Communication-Efficient MPC for General Adversary Structures

Joshua Lampkins[1] and Rafail Ostrovsky[2]
jlampkins@math.ucla.edu, rafail@cs.ucla.edu
[1] Department of Mathematics, University of California, Los Angeles
[2] Department of Computer Science, University of California, Los Angeles

October 5, 2013

Abstract. A multiparty computation (MPC) protocol allows a set of players to compute a function of their inputs while keeping the inputs private and at the same time securing the correctness of the output. Most MPC protocols assume that the adversary can corrupt up to a fixed fraction of the number of players. Hirt and Maurer initiated the study of MPC under more general corruption patterns, in which the adversary is allowed to corrupt any set of players in some pre-defined collection of sets [6]. In this paper we consider this important direction of research and present significantly improved communication complexity of MPC protocols for general adversary structures. More specifically, ours is the first unconditionally secure protocol that achieves linear communication in the size of Monotone Span Program representing the adversary structure in the malicious setting against any $Q2$ adversary structure, whereas all previous protocols were at least cubic.

Keywords: Multiparty Computation, Secret Sharing, General Adversaries, $Q2$ Adversary Structures

Paper	[5]	[7]	[4]	**This Paper**
Bandwidth	$O(\mathcal{C}d^{3.5})$	$O(\mathcal{C}\kappa^2 nd^3)$	$O(\mathcal{C}nd^3)$	$O(\mathcal{C}n^2d + n^3d + \kappa n^4 \log d)$

Table 1: Comparison of MPC protocols secure against active, $Q2$ adversaries with a broadcast channel. The "critical" parameter that we are minimizing (which dominates all other terms) is d which is the size of the smallest MSP representing adversary structure, which can be exponential in the number of players, denoted as n. $\mathcal{C}$ is the size of the circuit and κ is the security parameter. Bandwidth is measured in field elements, and counts both point-to-point communications and broadcasts.

1 Introduction

In a multiparty computation (MPC) protocol, it is assumed that some of the players might collude together to attempt to determine some other player's input or to alter the output of the function. This is generally modeled as a single adversary corrupting a subset of the players. In order for a protocol to work, one must assume that the adversary is limited in the number of players he can corrupt. Most MPC protocols have a simple threshold requirement on the adversary. For instance, if the total number of players is n and the number of players corrupted by the adversary is t, then a protocol might require $t < n/3$ or $t < n/2$.

In this paper, we consider requirements on the malicious and adaptive adversary which are more general than just threshold requirements. If $\mathcal{P}$ is the set of players, then the most general way of expressing the limitations of the adversary is to select a subset $\mathcal{A} \subset 2^{\mathcal{P}}$, called an adversary structure. The adversary is then allowed to corrupt any set of players in $\mathcal{A}$. This paper constructs a Multiparty Computation protocol that is secure against general $Q2$ adversary structures.[1]

1.1 Previous Work

The first MPC protocol for general adversaries was given in Hirt and Maurer [6]. The protocol was recursive, relying on the use of virtual processors/players and "nesting" the virtualization. The MPC protocol for active $Q2$ adversaries with a broadcast channel had communication complexity superpolynomial in the size of the description of the adversary structure; the protocol was slightly modified in Fitzi, Hirt and Maurer [5] to yield polynomial communication complexity. More explicit protocols were given in Cramer, Damgård and Maurer [3], Smith and Stiglic [7], and Cramer, Damgård, Dziembowski, Hirt and Rabin [4], each paper constructing an MPC protocol based on the Monotone Span Program (MSP) secret sharing scheme developed in Cramer, Damgard and Maurer [3]. An MPC protocol based on a different secret sharing scheme is given in Beaver and Wool [1], which dealt with passive adversaries only.

1.2 Our Contributions

This paper provides an MPC protocol in the setting of an active $Q2$ adversary with a broadcast channel. We strictly improve upon the amortized efficiency of all previous protocols, as shown in Table 1. (Since [3] dealt with active $Q3$ adversaries only, and since [1] only dealt with passive adversaries, they are not listed in the table.) In examining the table note that d is the dominating term[2] and can be exponential in n and is always at least n. So in addition to providing a strict improvement over all previous protocols, our result is the first MPC protocol secure against active and adaptive $Q2$ adversaries that has communication complexity linear in d.

[1] $Q2$ adversary structures are defined in section 2.

[2] where d is the size of the smallest monotone span program (MSP) representing the adversary structure as defined in section 2

1.3 Techniques

One way of dealing with disputes is with a technique called Kudzu shares, as first defined by Beerliová-Trubíniová and Hirt [2]. When one player accuses another of lying (or other such misbehavior), they are said to be in dispute. When a dealer distributes a secret s, the shares sent to players in dispute with the dealer are defined to be zero. That is, instead of using a standard Shamir secret sharing, the dealer picks a random polynomial f such that $f(0) = s$ and $f(i) = 0$ for each P_i in dispute with the dealer. Then the shares $f(i)$ are sent to each player P_i. The shares of the players in dispute with the dealer are called Kudzu shares. Since the set of players in dispute with the dealer is public knowledge, *every* player will know the shares sent to players in dispute with the dealer. This prevents the recipients from lying about the shares they received later in the protocol. The secret sharing scheme from [3] can also be adapted to implement Kudzu shares; ours is the first MPC protocol to implement Kudzu shares with MSP secret sharing for general adversaries.

There are 3 types of sharings that are used during the computation phase of the protocol: Inputs from players, random inputs, and multiplication triples for evaluating multiplication gates. These sharings are jointly generated by the players. Each such sharing is a sum of sharings from each of the players (other than those known to be corrupt). For our protocol, each player will remember for each share he holds which part of the sum came from which player. In general, we simply require that each player remember all messages sent and received. This is done in order to facilitate dispute control in the MPC protocol, and in particular in the sub-protocol LC-Reconstruct.

2 Definitions and Assumptions

Our MPC protocol is designed for a synchronous network with secure point-to-point channels and an authenticated broadcast channel.

We denote the player set by $\mathcal{P}$. The players are to compute an arithmetic circuit over a finite field $\mathbb{F}$ of size $|\mathbb{F}| = 2^\kappa$. We let c_I, c_M, c_R, c_O denote the number of input, multiplication, random, and output gates (respectively) in the circuit. The total size of the circuit is $\mathcal{C} = c_I + c_M + c_R + c_O$. The multiplicative depth of the circuit is $\mathcal{D}$. We denote the adversary structure by $\mathcal{A} \subset 2^{\mathcal{P}}$. Adversary structures are monotone, meaning that if $A \in \mathcal{A}$, then any subset of A is in $\mathcal{A}$. We denote by $\overline{\mathcal{A}}$ the set of maximal sets in $\mathcal{A}$ (i.e., the set of all sets in $\mathcal{A}$ that are not proper subsets of any other sets in $\mathcal{A}$). An adversary structure $\mathcal{A}$ is said to be $Q2$ if $A, B \in \mathcal{A} \Rightarrow A \cup B \neq \mathcal{P}$. Our MPC protocol is able to tolerate an active, adaptive adversary whose corruption pattern is specified by a $Q2$ adversary structure.

We denote by $\mathcal{D}isp$ the set of pairs of players who are in dispute with one another. If at any time a dispute arises between player P_i and player P_j, (i.e., one of them says that the other is lying), the pair $\{P_i, P_j\}$ is added to $\mathcal{D}isp$. Since all disputes are handled over the broadcast channel, each player has the same record of which pairs of players are in $\mathcal{D}isp$. We define $\mathcal{D}isp_i = \{P_j \mid \{P_j, P_i\} \in \mathcal{D}isp\}$. If at any time the set $\mathcal{D}isp_i$ is no longer in $\mathcal{A}$, that means that at least one honest player has accused P_i, and therefore all players know that P_i must be corrupt. We use the set $\mathcal{C}orr$ to denote the set of players known by all players to be corrupt. When a player is added to $\mathcal{C}orr$, he is also added to $\mathcal{D}isp_i$ for each P_i. (That is, players that are known to be corrupt are in conflict with all other players.) We define $\mathcal{G}ood = \mathcal{P} - \mathcal{C}orr$. Whenever $\mathcal{C}orr$ is changed, $\mathcal{G}ood$ is correspondingly changed.

