret key. In short, the only limitation is determined by whether or not the function f is computable.

Our communication model is as follows. We assume there is a private and authenticated communication channel between ever pair of players. However, we assume that the adversary is computationally unbounded, and so make no cryptographic hardness assumptions.[1] Also, unlike much past work on SMPC, we do *not* assume the existence of a public broadcast channel. Finally, we note that we assume that the good players strictly follow the protocol, and thus do not form coalitions in which information is exchanged (i.e. there are no so called "gossiping" good players).[2]

1.2 Our Results

The main result of this paper is as follows.

Theorem 1.1. Assume there are n players, no more than a 1/3 fraction of which are bad, and a n-ary function f that can be computed using m gates. Then if the good players follow Algorithm 1, with high probability, they can solve SMPC, while ensuring:

1. Each player sends at most $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ messages,
2. Each player performs $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ computations.

An additional result of this paper deals with the situation where all players are selfish but rational. Our precise assumption for the rational players is as follows. The rational players' utility functions are such that they prefer to learn the output of the function, but also prefer that other players not learn the output. Following previous work on rational secret sharing [19, 13, 14, 20], we assume all the players have the same utility function, which is specified by constants U_k where

[1] In the cryptographic community, this is frequently called *unconditional security.*

[2] Technically, we can maintain privacy, even with a certain amount of information exchange among the good players. See Section 3 for details.

k is the number of players who learn the output. Here U_1 is the utility to a player if she alone learns the output, U_n is the utility if she learns the secret and all other players learn it as well, and finally U_- is the utility when the player does not learn the output. We further assume that $U_1 > U_2 > \ldots > U_n > U_-$, so that the players' preferences are strict.

A key goal is to design a protocol that is a *Nash equilibrium* in the sense that no player can improve their utility by deviating from the protocol, given that all other players are following the protocol. Our main result in this model is the following.

Theorem 1.2. Assume there are n players and each player is rational with utility function given as above. Then there exists a protocol (see Section 2.5) such that 1) it is a Nash equilibrium for all players to run this protocol and 2) when all players run the protocol then, with high probability, they solve SMPC, while ensuring:

1. Each player sends at most $\tilde{O}(\frac{n+m}{n})$ messages,
2. Each player performs $\tilde{O}(\frac{n+m}{n})$ computations.

The rest of this paper is organized as follows. In Section 2, we describe our algorithms for scalable SMPC. We first present the case with an adversary, and then in Section 2.5, describe the changes that are needed to handle the case where all players are rational. The proofs of correctness for our algorithms are in Section 3. We conclude and give problems for future work in Section 4.

2 Our Algorithm

We now describe the algorithm that achieves our result for the model where players are either good or bad (Theorem 1.1).

The main idea behind reducing the amount of communication required for the computation is that rather than having each player communicate with all the other players, we will subdivide the players into groups called *quorums* of logarithmic size. The players in each group will communicate only with members of their own group and members of certain other groups. The number of other groups a particular group is required to communicate with is a function of the circuit size.

If any single quorum were to have too many bad players then they could severely disrupt the computation, so the subdivision must spread the bad players around so that the fraction of bad players in each quorum remains less than 1/3. We will call a quorum *good* if more than two thirds of the players in it are good. How are these quorums to be formed? This would be easy to achieve (with high probability) if there were a trusted mediator who could form the groups randomly and assign each player to their group. In the absence of a mediator, the players must achieve this subdivision themselves. To do this, we appeal to the following result of King, Lonergan, Saia and Trehan [18].

Theorem 2.1 (Theorem 2 of [18])**.** Let n be the number of processors in a fully asynchronous full information message passing model with a static adversary. Assume that there are at least $(2/3 + \epsilon)n$ good processors. Then for any positive constant ϵ, there exists a protocol which w.h.p. brings all good processors to agreement on n good quorums; runs in polylogarithmic time; and uses $\tilde{O}(\sqrt{n})$ bits of communication per processor. If all players are rational the algorithm runs in polylogarithmic time; and uses $\tilde{O}(1)$ bits of communication per processor.

We will also require certain primitives for multiparty protocols. A 1/3 fault-tolerant Verifiable Secret Sharing scheme for k players, henceforth VSS(k), is an algorithm for a dealer to deal shares of a secret which he holds to the players, such that (1) no set of fewer than a third of the players

can get any information about the secret and (2) the secret can be reconstructed from the shares even if upto a third of them are missing or corrupted (*i.e.* if upto a third of the players are bad) Moreover, the players then run a verification protocol, at the end of which the good players either agree that a valid secret has been shared or agree to disqualify the dealer (if he did not deal shares consistent with any secret) and take the secret to be a preset default value. Such a sharing and verification scheme is described in the work of Ben-Or, Goldwasser and Wigderson (BGW) [4]. This uses a constant number of rounds of communication and has zero probability of error. [3]

BGW [4] also describe an errorless protocol for SMPC that tolerates up to a third of the players being bad. (See BGW [4] Theorem 3). The number of rounds of communication depends at most linearly on the size of the circuit being computed.

We will make extensive black box use of both these primitives in our algorithm. Note that these protocols involve all-to-all communication amongst the players. For this reason we will refer to the SMPC primitive as HEAVYWEIGHT-SMPC. However, in our protocol, at most three quorums will be involved in any given run of the black box SMPC or VSS. Thus the amount of communication per run of HEAVYWEIGHT-SMPC will have only a polylogarithmic dependence on n.