Most of the protocols in this paper use dispute control and will terminate when one or more pairs of players are added to $\mathcal{D}isp$. In this case, the protocol terminates unsuccessfully. We handle unsuccessful termination of protocols as in [2]. Namely, the circuit is divided into (roughly) n^2 segments, and if one of the sub-protocols terminates unsuccessfully during the computation for a segment, that segment is started over from the beginning. A new dispute is found at each unsuccessful termination, and since there can be at most n^2 disputes, this does not affect the asymptotic complexity of the protocol. Throughout this

paper, we will assume without explicitly stating it that if a sub-protocol invoked by a parent protocol terminates unsuccessfully, then the parent protocol terminates unsuccessfully.

Let M be a matrix over $\mathbb{F}$ with d rows and e columns, and let[3] $\mathbf{a} = (1, 0, 0, \cdots, 0)^\top \in \mathbb{F}^e$. The triple $(\mathbb{F}, M, \mathbf{a})$ is called a *monotone span program* (MSP).

Define $(x_1, x_2, \ldots, x_\ell)^\top * (y_1, y_2, \ldots, y_\ell)^\top = (x_1y_1, x_2y_2, \ldots, x_\ell y_\ell)^\top$, and suppose $\boldsymbol{\lambda}$ is a vector in $\mathbb{F}^d$. We call $(\mathbb{F}, M, \mathbf{a}, \boldsymbol{\lambda})$ a *monotone span program with multiplication* if $(\mathbb{F}, M, \mathbf{a})$ is an MSP and if $\boldsymbol{\lambda}$ has the property that

$$\langle \boldsymbol{\lambda}, M\mathbf{b} * M\mathbf{b}' \rangle = \langle \mathbf{a}, \mathbf{b} \rangle \cdot \langle \mathbf{a}, \mathbf{b}' \rangle$$

for all $\mathbf{b}, \mathbf{b}'$. In this case, $\boldsymbol{\lambda}$ is called the *recombination vector*.

Each row of M will be labeled with an index i ($1 \le i \le n$), so that each row corresponds to some player. For any nonempty subset $A \subset \{1, 2, \ldots, n\}$, M_A denotes the matrix consisting of all rows whose index is in A.

For a given adversary structure $\mathcal{A}$, we say that the MSP $(\mathbb{F}, M, \mathbf{a})$ *computes* $\mathcal{A}$ if

$$A \notin \mathcal{A} \quad \Longleftrightarrow \quad \mathbf{a} \in \operatorname{Im} M_A^\top.$$

It was implicit in [6] that any $Q2$ adversary structure can be represented by circuit of threshold gates as follows: Each input to the circuit corresponds to a single player. (A player may be associated to more than one input.) The circuit is composed only of majority accepting threshold gates (i.e., k-out-of-n threshold gates such that $2k \le n + 1$). For a set $A \in 2^{\mathcal{P}}$, each input is set to 1 if the corresponding player is in A and 0 if that player is not in A. The circuit outputs 0 if $A \in \mathcal{A}$ and outputs 1 if $A \notin \mathcal{A}$. It was shown in [3] that if an adversary structure can be represented by a circuit as above, then there is an MSP with multiplication which computes that adversary structure.

The size of the monotone span program representing the adversary structure (measured as the number of rows in the matrix) is of prime importance in analyzing the communication complexity of the MPC protocol, because secrets are shared as a vector in the image of M. There is an algorithm (implicit in [6]) that allows one to construct a monotone span program with multiplication from $\overline{\mathcal{A}}$ for any $Q2$ adversary structure $\mathcal{A}$ such that d is polynomially related to $|\overline{\mathcal{A}}|$. (The number of columns of the matrix, which we denote by e, will always be less than d. This is important simply because $e < d$ implies that the image of M is not all of $\mathbb{F}^d$, a fact which will be used later.)

A "basic" sharing of a value w created using the MSP (as generated by the protocol Share below) is denoted by $[w]$. The share of player P_i is denoted by w_i. We denote the portion of the share w_i that came from player P_j by $w_i^{(j)}$. Note that in general w_i will be a vector, since it represents a portion of a vector in the image of M, although it could be a single-element vector. The length of P_i's share will depend on how P_i is represented in the adversary structure.

This secret sharing scheme is linear in that each player can compute a sharing of an affine combination of already-shared secrets by performing local computations. For instance, if secrets $[s^{(1)}], [s^{(2)}], \ldots, [s^{(\ell)}]$ have already been shared, then the players can compute a sharing of $r = a^{(0)} + \sum_{k=1}^{\ell} a^{(k)} s^{(k)}$ for publicly known constants $a^{(0)}, \ldots, a^{(\ell)}$ by each P_i locally computing $r_i = a_i^{(0)} + \sum_{k=1}^{\ell} a^{(k)} s_i^{(k)}$.

3 The Protocols

This section describes the MPC protocol and all sub-protocols. Proofs are deferred to Appendix A.1.

[3] The protocol actually works if $\mathbf{a}$ is any fixed vector, but it is convenient to choose $\mathbf{a}$ as we have.

3.1 Secret Sharing

Our MPC protocol uses a "basic" secret sharing protocol and constructs a verifiable secret sharing (VSS) protocol by combining the basic protocol with information checking [2]. The basic secret sharing protocol—which is described in this section—is essentially the secret sharing protocol of [3], except that it is implemented with Kudzu shares [2]. We first review the secret sharing protocol of [3] and then prove that this can be implemented with Kudzu shares.

Given an MSP with matrix M of size $d \times e$ as described in section 2, the secret sharing protocol of [3] proceeds as follows: The dealer with secret s picks $e-1$ random values $r_2, \ldots, r_e$, constructing a vector $\mathbf{s} = (s, r_2, r_3, \ldots, r_e)$. The dealer then computes $\mathbf{b} = M\mathbf{s} = [s]$ and sends some of the entries of the vector $\mathbf{b}$ to each player. It is shown in [3] how to construct MSPs suitable for secret sharing for any given $Q2$ adversary structure.

To implement Kudzu shares with this secret sharing scheme, we note that the secret sharing scheme described above is perfectly private (proved in [3]). In other words, the adversary's view of the vector $\mathbf{b}$ is independent of the secret being shared. So for a sharing $\mathbf{b} = [s]$, the dealer can construct a sharing of zero $[0]$ such that the adversary's view of $[0]$ is the same as his view of $[s]$. Then the sharing $[s] - [0]$ is a sharing of s with Kudzu shares, as the shares of all players controlled by the adversary will be zero.

Protocol: Share(D, s)

The dealer P_D wants to share a secret $s \in \mathbb{F}$. He selects random values $r_2, r_3, \ldots, r_e \in \mathbb{F}$, constructing a vector $\mathbf{s} = (s, r_2, r_3, \ldots, r_e) \in \mathbb{F}^e$. The random values $r_2, r_3, \ldots, r_e$ are chosen subject to the constraint that the shares of players in dispute with P_D must be all-zero vectors. The dealer then computes $[s] = \mathbf{b} = M\mathbf{s}$, where M is the MSP corresponding to $\mathcal{A}$. The dealer sends $\mathbf{b}_j$ to each $P_j \notin \mathcal{D}isp_D$ (where $\mathbf{b}_j$ is the vector of components of $\mathbf{b}$ corresponding to player P_j).

In this paper, we will represent the complexity of each protocol in a table. The columns denote communication bandwidth, broadcast bandwidth, communication rounds, and broadcast rounds (abbreviated CB, BCB, CR, and BCR, respectively). The two rows represent the complexity in the absence of a dispute and the added complexity per dispute. It is assumed that the communication and broadcast bandwidths are stated asymptotically (i.e., the big-O is not written, but is assumed). Bandwidth is measured in field elements, so one would have to multiply by κ to compute the bandwidth in *bits*.

Share	CB	BCB	CR	BCR
$WithoutDispute$	d	0	1	0
$PerDispute$	0	0	0	0

For a value $v \in \mathbb{F}$, we call the *canonical sharing* of v the sharing for which $r_2, r_3, \ldots, r_e$ are all zero.

Lemma 1. *The protocol* Share *is a secret sharing scheme secure against any active, adaptive adversary with $Q2$ adversary structure $\mathcal{A}$.*

3.2 Information Checking

Information checking (IC) is a scheme by which a sender can give a message to a receiver along with some auxiliary information (verification tags); the sender also gives some auxiliary information (authentication tags) to a verifier. This is done such that at a later time, if there is a disagreement about what the sender gave the receiver, the verifier can act as an "objective third party" to settle the dispute. We ensure that the verifier does not find out any information about the message (until a dispute arises).