We also note that the VSS and SMPC primitives require broadcast channels in addition to secure private channels. Within our quorums, we simulate broadcast channels using Byzantine Agreement to decide whether the same message was sent to everyone. Since the quorum size is just logarithmic, we can use any polynomial time and algorithm for Byzantine agreement, such as Bracha's protocol [6].

We assume that all computations are done over a finite field $\mathbb{F}$. It is convenient for presentation to assume that all gates have in degree 2 and out degree at most 2, but we can tolerate arbitrary constant degrees.

2.1 Setup

We first give a high level description of our algorithm. The first step is to build a network, which we call G, based on the circuit C. For every gate in the circuit C, there will be a node in G, which we will refer to as an internal node. In addition, G will have n extra nodes, corresponding to the n inputs of C. We will call these input nodes. For every wire from some gate to another gate in C, there will be an edge connecting the corresponding nodes in G. Further, for every wire from an input to a gate in C, there will be an edge from the corresponding input node to the corresponding internal node in G.

The players use the algorithm from Theorem 2.1 to divide themselves into n quorums each of size $\theta(\log n)$. Each quorum is assigned to some node in G. Recall we have $m + n$ nodes in G. We assume a canonical numbering of the nodes in G and of the n quorums, and we assign the quorum numbered i to to any node with number j s.t. $(j \mod n) = i$. Note that each quorum is thus assigned to at most $\lceil \frac{m+n}{n} \rceil$ nodes.

2.2 Extended Example

We now work through an extended example of our algorithm. The formal description of the algorithm is given in Section 2.4.

Figure 2 illustrates the example we will use to describe our algorithm. The top left illustration

[3]This scheme uses error correcting codes to achieve the verification. Other such schemes exist, which use Zero Knowledge proof techniques for verification and can tolerate up to half the players being faulty; see [21]. These, however, have an exponentially small but positive probability of error.

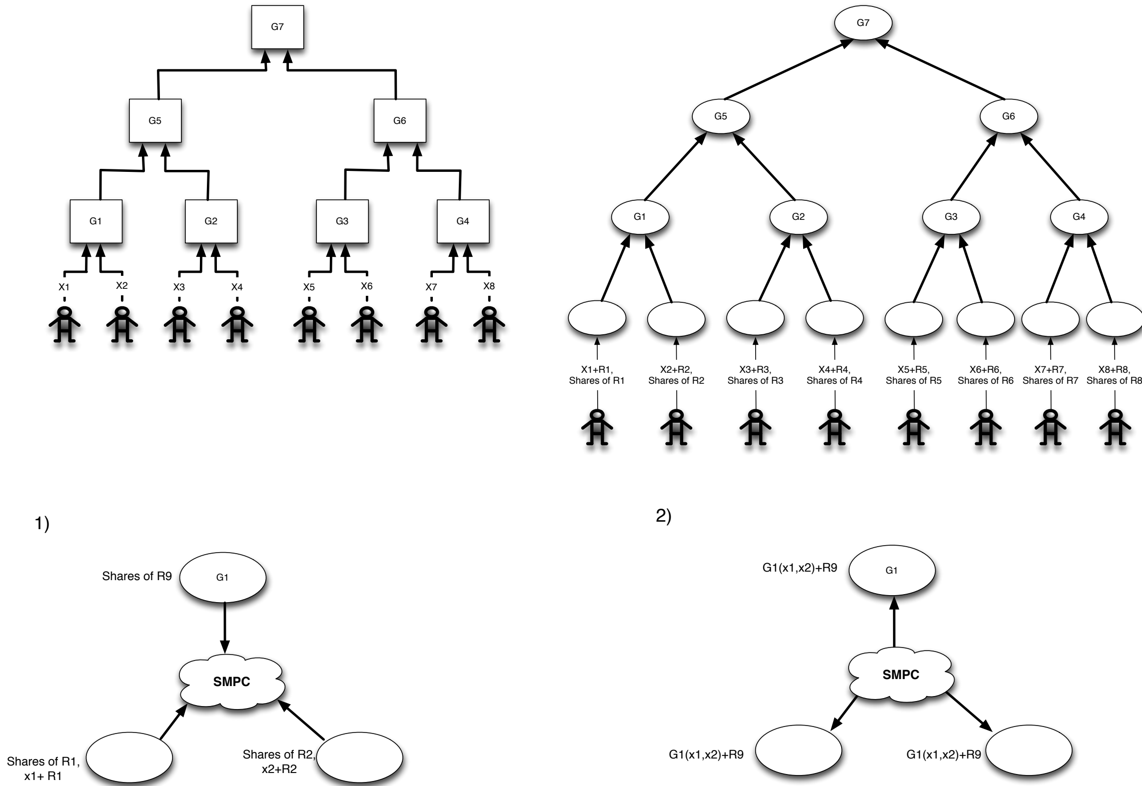

Figure 2: Example of Quorum based SMPC

in this figure describes a simple circuit with $m = 7$ gates and $n = 8$ inputs. For simplicity, this circuit is small; in a real application, we would expect both m and n to be much larger. Also, for simplicity, in this example, the circuit is a tree; however, our algorithm works for an arbitrary circuit. The circuit in this example computes an 8-ary function, f, that we want to compute in our SMPC. The gates have labels $G1, \ldots, G8$, that represent the functions computed by each gate. Each of the players at the bottom sends its input to some gate.

The top right illustration in the figure shows the layout of the quorums based on this circuit. Each oval in this illustration represents a quorum. There are $n + m$ ovals, m for each gate in the circuit and n for each player. Recall that using the algorithm from Theorem 2.1, we can create n quorums with the properties that 1) each quorum contains less than a $1/3$ fraction of bad players; 2) each quorum contains $\theta(\log n)$ players; and 3) each player is in $\theta(\log n)$ quorums. We will map these n quorums to the $m + n$ ovals. It will be the case that the number of ovals is larger than the number of actual quorums, requiring us to map some quorums to multiple ovals. However, each quorum will be mapped to at most $\lceil (n + m)/n \rceil$ ovals. Moreover, as we will see, it will not cause problems even if we map the same quorum to neighboring ovals. The algorithm begins by getting inputs from the players. In this illustration, each player i computes a value R_i selected uniformly at random from all values in the field $\mathbb{F}$. It then computes $x_i + R_i$, the value of its private input plus R_i and sends this value to all players in the quorum above it. Note that R_i "protects" the value x_i since $x_i + R_i$ is distributed completely uniformly at random. Finally, player i uses the

verifiable secret sharing (VSS) algorithm from [4] to create shares of R_i, and to send one share to each player in the quorum above. These shares have the property that any $2/3$ fraction of them can be used to reveal the value R_i, but less than a $2/3$ fraction reveals no information about R_i.

The two illustrations in the bottom part of the figure show how three quorums compute the output of each gate. We wish to maintain the following invariant: the value computed at any oval is the value that would be computed at the corresponding gate of C, masked by a random element of the field. The mask is jointly reconstructed by sufficiently many players at the oval, but it is not known to any individual player. For simplicity, these bottom illustrations focus solely on the computation occurring for $G1$; similar computations occur for all the other gates. Three quorums are involved in the computation for G1: the two bottom quorums provide the randomized inputs, and the top quorum provides a value ($R9$) that is used to randomize the output.

The bottom left illustration shows what is known at each quorum before the computation of $G1$. All players in the bottom left quorum know the value $x_1 + R_1$. Moreover, each player in this quorum has a share of the value R_1. These shares again have a $2/3$ threshold property: any $2/3$ fraction of them can be used to reconstruct R_1, but any set of less than $2/3$ of them reveals no information about R_1. The players in the bottom right quorum have similarly knowledge: they all know $x_2 + R_2$ and they each have shares of R_2 with a $2/3$ threshold property. Finally, the players in the top quorum have previously run a simple distributed algorithm to ensure that they each have a share of a value, R_9 that is selected uniformly at random from the field $\mathbb{F}$. These shares of R_9 are constructed with the $2/3$ threshold property; this property can be ensured done by repeated applications of the VSS algorithm from [4]. Finally, the players in all three quorums use HEAVYWEIGHT-SMPC to compute the value $G_1(x_1, x_2) + R_9$.

We note two important facts about this SMPC. First, the inputs ($x_1 + R_1$, $x_2 + R_2$, shares of R_1, shares of R_2, shares of R_9) contain enough information to compute the value $G_1(x_1, x_2) + R_9$ in the SMPC, even if the bad players lie about their inputs. Second, the SMPC algorithm occurs over only $\theta(\log n)$ players, so even a heavyweight protocol which runs in time and message cost polynomial in the number of players will incur latency and message costs that are just polylogarithmic in n.

The bottom right illustration shows the result after the computation of G_1. Each player in the three quorums has learned the value $G_1(x_1, x_2) + R_9$. Note that no player at any of the three quorums has (individually) learned any information about the value $G_1(x_1, x_2)$, since the mask R_9 which no individual knows, is uniformly random, and hence the computed value, $G_1(x_1, x_2) + R_9$ is also uniformly random over the field. In addition, note that we now have a situation for the top quorum where 1) every player knows the output value plus a random element R_9; and 2) the shares of R_9 are distributed among the players in such a way that the value R_9 can be reconstructed if and only if the good players in the quorum send the shares to each other. Thus, the top quorum is in the same situation now with respect to the value $G_1(x_1, x_2)$ as the bottom quorums were in with respect to x_1 and x_2 previously. Hence, the same procedure can be repeated as compute the values for the gates in the next layer of the circuit.

2.3 Some Details

The output of the quorum associated with the root node in G is the output of the entire algorithm. The last step of the algorithm is to send this output to all players. To do that, we construct a complete binary tree using the n quorums, with root quorum equal to the quorum that knows the output of the circuit. We then use majority filtering to pass the output down to all the players. Specifically, when a player receives the output message from all players in its parent quorum, it

computes the majority of all messages, and considers the majority of the messages as his correct output; then, it sends the output to all players in any quorums below.

Note that it may be the case that a player p participates $k > 1$ times in the quorums performing HEAVYWEIGHT-SMPC in Figure 2. In such a case, we allow p to play the role of k different players in the SMPC, one for each quorum to which p belongs. This ensures that the fraction of bad players in the heavy-weight SMPC is always less than 1/3. Also, the heavy-weight SMPC protocol of [21] maintains privacy guarantees even in the face of gossiping coalitions of constant size. Thus, p will learn no information beyond the output and its own inputs after running this protocol.