The protocols Distribute-Tags and Check-Message are are variants of those used in [2], so their explicit description is deferred to Appendix A.2. The main difference is that we use an extension field $\mathbb{G}$ of $\mathbb{F}$ to allow the sender to produce tags for messages of length at most d. Since d can be as much as exponential in n, this is a much larger message size than that allowed in [2].

Lemma 2. *The following four facts hold.*

1. *If* Distribute-Tags *succeeds and P_V, P_R are honest, then with overwhelming probability P_V accepts the linear combination of the messages in* Check-Message.
2. *If* Distribute-Tags *fails, then a new pair of players is added to $\mathcal{D}isp$, and at least one of the two players is corrupt.*
3. *If P_S and P_V are honest, then with overwhelming probability, P_V rejects any fake message $\mathbf{m}' \neq \mathbf{m}$ in* Check-Message.
4. *If P_S and P_R are honest, then P_V obtains no information about $\mathbf{m}$ in* Distribute-Tags *(even if it fails).*

The proof of this lemma and the complexities of the information checking protocols are given in Appendix A.2.

3.3 Verifiable Secret Sharing

A verifiable secret sharing (VSS) scheme consists of two protocols, VSS and VSS-Reconstruct. We use the following definition of secret sharing:

Definition 1. *Consider a protocol* VSS *for distributing shares of a secret s and a protocol* VSS-Reconstruct *for reconstructing s from the shares. We call this pair of protocols a VSS scheme if the following properties are satisfied (with overwhelming probability):*

1. ***Termination:*** *Either all honest players complete* VSS*, or a new dispute is found. All honest players will complete* VSS-Reconstruct.
2. ***Privacy:*** *If the dealer is honest, then before the beginning of* VSS-Reconstruct*, the adversary has no information on the shared secret s.*
3. ***Correctness:*** *Once all honest players complete* VSS *there is a fixed value r such that:*
   - *3.1 If the dealer was honest throughout* VSS*, then $r = s$.*
   - *3.2 Whether or not the dealer is honest, at the end of* VSS-Reconstruct *the honest players will reconstruct r.*

The following protocol allows a dealer $P_D \in \mathcal{G}ood$ to verifiably share ℓ values. To verify correctness, each player acts as verifier and requests a random linear combination of these sharings (masked by a random sharing) to be opened. If the sharing is inconsistent (meaning that it is not in the span of M), then dispute resolution occurs. In addition, when P_D shares secrets, he utilizes information checking to produce authentication and verification tags in case a disagreement occurs later as to what was sent.

Protocol: VSS$(P_D, \ell, s^{(1)}, \ldots, s^{(\ell)})$

We assume that $P_D \in \mathcal{G}ood$ wants to share $s^{(1)}, \ldots, s^{(\ell)}$. If $P_D \in \mathcal{C}orr$, then all the sharings will be defined to be all-zero sharings.

1. *Distribution*
   - *1.1* P_D selects n extra random values $u^{(1)}, \ldots, u^{(n)}$, and then invokes Share to share $\{u^{(i)}\}_{i=1}^{n}$ and $\{s^{(i)}\}_{i=1}^{\ell}$.

1.2 For each pair $P_R, P_V \notin \mathcal{D}isp_D$ such that $\{P_R, P_V\} \notin \mathcal{D}isp$, invoke Distribute-Tags$(P_D, P_R, P_V, \mathbf{s}_R)$, where

$$\mathbf{s}_R = (s_R^{(1)}, \ldots, s_R^{(\ell)}, u_R^{(1)}, \ldots, u_R^{(n)})$$

(remember that each $s_R^{(k)}$ and $u_R^{(k)}$ is a vector).

2. *Verification*

The following steps are performed in parallel for each $P_V \notin \mathcal{D}isp_D$, who acts as verifier.

2.1 P_V choses a random vector $(r_1, \ldots, r_\ell) \in \mathbb{F}^\ell$ and broadcasts it.

2.2 Each player $P_i \notin \mathcal{D}isp_D$ sends his share of $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ to P_V.

2.3 If P_V finds that the shares he received in the previous step (together with the Kudzu shares) form a consistent sharing, (i.e., it is a vector in the span of $M_{\mathcal{P}-\mathcal{C}orr}$), then P_V broadcasts $(\texttt{accept}, P_D)$, and the protocol terminates. Otherwise, P_V broadcasts $(\texttt{reject}, P_D)$.

3. *Fault Localization*

For the lowest P_V that broadcast "$(\texttt{reject}, P_D)$" in the previous step, then the following steps are performed.

3.1 P_D broadcasts each share of $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$. If this sharing is inconsistent, then P_D is added to $\mathcal{C}orr$ and the protocol terminates.

3.2 If the protocol did not terminate in the last step, then there is a share of some player $P_i \notin \mathcal{D}isp_D$ that broadcast a different share than P_D. So P_V broadcasts $(\texttt{accuse}, P_i, P_D, v_i, v_D)$, where v_i is the value of the share sent by P_i and v_D the value sent by P_D.

3.3 If P_i disagrees with the value v_i broadcast by P_V, then P_i broadcasts $(\texttt{dispute}, P_i, P_V)$, the set $\{P_i, P_V\}$ is added to $\mathcal{D}isp$, and the protocol terminates.

3.4 If P_D disagrees with the value v_D broadcast by P_V, then P_D broadcasts $(\texttt{dispute}, P_D, P_V)$, the set $\{P_D, P_V\}$ is added to $\mathcal{D}isp$, and the protocol terminates.

3.5 If neither P_i nor P_D complained in the previous two steps, then $\{P_i, P_D\}$ is added to $\mathcal{D}isp$, and the protocol terminates.

VSS	CB	BCB	CR	BCR
$WithoutDispute$	$\ell d + nd + n^2\kappa \log d$	$n\ell + n^2$	4	3
$PerDispute$	0	d	0	4

Note that this protocol can be easily modified to (verifiably) construct multiple sharings of $1 \in \mathbb{F}$, (i.e., the multiplicative identity). We simply require that all $s^{(k)} = 1$ for all $k = 1, \ldots, \ell$ and $u^{(k)} = 1$ for all $k = 1, \ldots, n$, and in step 2.3, P_V checks not only that the sharing is consistent, but that it is a sharing of $\sum_{k=1}^{\ell} r_k + 1$; step 3.1 is similarly altered. Furthermore, in the fault localization section, the players check not only that sharings are consistent, but that they are sharings of the correct values. We refer to this modified protocol by VSS-One.

Lemma 3. *The protocol* VSS *is statistically correct and perfectly private. More explicitly:*

1. *If* VSS *terminates successfully:*
   - *1.1 With overwhelming probability, the values $s^{(1)}, \ldots, s^{(\ell)}$ are correctly shared.*
   - *1.2 With overwhelming probability, for each ordered triple of players (P_i, P_j, P_k) that are not in dispute with one another,[4] P_k has correct authentication tags for the shares sent from P_i to P_j.*
2. *If the protocol terminates with a dispute, then the dispute is new.*
3. *Regardless of how the protocol terminates, the adversary gains no information on the $s^{(1)}, \ldots, s^{(\ell)}$ shared by honest players.*

The protocol VSS-Reconstruct, is used to reconstruct a sharing generated by a single player. The reconstruction protocol used in the main MPC protocol (called LC-Reconstruct) will be used to reconstruct

[4] That is, no pair of players in the triple are in dispute with each other.

linear combinations of sharings that were shared by multiple dealers. Since VSS-Reconstruct is largely the same as the reconstruction protocol in [2], using the authentication and verification tags generated in VSS-Share, it is deferred to Appendix A.3.

Lemma 4. *The pair* VSS *and* VSS-Reconstruct *described above constitute a VSS scheme.*

3.4 Reconstructing Linear Combinations of Sharings

The following protocol is used to reconstruct linear combinations of sharings of secrets that have been shared using VSS. It assumes that each sharing $[w]$ is a sum of sharings $[w^{(1)}] + \cdots + [w^{(n)}]$, where $[w^{(i)}]$ is a linear combination of sharings shared by player P_i. Note that the protocol has some chance of failure. However, whenever the protocol fails, a new player is added to $\mathcal{C}orr$, so it can fail only $O(n)$ times in the entire MPC protocol.