We observe that the output of the last node of G is the output of the algorithm. The last step of the algorithm is to send the output to all players. To do that, players reuse their quorums and build a complete binary tree with n nodes and assign quorum i to node i in the tree. Each player receives the output message from all players in its parent node and considers the majority of the messages as its correct output. Then, it sends the output to all players of its children nodes.

Finally, note that in this algorithm, each player participates in $\theta(\log n)$ quorums; each quorum is responsible for at most $\lceil (n+m)/n \rceil$ ovals; and the SMPC performed at an oval has resource cost which is polylogarithmic in n. Moreover, each player runs the VSS algorithm to send its input to a single quorum initially. Thus, in this algorithm, each player sends $\tilde{O}(\frac{n+m}{n})$ bits and is involved in the computation of $\tilde{O}(\frac{n+m}{n})$ gates.

2.4 Formal Description

We assume that the function to be computed is presented as a circuit C with c gates, numbered $1, 2, \ldots, m$, where the gate numbered 1 is the output gate. The high level picture of the comminication network is a directed graph G, with $c+n$ nodes numbered $1, 2, \ldots c+n$. The first c of these are "gate nodes", node i corresponding with gate i of the circuit, and there are edges between pairs of them whenever the corresponding pair of gates is connected by a wire. The direction of the edge is the direction of flow of computation in the circuit C. Note that the node numbered 1 is the node corresponding to the output gate. We will sometimes refer to this as the root node and denote it ρ. The additional n nodes are "input nodes" and input node i has an edge pointing to gate node j if the ith input wire feeds into gate j in C. For a given node v, we will refer to any node w to which v has an edge as a *parent* of v, and we will refer to any node x which has an edge to v as a *child* of v. Finally, for a given node v, we will say the *height* of v is the number of edges on the longest path from any leaf node to v.

The basic structure of the algorithm is as follows. First, all the players form quorums and each quorum is assigned to multiple nodes in G, so that each node in G is represented by a unique quorum (Algorithm 2). Then each player commits its input to the quorum at the corresponding input node in G (Algorithm 3). Then all quorums representing gate nodes generate shares of uniformly random field elements. These shares will be needed as inputs to the subsequent heavyweight SMPC protocols.

Next we begin computation of the gates of the circuit. For every node g in G associated with a gate in C, we do the following. At a time proportional to the height of the gate g, all participants in the computation of g (*i.e.* the quorums at g and the quorums at the two nodes pointing to g in the circuit) will run a heavyweight SMPC protocol to compute a masked version of the value at g. (Algorithm 5). Then the quorum at the root node will unmask the output (Algorithm 6) and it will be sent to all the players via a binary tree (Algorithm 7). In order for the players to coordinate their operations, we will need to define the following quantities. Let

$T_{\text{QF}} = T_{\text{QF}}(n)$ denote an upper bound on the time taken for n players to run the quorum formation algorithm.

$T_{\text{VSS}} = T_{\text{VSS}}(\log n)$ denote an upper bound on the input commitment via VSS.

$T_{\text{R}} = T_{\text{R}}(\log n)$ is the maximum time taken by the players in a single quorum to jointly generate shares of a random field element.

$T_{\text{SMPC}} = T_{\text{SMPC}}(\log n)$ denote an upper bound on the time it takes $O(\log n)$ players to perform a heavyweight SMPC.

We remind the reader that in our model local computation is instantaneous, and that a single "time unit" refers to the time taken for a message sent by a processor to reach its intended recipient.

We now present a formal description of our scalable SMPC protocol in Algorithm 1 and related subroutines. For convenience, we will sometimes abuse notation by allowing a node $v \in G$ to refer both to the node itself and to the quorum associated with the node.

Algorithm 1 Main Algorithm

Phase 1

1. At time $t = 0$ all players run the quorum formation algorithm (Algorithm 2).
2. At time $t = T_{\text{QF}}$ all players run the input commitment algorithm (Algorithm 3).
3. At time $t = T_{\text{QF}} + T_{\text{VSS}}$, for each gate simultaneously, players run the random number generation algorithm (Algorithm 4).
4. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}}$, for each gate g simultaneously, players initiate the computation of gate g (Algorithm 5).

Phase 2

5. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + h_\rho T_{\text{SMPC}}$, the players at the root node reconstruct the output (Algorithm 6). Here h_ρ is the height of the root node.
6. At time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + (h_\rho + 1)T_{\text{SMPC}}$, all players perform the output propagation algorithm (Algorithm 7)

2.5 Rational Players

We now show how to modify Algorithm 1 to handle rational players (Theorem 1.2); First, we note for the rational case, the graph G is equivalent to that in Algorithm 1. Moreover, the mapping from quorums to nodes in G is equivalent, except for the efficiency of the algorithm that creates the quorums. In particular, in the case where all players are rational, as is stated in Theorem 2.1, we require each player to send only $\tilde{O}(1)$ bits in order to create the set of n quorums.

Once the quorums have been formed, much of the algorithm, remains the same, including the input commitment and the (masked) computation of each gate. It is only at the output reconstruction stage of the algorithm that things need to change. The problem is that the SMPC protocol being used as a black box does not make any guarantees about all the players learning

Algorithm 2 Quorum Formation

This algorithm begins at time $t = 0$ and all players participate.

1. Run the algorithm in [18] to form n good quorums of size $O(\log n)$, numbered $1, 2, \ldots, n$, with the following properties:
   - All quorums have at least a 2/3 fraction of good players.
   - Each player participates in $O(\log n)$ quorums.
2. Each player identifies the nodes in G represented by his quorums, and the neighboring nodes in the graph G. The rule here is that quorum i represents gate j if $i = j \mod n$.
3. At the end of this protocol, each player knows
   - which $O(\log n)$ quorums to participate;
   - which other players are in each of those quorums;
   - which gates/nodes are represented by those quorums; and
   - which quorums represent the neighboring nodes (with whom it is necessary to communicate) and which players are in each of those quorums.

Algorithm 3 Input Commitment

This protocol for each input node begins at time $t = T_{\text{QF}}$. Recall that x_i is the input associated with player i.

1. Each player i chooses a uniformly random element $r_i \in \mathbb{F}$.
2. Each player i computes $s_i \leftarrow x_i + r_i$
3. Each player i creates VSS shares of r_i for each player in the quorum at input node $m + i$, using the BGW scheme, and sends one share to each member of this quorum. These shares have the property that r_i can be reconstructed from them even if upto a third of them are suppressed or misrepresented.
4. Each player i sends s_i to each member of the quorum at input node $m + i$.
5. Quorums mapped to each input node $m+i$ do the following: Run the VSS verification protocol to determine whether a valid secret has been shared. Also verify, using Byzantine agreement, that the same s_i has been sent to everyone. If either of these checks fails, set x_i to some preset default value, r_i and its shares to zero.

Algorithm 4 Random Number Generation

This protocol is run simultaneously by each quorum associated with each gate node $v \in G$ at time $t = T_{\mathrm{QF}} + T_{\mathrm{VSS}}$.
The following is done by each player $p \in v$:

1. Player $p \in v$ chooses uniformly at random an element $r_{p,v} \in \mathbb{F}$ (this must be done independently each time this algorithm is run and independently of all other randomness used to generate shares of inputs etc.)
2. Player p creates verifiable secret shares of $r_{p,v}$ for each player in g and deals these shares to all players in g (including itself).
3. Player p participates in the verification protocol for each received share. If the verification fails, set the particular share value to zero.
4. Player p adds together all the shares (including the one it dealt to itself). This sum will be player p's share of the value r_v.

Algorithm 5 Computation of a gate

This protocol is run simultaneously for each gate node $g \in G$, starting at time $t = T_{\mathrm{QF}} + T_{\mathrm{VSS}} + T_{\mathrm{R}} + (h_g - 1)T_{\mathrm{SMPC}}$, where h_g is the height of g. Let $v_1, v_2, \ldots v_k$ be the children of the node g in the graph G; and let $O_1, O_2, \ldots, O_k$ be the outputs of the gates associated with these children. The algorithm maintains the invariant that for each child node v_i, there is a uniformly random element $r_i \in \mathbb{F}$ and a value $s_i = O_i + r_i$, such that each player in v_i knows s_i and a unique VSS share of r_i. Also, each player at g has a VSS share of a value r_g that is a uniformly random element of $\mathbb{F}$. Let $f_g(O_1, O_2, \ldots, O_k)$ be the function computed by the gate in the circuit C associated with g.

1. Every player in the quorums $g, v_1, v_2, \ldots, v_k$ run HEAVYWEIGHT-SMPC with the inputs (s_1, shares of r_1, s_2, shares of $r_2, \ldots, s_k$, shares of r_k, shares of r_g) to compute a value s_g, where $s_g = f_g(O_1, O_2, \ldots, O_k) + r_g$. If a single player p appears in $k' > 1$ of these quorums p plays the role of k' different players in HEAVYWEIGHT-SMPC, one for each quorum to which p belongs.
2. The players in the quorum at g now have s_g and shares of r_g

---

Algorithm 6 Output Reconstruction

This protocol is run by all players in the quorum at the root node ρ, at time $t = T_{\mathrm{QF}} + T_{\mathrm{VSS}} + T_{\mathrm{R}} + h_\rho T_{\mathrm{SMPC}}$.

1. Reconstruct r_ρ from its shares using VSS.
2. Set the output $\mathbf{o} \leftarrow s_\rho - r_\rho$.
3. Send $\mathbf{o}$ to all players in the quorums numbered 2 and 3

Algorithm 7 Output Propagation

Performed by the players at each node by the players at each quorum, q other than the quorum numbered 1, starting at time $t = T_{\text{QF}} + T_{\text{VSS}} + T_{\text{R}} + (h_\rho + 1)T_{\text{SMPC}}$ wait)

1. $i \leftarrow$ quorum number of q
2. Each player $p \in q$ waits until it receives values from at least a 2/3 fraction of the players in the quorum numbered$\lfloor i/2 \rfloor$, and sets $\mathbf{o} \leftarrow$ the unique value that occurs as at least 2/3 of the received values.
3. Each player $p \in q$ sends $\mathbf{o}$ to all the players in quorums numbered $2i$ and $2i + 1$.