The technique for using the authentication tags in LC-Reconstruct is non-standard, and deserves a bit of explanation. If the initial broadcasting of shares of $[w]$ is inconsistent, then the players open each $[w^{(j)}]$. If $[w^{(j)}]$ is the first such sharing that is inconsistent, then the players will want to use the authentication tags to determine who is lying. However, $[w^{(j)}]$ is a *linear combination* of sharings that were generated with VSS. Each of these initial sharings has authentication tags, but there is no means for combining the tags to get tags for $[w^{(j)}]$.

So the players need to localize which of the sharings in the linear combination $[w^{(j)}] = a_1[s^{(1)}] + \cdots + a_m[s^{(m)}]$ is inconsistent. One way to do this would be to have P_j state which player he accuses of lying and have that player broadcast shares of *each* $[s^{(k)}]$ (or if P_j is corrupt, all players broadcast their shares of each $[s^{(k)}]$). Once this is done, the players could use the tags for whichever share P_j claims is corrupt to determine who was lying. Although this approach would work, it would result in an enormous communication complexity. Therefore, instead of opening all of the $[s^{(k)}]$ all at once, the players use a "divide-and-conquer" technique: Break the sum into two halves, determine which sum has the inconsistency, break that sum in half, and so on until the players reach an individual sharing, at which point they can use the authentication tags.

Protocol: LC-Reconstruct$([w])$

Throughout this protocol, if a player ever refuses to send or broadcast something that the protocol requires, that player is added to $\mathcal{C}orr$, and the protocol terminates.

1. Each $P_i \notin \mathcal{C}orr$ broadcasts his share w_i of $[w]$.
2. If the sharing broadcast in the previous step is consistent, then the players reconstruct w as described in the introduction to VSS-Reconstruct, and the protocol terminates.
3. If the sharing was inconsistent, each $P_i \notin \mathcal{C}orr$ broadcasts his share $w_i^{(j)}$ for each $P_j \in \mathcal{P}$.
4. If any player P_i broadcasted values such that his summands do not match his sum (i.e., if $w_i \neq \sum_{j=1}^{n} w_i^{(j)}$), then all such players are added to $\mathcal{C}orr$, and the protocol terminates.
5. For the lowest j such that the shares of $w^{(j)}$ broadcast in step 3 are inconsistent, one of two steps is performed: If $P_j \notin \mathcal{C}orr$ proceed to step 6. Otherwise, proceed to step 7.
6. $P_j \notin \mathcal{C}orr$
   - 6.1 P_j broadcasts $(\texttt{accuse}, i)$ for the player P_i he believes to have sent an incorrect share.
   - 6.2 Since $[w^{(j)}]$ is a linear combination of sharings dealt by P_j, the players (internally) think of $[w^{(j)}]$ as $a_1[s^{(1)}] + \cdots + a_m[s^{(m)}]$, where each $[s^{(k)}]$ was generated with VSS and each a_k is non-zero. We arrange the $s^{(k)}$'s according to the order in which they were dealt.
   - 6.3 From the sharings $a_1[s^{(1)}], \ldots, a_m[s^{(m)}]$, define two sharings $a_1[s^{(1)}] + \cdots + a_{\lfloor m/2 \rfloor}[s^{(\lfloor m/2 \rfloor)}]$ and $a_{\lfloor m/2 \rfloor + 1}[s^{(\lfloor m/2 \rfloor + 1)}] + \cdots + a_m[s^{(m)}]$. The player P_i accused in step 6.1 broadcasts his share of each of these two sharings.

6.4 If P_i broadcast shares of summands in the previous step that do not match up with the previously sent share of their sum, then P_i is added to $\mathcal{C}orr$, and the protocol terminates.

6.5 Player P_j broadcasts which of the sharings broadcast in step 6.3 he disagrees with. If this is a single sharing $a_k[s^{(k)}]$, then the players proceed to step 6.6. Otherwise, if the sharing is some sum $a_{k_1}[s^{(k_1)}] + \cdots + a_{k_2}[s^{(k_2)}]$, then the players return to step 6.3, but with $a_1[s^{(1)}], \ldots, a_m[s^{(m)}]$ replaced by $a_{k_1}[s^{(k_1)}] + \cdots + a_{k_2}[s^{(k_2)}]$.

6.6 At this point, P_i has broadcast his share of $a_k[s^{(k)}]$, and P_j has broadcast that he disagrees with this share. For each $P_V \notin \mathcal{D}isp_j \cup \mathcal{D}isp_i$, the players invoke Check-Message$(P_i, P_V, \mathbf{s}_i)$, where $\mathbf{s}_i$ is the vector defined in step 1.2 of the invocation of VSS in which $[s^{(k)}]$ was shared.

6.7 If P_i sent shares to P_V in the invocation of Check-Message that do not match with the share of $a_k[s^{(k)}]$, then P_V broadcasts $(\texttt{accuse}, i)$, and $\{P_i, P_V\}$ is added to $\mathcal{D}isp$.

6.8 For each $P_V \notin \mathcal{D}isp_i$ that rejected the message sent by P_i in the invocation of Check-Message, $\{P_i, P_V\}$ is added to $\mathcal{D}isp$. For each P_V that accepted the message, $\{P_j, P_V\}$ is added to $\mathcal{D}isp$.

6.9 At this point, all players are in dispute with either P_i or P_j. By the $Q2$ property of the adversary structure $\mathcal{A}$, this means that one of $\mathcal{D}isp_i$ or $\mathcal{D}isp_j$ is no longer in $\mathcal{A}$. If $\mathcal{D}isp_i \notin \mathcal{A}$ then P_i is added to $\mathcal{C}orr$, and if $\mathcal{D}isp_j \notin \mathcal{A}$, then P_j is added to $\mathcal{C}orr$. Then the protocol terminates.

7. $P_j \in \mathcal{C}orr$

7.1 Since $[w^{(j)}]$ is a linear combination of sharings dealt by P_j, the players (internally) think of $[w^{(j)}]$ as $a_1[s^{(1)}] + \cdots + a_m[s^{(m)}]$, where each $[s^{(k)}]$ was generated with VSS and each a_k is non-zero. We arrange the $s^{(k)}$'s according to the order in which they were dealt.

7.2 From the sharings $a_1[s^{(1)}], \ldots, a_m[s^{(m)}]$, define two sharings $a_1[s^{(1)}] + \cdots + a_{\lfloor m/2 \rfloor}[s^{(\lfloor m/2 \rfloor)}]$ and $a_{\lfloor m/2 \rfloor + 1}[s^{(\lfloor m/2 \rfloor + 1)}] + \cdots + a_m[s^{(m)}]$. Each player not in $\mathcal{C}orr$ broadcasts his share of each of these two sharings.

7.3 Any player who broadcast shares of summands in the previous step that do not match up with the previously sent share of their sum is added to $\mathcal{C}orr$, and the protocol terminates.

7.4 If the players reach this step, then one of the sharings broadcast in step 7.2 is inconsistent. If this is a single sharing $a_k[s^{(k)}]$, then the players proceed to step 7.5. Otherwise, if the sharing is some sum $a_{k_1}[s^{(k_1)}] + \cdots + a_{k_2}[s^{(k_2)}]$, then the players return to step 7.2, but with $a_1[s^{(1)}], \ldots, a_m[s^{(m)}]$ replaced by $a_{k_1}[s^{(k_1)}] + \cdots + a_{k_2}[s^{(k_2)}]$.

7.5 The players invoke VSS-Reconstruct for the sharing $[s^{(k)}]$ decided upon in the last execution of step 7.4 (however, they skip step 1 of VSS-Reconstruct, since shares of $a_k[s^{(k)}]$ have already been broadcast).

7.6 The invocation of VSS-Reconstruct in the previous step will have added a new player to $\mathcal{C}orr$, so the protocol terminates.

LC-Reconstruct	CB	BCB	CR	BCR
$WithoutDispute$	0	d	0	1
$PerDispute$	$n^2\ell + n^2\kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$6 + \log \mathcal{C}$

Lemma 5. *If $[w]$ is a linear combination of sharings generated with* VSS*, then with overwhelming probability, an invocation of* LC-Reconstruct$([w])$ *will either reconstruct the correct value w or add a new player to $\mathcal{C}orr$. Furthermore,* LC-Reconstruct *does not leak any information about* any *sharing other than $[w]$ to the adversary.*

3.5 Generating Random Challenges

The following protocol allows the players to generate a publicly known challenge vector $s^{(1)}, \ldots, s^{(\ell)}$, or the protocol fails (if one of its sub–protocols fails) and outputs a new dispute pair.