---

the output at the same time. This did not matter for the computations at internal gates since the intermediate output there was masked and therefore uniformly random, and gave the players no information about either the output or anybody's input. However at the end of the output reconstruction stage, players at the root actually learn the output. Thus if any single player learns it first, then he may simply stop sending messages and the other players will not learn the output. To overcome this difficulty, in the output reconstruction phase, instead of using the usual heavyweight SMPC protocol, we use a rational SMPC protocol due to Abraham, Dolev, Gonen and Halpern [1, Theorem 2(a)]. This ensures that all players at the root node learn the output simultaneously.

Finally the players at the root use Algorithm 7 in order to send the output to all n players. We note that to run Algorithm 7 at this point is a Nash equilibrium since if all other players are running Algorithm 7, there is no expected gain in utility for a single player by deviating from Algorithm 7.

3 Analysis

In this section, we give the proof of Theorem 1.1.

We begin by noting that the error probability in Theorem 1.1 comes entirely from the possibility that the quorum formation algorithm of Lonergan *et al.* [18] may fail to result in good quorums (see Theorem 2.1). All other components of our algorithm: the VSS and heavyweight black boxes, Byzantine agreement and majority filtering, are all exact algorithms with no error probability. For the remainder of this section we will assume that we are in the good event, *i.e.* that the players have successfully formed n good quorums.

For each node j in the graph G, let V_j be the value of the node in the computation of f. Thus, for input nodes, V_j is the input which has been committed to by the corresponding player (set to a default value if the player faulted on the input commitment algorithm), while for gate nodes, V_j is the value on the output wires of the gate associated with j in the circuit, once the inputs have been fixed to the committed values. Also for each node j we have a *mask* $r_j \in \mathbb{F}$. For input nodes, r_j is the random number set by the player in the input commitment algorithm (set to zero if the player faulted). For gate nodes, r_j is the random number jointly generated by the quorum at j. Let G' be the set of all nodes in G which are either input nodes corresponding to good players or gate nodes.

Lemma 1. The masks $\{r_j\}_{j \in G'}$ are fully independent and uniformly random in $\mathbb{F}$.

Proof. The masks corresponding to input nodes for good players are uniformly random by choice (see Algorithm 3). To see that the masks for the gates are uniformly random, recall that if j is

a gate node $r_j = \sum_i r_{j,i}$ where $r_{j,i}$ is the value selected by player i in Algorithm 4. The players commit to the $r_{j,i}$ values by sending each other VSS shares of them and then running the verification protocol on the shares. If player i is good, $r_{j,i}$ is uniformly random. If player i is bad then $r_{j,i}$ could be anything (including zero, if player i's shares failed the subsequent verification). However, once the players have committed to the values the bad players can no longer influence the sum of the $r_{j,i}$, nor can they bias the distributions of the $r_{j,i}$ in any way, because of the security provided by the VSS algorithm. Since the sum of elements of $\mathbb{F}$ is uniformly random if at least one of them is uniformly random, it follows that r_j is uniformly random. The independence of the $\{r_j\}$, $j \in G'$ follows from the fact that all players have sampled their values independently. □

In the following, the "computation of a node j" will refer either 1) the input commitment algorithm if j is an input nodes; or 2) Algorithm 5 if j is a gate node.

Lemma 2. For each node j in G, after the computation of j each player in the corresponding quorum knows a share of a number r_j. Moreover all good players in the quorum at j agree on a value $s_j \in \mathbb{F}$ such that $s_j - r_j = V_j$.

Proof. The players already have the shares of r_j at the end of the random number generation stage. We prove the claims about s_j by induction. For the base case, note that for each input node, since the corresponding quorum has at least two thirds good players, the conclusion follows from the correctness of the VSS protocol, and the Byzantine agreement protocol used in the input commitment algorithm.

Now let j be a gate node and suppose for all nodes j' whose height is less than the height of j, that all the good players at j agree on $s_{j'}$ and $s_{j'} - r_{j'} = V_{j'}$. Then the inductive hypothesis holds for all nodes v_1, v_2, v_k whose outputs are connected to the inputs of j. Thus, we can assume that for all i between 1 and k, the players at node v_i have shares of some value r_i chosen uniformly at random in $\mathbb{F}$, and that all players in node v_i know the value $s_i = V_i + r_i$. In the computation at node j, the $k+1$ quorums involved run HEAVYWEIGHT-SMPC with inputs $s_1, s_2, \ldots s_k$ and the shares of $r_j, r_1, r_2, \ldots, r_k$. At the end of this protocol, all good players agree on a common value s_g. (This is by the correctness of HEAVYWEIGHT-SMPC).

To see that this common value is actually $V_g + r_g$ we note that the function computed by HEAVYWEIGHT-SMPC consists of reconstructing $r_g, r_1, r_2 \ldots, r_k$ from their shares; inferring the values $V_1, V_2, \ldots, V_k$; computing V_g from them; and adding r_g back in. (All of this will, of course, be opaque to the players involved.) Attempts to corrupt this computation by lying about $s_1, s_2, \ldots, s_k$ are easily thwarted, because of the high redundancy in these as inputs. For each of these values, at least twice as many players provide them correctly as try to lie (since each of the input quorums have at most a third bad players). Moreover, note that the VSS used to reconstruct the masks from the shares can tolerate up to a third of the shares being corrupted. Thus, since all quorums are good, even if the bad players lie about their shares of the masks, they cannot change the value of the computation. It follows that $s_g = V_g + r_g$. By induction, all the nodes in G compute the correct masked values □

Corollary 1. After the Output Reconstruction (Algorithm 6), all players at the root node know the output.