Protocol: Generate-Challenges(ℓ)

---

1. Every player $P_i \notin \mathcal{C}orr$ selects a random summand vector $s^{(1,i)}, \ldots, s^{(\ell,i)}$.

2. Call VSS$(P_i, \ell, s^{(1,i)}, \ldots, s^{(\ell,i)})$ to let every $P_i \notin \mathcal{C}orr$ verifiably share his summand vector.

3. Call LC-Reconstruct ℓ times in parallel to reconstruct the sum sharings $\sum_{P_i \notin \mathcal{C}orr} s^{(1,i)}, \ldots, \sum_{P_i \notin \mathcal{C}orr} s^{(\ell,i)}$.

Generate-Challenges	CB	BCB	CR	BCR
$WithoutDispute$	$n\ell d + n^2 d + n^3 \kappa \log d$	$n^2\ell + n^3 + d\ell$	4	4
$PerDispute$	$n^2\ell + n^2 \kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$6 + \log \mathcal{C}$

Lemma 6. *If* Generate-Challenges *terminates successfully, then the reconstructed vector is random. If* Generate-Challenges *terminates unsuccessfully, then a new dispute is found.*

3.6 Generating Multiplication Triples

The following protocol allows the players to verifiably generate random sharings of triples (a, b, c) such that $ab = c$. The idea is that a random $a^{(k)}$ is generated, and then each P_i is "responsible for" creating a random triple $a^{(k)}b^{(i,k)} = c^{(i,k)}$. To verify correctness, the P_i also creates a triple $a^{(k)}\widetilde{b}^{(i,k)} = \widetilde{c}^{(i,k)}$, and this is used to mask an opening of $a^{(k)}b^{(i,k)} - c^{(i,k)}$. Once all these triples are checked, the final triple is defined to be $(a^{(k)}, \sum_{i=1}^n b^{(i,k)}, \sum_{i=1}^n c^{(i,k)})$.

Protocol: Multiplication-Triple(ℓ)

1. Generating Triples

1.1 Each $P_i \notin \mathcal{C}orr$ invokes VSS($P_i, 2\ell n + 3\ell$) to generate uniformly random sharings and VSS-One($P_i, 2\ell n$) to generate sharings of $1 \in \mathbb{F}$; these invocations are done in parallel. Denote the random sharings of player P_i by $([a^{(i,1)}], \ldots, [a^{(i,\ell)}])$, $([b^{(i,1)}], \ldots, [b^{(i,\ell)}])$, $([\widetilde{b}^{(i,1)}], \ldots, [\widetilde{b}^{(i,\ell)}])$, and $\{([r^{(i,j,1)}], \ldots, [r^{(i,j,\ell)}]), ([\widetilde{r}^{(i,j,1)}], \ldots, [\widetilde{r}^{(i,j,\ell)}])\}_{j=1}^n$ and the sharings of ones by $\{([1^{(i,j,1)}], \ldots, [1^{(i,j,\ell)}]), ([\widetilde{1}^{(i,j,1)}], \ldots, [\widetilde{1}^{(i,j,\ell)}])\}_{j=1}^n$. The sharings of players in $\mathcal{C}orr$ are defined to be all-zero sharings.

1.2 For each $k = 1, \ldots, \ell$ and each i such that $P_i \notin \mathcal{C}orr$, the players define and locally compute

$$[a^{(k)}] = \textstyle\sum_{m=1}^n [a^{(m,k)}]$$

$$[r^{(i,k)}] = \textstyle\sum_{m=1}^n [r^{(i,m,k)}]$$

$$[1^{(i,k)}] = \textstyle\sum_{m=1}^n [1^{(m,i,k)}] + w[1^{(i,i,k)}],$$

where $w \in \mathbb{F}$ is the unique element that makes $[1^{(i,k)}]$ a sharing of 1. The sharings $[\widetilde{r}^{(i,k)}]$ and $[\widetilde{1}^{(i,k)}]$ are similarly defined.

1.3 Each $P_j \notin \mathcal{C}orr$ sends his share of $[a^{(k)}][b^{(i,k)}] + [r^{(i,k)}][1^{(i,k)}]$ and $[a^{(k)}][\widetilde{b}^{(i,k)}] + [\widetilde{r}^{(i,k)}][\widetilde{1}^{(i,k)}]$ to $P_i \notin \mathcal{C}orr$ for each $k = 1, \ldots, \ell$. (The shares of players in $\mathcal{C}orr$ will be Kudzu shares, so P_i knows those shares as well.)

1.4 Each $P_i \notin \mathcal{C}orr$ applies the recombination vector $\boldsymbol{\lambda}$ to the shares of $D^{(i,k)} = a^{(k)}b^{(i,k)} + r^{(i,k)}$ and $\widetilde{D}^{(i,k)} = a^{(k)}\widetilde{b}^{(i,k)} + \widetilde{r}^{(i,k)}$ received in the previous step to compute $D^{(i,k)}$ and $\widetilde{D}^{(i,k)}$ for each $k = 1, \ldots, \ell$.

1.5 Each P_i broadcasts $D^{(i,k)}$ and $\widetilde{D}^{(i,k)}$ for each $k = 1, \ldots, \ell$.

1.6 Each player locally computes $[c^{(i,k)}] = D^{(i,k)} - [r^{(i,k)}]$ and $[\widetilde{c}^{(i,k)}] = \widetilde{D}^{(i,k)} - [\widetilde{r}^{(i,k)}]$ (using the canonical sharings of $D^{(i,k)}$ and $\widetilde{D}^{(i,k)}$).

2. Error Detection

2.1 The players invoke Generate-Challenges(ℓ) to generate a random vector $(s^{(1)}, \ldots, s^{(\ell)})$.

2.2 Each player not in $\mathcal{D}isp_i$ broadcasts his share of $[\widehat{b}^{(i,k)}] = [\widetilde{b}^{(i,k)}] + s^{(i)}[b^{(i,k)}]$ for each $i = 1, \ldots, n$ and each $k = 1, \ldots, \ell$.

2.3 If the sharing of some $[\widehat{b}^{(i,k)}]$ broadcast in the previous step is inconsistent, P_i broadcasts $(\texttt{accuse}, P_j)$ for some $P_j \notin \mathcal{D}isp_i$ who broadcasted an incorrect share, then $\{P_i, P_j\}$ is added to $\mathcal{D}isp$ and the protocol terminates.

2.4 The players invoke multiple instances of LC-Reconstruct in parallel to reconstruct $z^{(i,k)} = [a^{(k)}]\widehat{b}^{(i,k)} - [\widetilde{c}^{(i,k)}] - s^{(k)}[c^{(i,k)}]$ for each $i = 1, \ldots, n$ and each $k = 1, \ldots, \ell$.

2.5 If all the $z^{(i,k)}$ reconstructed in the previous step are zero, then we define

$$[b^{(k)}] = \sum_{m=1}^{n} [b^{(m,k)}]$$

$$[c^{(k)}] = \sum_{m=1}^{n} [c^{(m,k)}],$$

and the protocol terminates successfully with the multiplication triples taken to be $(a^{(k)}, b^{(k)}, c^{(k)})$ for $k = 1, \ldots, \ell$.

3. *Fault Localization*

If any $z^{(i,k)}$ reconstructed in step 2.4 is not zero, the following is done for the lexicographically lowest pair (i, k) such that $z^{(i,k)} \neq 0$.

3.1 Each P_j broadcasts his share of $[a^{(m,k)}]$, $[\tilde{r}^{(m,i,k)}]$, and $[r^{(m,i,k)}]$ for each $P_m \notin P_j$.

3.2 If P_i sees that the shares of some $P_j \notin \mathcal{D}isp_i$ sent in the previous step are inconsistent with the share sent in step 1.3 or 2.4, then P_i broadcasts $(\texttt{accuse}, P_j)$; then $\{P_j, P_i\}$ is added to $\mathcal{D}isp$ and the protocol terminates.

3.3 Each P_m examines the shares broadcast in the previous step of all sharings that P_m generated. If P_m notices that some $P_j \notin \mathcal{D}isp_m$ broadcast an incorrect share in the previous step, then P_m broadcasts $(\texttt{accuse}, P_j)$; then $\{P_m, P_j\}$ is added to $\mathcal{D}isp$ and the protocol terminates.

3.4 If no P_m broadcast an accusation in the previous step, then P_i is added to $\mathcal{C}orr$ and the protocol terminates.

Multiplication-Triple	CB	BCB	CR	BCR
$WithoutDispute$	$n^2\ell d + n^3\kappa \log d$	$n^3\ell + n\ell d$	9	11
$PerDispute$	$n^2\ell + n^2\kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$6 + \log \mathcal{C}$

Lemma 7. *If* Multiplication-Triple *terminates unsuccessfully, then a new dispute is localized. If* Multiplication-Triple *succeeds, then it maintains statistical correctness and perfect privacy. That is, with overwhelming probability, at the end of the protocol the players hold sharings of ℓ multiplication triples (a, b, c) with $c = ab$; in addition, the adversary has no information on a, b, or c (other than that $c = ab$).*

3.7 Preparation Phase

The following protocol generates the multiplication triples for the multiplication gates and random sharings for random gates. The task is broken into n^2 segments. The number of multiplication triples and random sharings generated in each segment are denoted by L_M and L_R (respectively), and we require $L_M \leq \lceil c_M/n^2 \rceil$ and $L_R \leq \lceil (c_R)/n^2 \rceil$.

Protocol: Preparation-Phase

Initialize $\mathcal{C}orr$ and $\mathcal{D}isp$ to the empty set. For each segment handling L_M multiplication gates and L_R random gates, the following steps are performed. If any of the subprotocols fails, then the segment is repeated.

1. Invoke Multiplication-Triple(L_M). Assign one multiplication triple to each multiplication gate in this segment.
2. Each $P_i \notin \mathcal{C}orr$ invokes VSS($P_i, L_R, r^{(1,i)}, \ldots, r^{(L_R,i)}$), sharing uniformly random values. (The sharings of corrupt players are defined to be all-zero sharings.)
3. We define L_R random sharings by $[r^{(k)}] = \sum_{i=1}^{n} [r^{(k,i)}]$ for each $k = 1, \ldots, L_R$. Assign one random sharing to each random gate in this segment.

Preparation-Phase	CB	BCB	CR	BCR
$WithoutDispute$	$n^2(c_M + c_R)d + n^5\kappa \log d$	$n^3(c_M + c_R) + n(c_M + c_R)d$	$13n^2$	$14n^2$
$PerDispute$	$(c_M + c_R) + n^2\kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$6 + \log \mathcal{C}$

3.8 Input Phase

The goal of the input phase is to allow each player to share their inputs. We denote the number of inputs in a given segment by L. We require $L \leq \lceil c_I/n^2 \rceil$, and we also require that each segment contain inputs from only one player.

Protocol: Input-Phase

For each segment, the following steps are executed to let the dealer $P_D \notin \mathcal{C}orr$ verifiably share L inputs $s^{(1)}, \ldots, s^{(L)}$. If some invocation of VSS fails, then the segment fails and is repeated.

1. Each $P_i \notin \mathcal{C}orr$ invokes $\mathsf{VSS}(P_i, L, r^{(1,i)}, \ldots, r^{(L,i)})$, sharing uniformly random values. (The sharings of corrupt players are defined to be all-zero sharings.)
2. We define L random sharings by $[r^{(k)}] = \sum_{i=1}^{n} [r^{(k,i)}]$ for each $k = 1, \ldots, L$. Assign one random sharing to each input gate in this segment.
3. Each $P_i \notin \mathcal{D}isp_D$ sends his share of each $[r^{(k)}]$ to P_D.
4. If P_D finds that one of the sharings was inconsistent, he broadcasts the index of this sharing, and the following steps are performed. If they are all consistent, then the players proceed to step 5.
   - *4.1* If P_D indicated that the random sharing $[r]$ was inconsistent, then each $P_i \notin \mathcal{D}isp_D$ broadcasts their share of $[r]$.
   - *4.2* If P_D sees that some P_i broadcast a different share than was sent privately, then P_D broadcasts $(\texttt{accuse}, i)$, $\{P_D, P_i\}$ is added to $\mathcal{D}isp$, and the segment fails and is repeated.
   - *4.3* The players invoke LC-Reconstruct to reconstruct $[r]$ (but skipping the first step, because shares of $[r]$ have already been broadcast).
   - *4.4* Since the sharing $[r]$ was inconsistent, the invocation of LC-Reconstruct in the previous step will have located a new corrupt player, so the segment fails and is repeated.
5. Using the method for reconstructing secrets described in the introduction to VSS-Reconstruct, P_D computes the random value r associated with each of his L input gates in this segment.
6. For each input gate with input s and random sharing $[r]$, P_D broadcasts $s - r$.
7. For each $s - r$ broadcast in the previous step, each player locally computes $s - r + [r]$ (using the canonical sharing of $s - r$) as the sharing for that input gate. Since each player is storing each share as a sum of shares (one from each player), we update $[r]$ by adding the canonical sharing of $s - r$ to $[r^{(D)}]$ and leaving $[r^{(i)}]$ the same for $i \neq D$. In the dealer failed to broadcast a value for an input gate, or if the dealer was already in $\mathcal{C}orr$, then the sharing for that gate is taken to be $[r]$.

Input-Phase	CB	BCB	CR	BCR
$WithoutDispute$	$nc_I d + n^4 d + n^5 \kappa \log d$	$n^2 c_I + n^5$	$5n^2$	$4n^2$
$PerDispute$	$c_I + n^2 \kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$9 + \log \mathcal{C}$

3.9 Computation Phase

After the circuit preparation has been done, the computation phase is just a matter of opening linear combinations of sharings and possibly resolving disputes.

Each affine gate is computed by performing local computations. Each multiplication gate is computed by opening affine combinations of known sharings. Each output gate is computed by publicly opening it.[5] This means that the computation phase will consist of local operations and $c_M + c_O$ public openings.

The circuit will be divided into segments and evaluated one segment at a time. The segments will be constructed such that each segment has no more than $\lceil (c_M + c_O)/n^2 \rceil$ gates, and a single segment only

[5] We assume that each player receives all the outputs, although the protocol could easily be modified to allow for private outputs.

contains gates from one multiplicative layer of the circuit. This means that if $\mathcal{D}$ is the multiplicative depth of the circuit, then there are at most $n^2 + \mathcal{D}$ segments. Each affine gate will be included in the first possible segment in which it can be evaluated.

If a fault occurs in some segment (which is to say that one of the opened sharings is inconsistent), then one or more new disputes are localized, and the segment is repeated.

It is important to remember that all sharings generated by VSS and Multiplication-Triple are sums of sharings such that one summand comes from each player. Since all sharings opened are affine combinations of these, this means that every sharing we will be opening in the computation phase is a sum of sharings with one summand coming from each player. Thus the protocol LC-Reconstruct can be performed.

Protocol: Computation-Phase

For each segment with L reconstructions, the following steps are executed. If one of the reconstructions is inconsistent, then a new dispute is found, and the segment is repeated.

1. For each affine gate in the segment, the players evaluate the gate by local computations.
2. The players invoke LC-Reconstruct multiple times in parallel for each output gate in the segment.
3. For each multiplication gate in the segment with inputs $[x]$ and $[y]$ and associated multiplication triple $([a], [b], [c])$, the following steps are performed in parallel.
   - 3.1 In parallel with step 2, the players invoke LC-Reconstruct$([x - a])$ and LC-Reconstruct$([y - b])$.
   - 3.2 The players assign the sharing $(x - a)(y - b) - (x - a)[b] - (y - b)[a] + [c]$ as the output of the gate.

Computation-Phase	CB	BCB	CR	BCR
$WithoutDispute$	0	$\mathcal{C}d$	0	$\mathcal{D}$
$PerDispute$	$\mathcal{C}n + n^3 + n^2\kappa \log d$	$n^2 + nd + d \log \mathcal{C}$	2	$6 + \log \mathcal{C}$

3.10 Putting it All Together

We perform the MPC protocol by invoking Preparation-Phase, Input-Phase, and Computation-Phase in succession. Note that there is a term n added to the number of communication rounds to account for the fact that when a player is corrupted, all players will broadcast their shares sent by that player.