Proof. By Lemma 2, at the end of Phase 1 of the main algorithm, all the players at the root node know the value s_ρ and shares of r_ρ, where $s_\rho - r_\rho$ is the output of the circuit. During Algorithm 6, these players run the VSS secret reconstruction protocol. Since at least two thirds of them are

good, by properties of VSS, they correctly reconstruct r_ρ. Since all players at the root node know the value of s_ρ, subtracting from it the reconstructed r_ρ, they all learn the correct output. □

Lemma 3. At the end of the algorithm, the correct output is learned by all good players.

Proof. This follows by induction. Since quorum 1 is at the root, Corollary 1 provides a base case. Now suppose the correct output has been learned by all the players in quorums numbered j for all $j < i$. Consider the players in quorum i. During the run of the output propagation algorithm, they will receive putative values for the output from the players at quorum $\lfloor i/2 \rfloor$. Since at least two thirds of the players at quorum $\lfloor i/2 \rfloor$ are good, and by induction hypothesis have learned the correct output, it follows that at least two thirds of the values received by the players at quorum i equal correct output. Since good players set their output to be the the unique value that occurs as at least $2/3$ of the received values, they get the correct output. By induction, all the players learn the correct value. □

We devote the rest of the section to showing that privacy of the inputs is preserved. We remark that privacy is only gurarranteed with high probability. However, as in the case of correctness, the error arises only from the possibility that the quorum formation algorithm fails to spread the bad players out so that less than a third of the players in any quorum are bad. Thus if we condition on having formed n good quorums, then all the privacy claims hold with probability 1. For what follows we will continue to condition on this good event.

As discussed in previous works (see [21]), we have no recourse against players who voluntarily send their inputs to other players, naturally we cannot preserve the privacy of such players. In particular, we are only concerned with preserving the privacy of good players, who perform no actions except those specified by the protocol.

We are primarily concerned with preserving the privacy of *inputs of players*. However, note that if some player's input feeds into a multiplication gate then learning that the value in the computation of that gate is zero, increases the Bayesian probability that the player's input is zero, and this is a privacy violation. Thus we are also concerned about the ability of players to learn the value of a gate other than the root or output gate.

Recall that G' is the set of nodes in G that are either input nodes corresponding to good players or gate nodes.

Lemma 4. Let j be any node in G', other than the root node, ρ. Using only messages sent to him as part of the algorithm, no player can learn any information about the value V_j, except what is implicit in his own input and the final output of the circuit.

Proof. We prove this for a gate node g. By Lemmas 1 and 2, the value recovered by HEAVYWEIGHT-SMPC during the computation of g is $s_g = V_g + r_g$, where r_g is a uniformly random element of $\mathbb{F}$, independent of all other randomness in the algorithm. In particular this means that s_g holds no information about V_g. If the player i is not in any of the quorums at g or its neighbors, then all the messages he receives during the algorithm are independent of r_g, and hence s_g, and hence he cannot learn anything about V_g. On the other hand, if player i is involved in the computation of g or one of its neighbors, then he may hold a share r_g as well as shares of other shares. In this case we appeal to the privacy of HEAVYWEIGHT-SMPC and the embedded VSS algorithm to see that although he may learn s_g, he cannot learn any information about the shares of r_g and hence about r_g itself. Thus, he cannot learn any information about V_g except what is implicit in his input and the circuit output.

The proof for an input node of a good player is similar except that we will have to appeal to the privacy of the black box VSS protocol rather than the privacy of HEAVYWEIGHT-SMPC. □

We now explore a stronger notion of privacy. BGW [4] distinguish between the two kinds of deviant behaviour among players. The bad players are players controlled by an adversary who may indulge in arbitrary kinds of erratic behaviour to try to break the protocol in any way they can. However BGW also consider players who are good, in the sense that they follow the protocol, but may also send and receive messages external to the protocol, to attempt to learn whatever additional information they can. Such players are called *gossiping* players. A protocol is called t-private if no coalition of size t (including coalitions of gossiping players) can learn anything more than what is implied by their private inputs and the circuit output. The SMPC protocol of BGW [4] is $(n/3 - \delta)$-private for any $\delta > 0$.

We note that our algorithm *is* susceptible to adaptively chosen coalitions of gossiping players. Indeed, if all the players in a quorum at a node j gossip with each other, they can reconstruct the corresponding random mask r_j and hence the value V_j. In particular, the players in the quorum at an input node can jointly reconstruct the corresponding input.

However, we can establish the following result, which shows that for large coalitions chosen non-adaptively (in particular, the adversarial players) our algorithm will preserve privacy.

Lemma 5. Let S be any set of players such that for every quorum Q, $S \cap Q$ contains fewer than a third of the players in Q. Let j be any node in G'. Then the coalition S cannot learn any information about V_j that cannot be computed from their (collective) private inputs and the circuit output.