Theorem 1. *A set of n players communicating over a secure synchronous network can evaluate an agreed function of their inputs securely against an active, adaptive adversary with an arbitrary Q^2 adversary structure $\mathcal{A}$ with point-to-point communication bandwidth $O(n^2\mathcal{C}d + n^4 d + n^5 \kappa \log d)$ and broadcast bandwidth $O(n^3\mathcal{C} + n\mathcal{C}d + n^5 + n^3 d + n^2 d \log \mathcal{C})$, taking $20n^2$ communication rounds and $27n^2 + \mathcal{D} + n^2 \log \mathcal{C}$ broadcast rounds. Here, d is the number of rows in the smallest MSP (with multiplication) representing $\mathcal{A}$, and κ is the size of an element of $\mathbb{F}$.*

References

[1] Donald Beaver and Avishai Wool. Quorum-based secure multi-party computation. In *EUROCRYPT*, pages 375–390, 1998.

[2] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer, 2006.

[3] Ronald Cramer, Ivan Damågard, and Ueli Maurer. Span programs and general multi-party computation. *Preliminary version appeared as BRICS tech. report*, BRICS-RS-97-28, 1998.

[4] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, pages 311–326, 1999.

[5] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. General adversaries in unconditional multi-party computation. In *ASIACRYPT*, pages 232–246, 1999.

[6] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in general multiparty computations. *Proc. PODC*, 1997.

[7] Adam Smith and Anton Stiglic. Multiparty computation unconditionally secure against Q^2 adversary structures. *CoRR*, cs.CR/9902010, 1999.

A.1 Proofs

Proof of Lemma 1. See [3] for the proof that the scheme is secure without the use of Kudzu shares (i.e., the case in which $r_2, \ldots, r_e$ are chosen completely randomly).

To see that using Kudzu shares does not compromise security, note that if two players are in dispute, then at least one of them is corrupt, so the adversary will know the zero-share anyway. Thus making the share public knowledge doesn't give the adversary any more information than he already had. □

Proof of Lemma 3.

1. Assuming VSS terminates successfully:
   - *1.1* We are guaranteed that there is at least one honest $P_V \notin \mathcal{D}isp_D$ (because otherwise P_D would be in $\mathcal{C}orr$, and would not be sharing any values). So we are guaranteed that there is at least one random vector $(r_1, \ldots, r_\ell)$ such that $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ will be opened toward an honest player. Let R denote the subspace of $\mathbb{F}^\ell$ defined as the set of vectors $(r_1, \ldots, r_\ell)$ such that $\sum_{k=1}^{\ell} r_k[s^{(k)}]$ is a consistent sharing.[6] Let $\mathbf{r} = (r_1, \ldots, r_\ell)$ be some vector such that $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ is a consistent sharing. Then the set of all vectors $(r_1, \ldots, r_\ell)$ such that $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ is a consistent sharing is $\mathbf{r} + R$.

     Now if some $[s^{(k)}]$ is inconsistent, R will be a proper subspace of $\mathbb{F}^\ell$. In this case, the probability that the sharing $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ is inconsistent is $|\mathbf{r} + R|/|\mathbb{F}^\ell| \leq \kappa^{\ell-1}/\kappa^\ell = 1/\kappa$, which is negligible.
   - *1.2* The correctness of the authentication tags follows from the correctness of Distribute-Tags.
2. The fact that any localized dispute is new follows from inspection of the protocol.
3. The only information the adversary receives are the corrupt players' shares of each $s^{(k)}$ and all the shares of $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$. Since the underlying secret sharing protocol is perfectly private (Lemma 1), the shares of $s^{(k)}$ give the adversary no information on $s^{(k)}$, and since the sharing $[u^{(V)}]$ is random, the shares of $\sum_{k=1}^{\ell} r_k[s^{(k)}] + [u^{(V)}]$ are independent of the $s^{(k)}$.

□

Proof of Lemma 4.

1. *Termination*

   The fact that VSS satisfies the termination property of definition 1 follows from Lemma 3.

[6] It may be the case that $R = \{0\}$.

In VSS-Reconstruct, the players either execute step 6 or step 7. In the last sub-step of step 6, if the shares of non-corrupt players are consistent, then the protocol terminates successfully, and otherwise the players move on to step 7. So to prove that the honest players always complete VSS-Reconstruct, it suffices to prove the claim in the last sub-step of step 7 (which is sub-step 7.3); namely, we want to show that at step 7.3, any player who broadcast an incorrect share in the first step of VSS-Reconstruct will be in $\mathcal{C}orr$. So suppose some P_i sent an incorrect share in step 1. If P_i sends the same share to some $P_m \notin \mathcal{D}isp_i$ in step 7.1, then by Lemmas 2 and 3, P_m will label him corrupt. If P_i sends a different share in step 7.1, then P_m accuse P_i in step 7.2. This means that any P_i who broadcast an incorrect share in step 1 will now be in dispute will all honest players. Since the set of honest players is not in $\mathcal{A}$, this means that P_i will be labeled corrupt. So only good shares are used in step 7.3, meaning that the protocol terminates successfully.

2. *Privacy*

   Privacy of a secret w before *any* execution of VSS-Reconstruct follows from Lemma 3. In addition, we need to show that invoking VSS-Reconstruct for some secret does not reveal any information about other secrets. This is because shares of other secrets are sent during invocations of Check-Message. However, note that whenever Check-Message is invoked, the person sending the shares is always in dispute with the dealer, (which will certainly be the case if the dealer is in $\mathcal{C}orr$). Since the dealer and the share-holder are in dispute, one of them is corrupt, so the adversary already knows all shares sent from the dealer to the share-holder, and revealing these shares to other players does not give the adversary any additional information.

3. *Correctness*

   It follows from Lemma 3 that at the end of VSS, each of the sharings is consistent, which means that the vector of shares is in the span of the MSP matrix M. Any vector in the span of M corresponds to a sharing of a unique value, and thus there is a unique value r such that the vector of shares is a sharing of r. Furthermore, if the dealer is honest, then it is clear that r is the value that was supposed to be shared.

   Now we must verify that the honest players reconstruct r at the end of VSS-Reconstruct. It is clear from inspection of VSS-Reconstruct that no matter how the protocol ends, the players always end up reconstructing from a consistent set of shares. Also note that this reconstruction uses the shares of all honest players (including Kudzu shares), because honest players broadcast correct shares in step 1, and even if the dealer accuses them of lying in step 3, they will be vindicated in step 6.4. Since the shares of honest players are sufficient to reconstruct a sharing, all honest players will reconstruct r.

□

Proof of Lemma 5. It is clear that if LC-Reconstruct succeeds, it will reconstruct the correct value, because in that case the shares broadcast in the first step were consistent. The fact that unsuccessful termination leads to finding a new corrupt player follows by examining steps 6.9 and 7.6.

To show that no information about any sharing other than $[w]$ is leaked to the adversary, note that in step 6, the only shares that are opened are shares held by a player that the dealer accuses of lying, and so the adversary already knew those shares; in step 7, the shares revealed are those dealt by a corrupt dealer, so again, the adversary learns nothing new. □

Proof of Lemma 6. Since at least one player is honest, at least one summand vector $s^{(1,i)}, \ldots, s^{(\ell,i)}$ will be completely random. Thus the sum vector will be random.

The only way in which Generate-Challenges might terminate unsuccessfully is if VSS or LC-Reconstruct terminates unsuccessfully, in which case a new dispute will be found. □

Proof of Lemma 7. That a new dispute is localized when Multiplication-Triple terminates unsuccessfully is clear from inspection of the protocol.

Privacy of $a^{(k)}$ follows from the fact that it is a sum of sharings, one sharing coming from each player. The only time that a sharing involving $a^{(k)}$ is opened is in steps 1.3 and 2.4, and in both cases it is masked (with $[r^{(i,k)}][1^{(i,k)}]$ and $[\widetilde{c}^{(i,k)}]$, respectively).

Privacy of $b^{(k)}$ follows from the fact that it is a sum of sharings, one sharing coming from each player. Sharings involving each of the summands $b^{(i,k)}$ are only opened in steps 1.3 and 2.2; in step 1.3, the sharing is opened to the player who generated it, and in step 2.2, the sharing is masked by $\widetilde{b}^{(i,k)}$.

Privacy of $c^{(k)}$ follows from the fact that it is a sum of sharings, one sharing coming from each player, and that the only time that a sharing involving a $c^{(i,k)}$ is opened is in step 2.4, where it is masked with $\widetilde{c}^{(i,k)}$.