Proof. Once again we prove this only for gate node g in G'. The proof for an input node is similar. We know that HEAVYWEIGHT-SMPC when run at g computes $s_g = V_g + r_g$, where r_g is uniform in $\mathbb{F}$ and independent of all other randomness in the algorithm. As noted in the proof of Lemma 4, the players in S who are not in the quorums at g or any of its neighbors are irrelevant to the coalition: all of the information that they hold is completely independent of r_g and s_g, so they cannot assist in uncovering any information about V_g, except what is implicit in their private inputs.

Now consider the players in the quorums at g or any of its neighbors. These players participate in one or more of the SMPCs which involve g: the computation of g itself or the computations in which the output of g is an input. Here we appeal to the privacy of HEAVYWEIGHT-SMPC to see that the players cannot learn any additional information that is not implied by their inputs. The players in S are unable to directly determine r_g, since the only relevant inputs are the shares of r_g, and they do not have enough of those.

Finally, let us consider the players from S at g itself. These players also do not have enough shares of r_g to reconstruct it on their own. However, they recieve shares of each of the other shares of r_g multiple times: once during the input commitment phase of each SMPC in which g is involved. Each time, they do not get enough shares of shares r_g to reconstruct any shares of r_g. However, can they combine the shares of shares from different runs of the VSS protocol for the same secret to gain some information? Since fresh, independent randomness was used by the dealers creating these shares on each run, the shares from each run are independent of the other runs, and so they do not collectively give any more information than each of the runs give separately. Since each run of the VSS input commitment does not give the players in S enough shares to reconstruct anything, it follows that they do not learn any information about r_g. Since r_g is uniformly random, so is s_g and it follows that the coalition S cannot get any extra information about V_g. □

Corollary 2. The bad players cannot learn any information, except what is implied by the output and the inputs to which they committed, about the input of any good player.

Proof. This follows immediately from Lemma 5, since each quorum consists of no more than a third bad players. □

Let q be the size of the smallest quorum. Recall that $q = \Theta(\log n)$.

Corollary 3. Our algorithm is $q/3$-private.

Proof. Since q is the size of the smallest quorum, any set of size $q/3$ intersects a quorum Q in at most a third of its members. The result follows from Lemma 5 □

We end with a simple analysis of the resource cost of our algorithm.

Lemma 6. If all good players follow Algorithm 1, with high probability, each players sends at most $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ messages. m is size of G

Proof. To analyze the cost of algorithm 1, we have to first analyze the cost of its sub-algorithms.

Cost of Algorithm 2 and Algorithm 3: Based on the theorem 2.1 we need to send $\tilde{O}(\sqrt{n})$ messages to build the quorums. In Algorithm 3, each player must commit its secret and a random variable using verified secret sharing between $O(\log n)$ players of a quorum (input node). This requires sending a polylogarithmic number of messages.

Cost of Algorithm 5: Each player will participate in $\theta(\log n)$ quorums. For each quorum, he has to participate in a secure multi-party computation for $\theta(\frac{m+n}{n})$ (m is number of operations in circuit G) gates between three quorums or $3 \log n$ players which is polylogarithmic, so this algorithm requires sending $\tilde{O}(\log n \frac{n+c}{n})$ messages.

Cost of Algorithm 7: output tree, Each player should send $\tilde{O}(1)$ messages (output message) to the players of its children.

So the cost of the algorithm 1 is $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$.

□

4 Conclusion

We have described scalable algorithms to perform Secure Multiparty Computation in a scalable manner. Our algorithms are scalable in the sense that they require each player to send $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ messages and perform $\tilde{O}(\frac{n+m}{n} + \sqrt{n})$ computations. They tolerate an adversary that controls up to a $1/3 - \epsilon$ fraction of the players, for ϵ any positive constant. We have also described a variant of this algorithm that tolerates the case where all players are rational; this variant requires each player to send $\tilde{O}(\frac{n+m}{n})$ messages and perform $\tilde{O}(\frac{n+m}{n})$ computations.

Many open problems remain including the following. First, Can we design scalable algorithms to solve SMPC in the completely asynchronous communication model? We believe this is possible with some work. Second, Can we prove lower bounds for the communication and computation costs for Monte Carlo SMPC? Finally, Can we implement and adapt these algorithms to make them practical for a SMPC problem such as the one described in [5].

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 53–62. ACM, 2006.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: efficient verification via secure computation. *Automata, Languages and Programming*, pages 152–163, 2010.

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

[4] 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.

[5] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. *Financial Cryptography and Data Security*, pages 325–343, 2009.

[6] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, New York, NY, USA, 1984. ACM.

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

[8] 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.

[9] 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.

[10] 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*, pages 13–22. ACM, 2001.

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

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

[13] S. Gordon and J. Katz. Rational secret sharing, revisited. *Security and Cryptography for Networks*, pages 229–241, 2006.

[14] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, page 632. ACM, 2004.

[15] W. Henecka, A.R. Sadeghi, T. Schneider, I. Wehrenberg, et al. Tasty: Tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462. ACM, 2010.

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

[17] 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.

[18] 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.

[19] G. Kol and M. Naor. Games for exchanging information. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 423–432. ACM, 2008.

[20] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. *Advances in Cryptology-CRYPTO 2006*, pages 180–197, 2006.

[21] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[22] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.