With overwhelming probability, the invocations of VSS used to construct the sharings in the first step generate valid sharings, and since $a^{(k)}$, $b^{(k)}$, and $c^{(k)}$ are affine combinations of these sharings, they are valid sharings. In order to show that $c^{(k)} = a^{(k)}b^{(k)}$, it suffices to show that $c^{(i,k)} = a^{(k)}b^{(i,k)}$ for each i. In particular, we want to show that if $z^{(i,k)}$ reconstructed in step 2.4 is zero, then with overwhelming probability, $c^{(i,k)} = a^{(k)}b^{(i,k)}$. Note that $z^{(i,k)} = 0$ if and only if $s^{(k)}(c^{(i,k)} - a^{(k)}b^{(i,k)}) = -(\widetilde{c}^{(i,k)} - \widetilde{a}^{(k)}\widetilde{b}^{(i,k)})$. If $c^{(i,k)} - a^{(k)}b^{(i,k)} \neq 0$, then since $s^{(k)}$ is random, $z^{(i,k)}$ could only be zero with negligible probability. □

A.2 The Information Checking Protocols

We now give the information checking protocols explicitly.

Protocol: Distribute-Tags$(P_S, P_R, P_V, (m^{(1)}, \ldots, m^{(\ell)}))$

We call the sender P_S, the receiver P_R, and the verifier P_V. We assume that a vector of messages $\mathbf{m} = (m^{(1)}, \ldots, m^{(\ell)}) \in \mathbb{F}^\ell$ (with $\ell \leq d$) has already be sent from the P_S to P_R. We also assume there is an extension field $\mathbb{G}$ of $\mathbb{F}$ such that $\mathbb{G}$ has minimal size subject to $|\mathbb{G}| \geq d|\mathbb{F}|$. (The field $\mathbb{G}$ is fixed throughout the *entire* MPC protocol.)

1. *Generating Tags:*
   - *1.1* P_S picks 2κ random elements $y_1, \ldots, y_\kappa, u_1, \ldots, u_\kappa \in \mathbb{G}$.
   - *1.2* For each $i = 1, \ldots, \kappa$, P_S determines v_i such that the points $(0, y_i), (1, m^{(1)}), \ldots, (\ell, m^{(\ell)}), (u_i, v_i)$ lie on a degree ℓ polynomial (over $\mathbb{G}$).
   - *1.3* P_S sends the authentication tags $y_1, \ldots, y_\kappa$ to P_R and the verification tags $z_i = (u_i, v_i)$ to P_V for each i.
2. *Fault Detection:*
   - *2.1* P_V partitions the set $\{1, \ldots, \kappa\}$ into sets I and $\overline{I}$ of almost equal size ($\big||I| - |\overline{I}|\big| \leq 1$) and sends $\{z_i\}_{i \in I}$ to P_R.
   - *2.2* P_R checks for each z_i sent by P_V that the points $(0, y_i), (1, m^{(1)}), \ldots, (\ell, m^{(\ell)}), (u_i, v_i)$ lie on a polynomial of degree ℓ. If any one of these checks fails, P_R broadcasts (`reject`). Otherwise, P_R broadcasts (`accept`).
3. *Fault Localization:*
   If P_R broadcasted (`reject`) above, then the following steps are executed.
   - *3.1* P_R picks one z_i that failed the check in step 2.2 and broadcasts (i, s, z_i).
   - *3.2* P_V and P_S broadcast z_i.
   - *3.3* If the value broadcasted by P_S and P_V differ, then $\{P_S, P_V\}$ is added to $\mathcal{D}isp$. If the value broadcasted by P_R and P_V differ, then $\{P_R, P_V\}$ is added to $\mathcal{D}isp$. Otherwise, $\{P_R, P_S\}$ is added to $\mathcal{D}isp$.

Distribute-Tags	CB	BCB	CR	BCR
$WithoutDispute$	$\kappa \log d$	1	2	1
$PerDispute$	0	$\log d$	0	2

In the execution of the protocol, a dispute may arise between the sender and receiver as to what the value of the vector of messages was. The vector is revealed to the verifier using Check-Message, and then the verifier either confirms or denies what the receiver claims to have received.

Protocol: Check-Message$(P_R, P_V, (m^{(1)}, \ldots, m^{(\ell)}))$

---

1. The receiver sends the vector of messages $\mathbf{m} = (m^{(1)}, \ldots, m^{(\ell)})$ along with authentication tags $\{y_i\}_{i \in \overline{I}}$ to P_V.
2. The verifier checks that the points $(0, y_i), (1, m^{(1)}), \ldots, (\ell, m^{(\ell)}), (u_i, v_i)$ all lie on a polynomial of degree ℓ for each $i = \overline{I}$. If *any one* of these $|\overline{I}|$ checks passes, then the verifier broadcasts (`accept`). Otherwise, the verifier broadcasts (`reject`).

Check-Message	CB	BCB	CR	BCR
$WithoutDispute$	$\ell + \kappa \log d$	1	1	1
$PerDispute$	0	0	0	0

Proof of Lemma 2. The proof is the same as in [2] with the exception of claim 3, whose proof is as follows: The probability that P_R could guess a point $(0, y_i)$ that worked for some $\mathbf{m}' \neq \mathbf{m}$ is no more than $\kappa/(|\mathbb{G}| - \ell - 1) \leq \kappa/(d2^\kappa - \ell - 1) \leq \kappa/(d(2^\kappa - 1) - 1)$, which is negligible. □

A.3 The VSS Reconstruct Protocol

Suppose we want to reconstruct a secret using the shares of some set A of players satisfying $A \notin \mathcal{A}$. By the definition of an MSP, this meas that $\mathbf{a} \in \operatorname{Im} M_A^\top$. So there is some vector $\boldsymbol{\omega}_A$ satisfying $M_A^\top \boldsymbol{\omega}_A = \mathbf{a}$. If $[w]_A$ represents the shares of $[w]$ held by players in A and $\mathbf{s} = (w, r_2, \ldots, r_e)^\top$ represents the vector used in Share to generate the sharing $[w]$, then we can reconstruct the secret as

$$\langle \boldsymbol{\omega}_A, [w]_A \rangle = \langle \boldsymbol{\omega}_A, M_A \mathbf{s} \rangle = \langle M_A^\top \boldsymbol{\omega}_A, \mathbf{s} \rangle = \langle \mathbf{a}, \mathbf{s} \rangle = w.$$

This is the reconstruction procedure used in the following protocol.

Protocol: VSS-Reconstruct$([w])$

We assume that the sharing $[w]$ has been shared by a dealer P_D using VSS.

1. Each player not in $\mathcal{C}orr$ that holds a non-Kudzu share of $[w]$ broadcasts their share.
2. If the shares broadcast in the previous step and the Kudzu shares form a consistent sharing (that is, they are in the span of $M_{\mathcal{P}-\mathcal{C}orr}$), then the secret w is reconstructed as described in the introduction to this protocol, and the protocol terminates.
3. If the shares broadcast in step 1 form an inconsistent sharing, then P_D broadcasts the index i of *each* player he accuses of sending an incorrect share.
4. If P_D did not broadcast an index in the previous step, then P_D is added to $\mathcal{C}orr$ (if he is not already in $\mathcal{C}orr$). Also, if we remove the shares P_D accused of being corrupted, and the remaining shares are still inconsistent, then P_D is added to $\mathcal{C}orr$. Lastly, if the set of players in dispute with P_D is no longer in $\mathcal{A}$, then P_D is added to $\mathcal{C}orr$.
5. If $P_D \notin \mathcal{C}orr$, then proceed to step 6. Otherwise, proceed to step 7.
6. *Dealer not in Corr*
   - *6.1* For each player P_i accused by P_D in step 3, the players invoke Check-Message$(P_i, P_m, \mathbf{s}_i)$ for each player $P_m \notin \mathcal{D}isp_i \cup \mathcal{D}isp_j$, where $\mathbf{s}_i$ is the vector defined in step 1.2 of the invocation of VSS in which $[w]$ was shared.
   - *6.2* For any P_i who sent a share to P_m that was different than the share broadcast in step 1, P_m broadcasts $(\texttt{accuse}, i)$, and $\{P_m, P_i\}$ is added to $\mathcal{D}isp$.

6.3 For each $P_m \notin \mathcal{D}isp_i$ that rejected the message sent by P_i in the invocation of Check-Message, $\{P_i, P_m\}$ is added to $\mathcal{D}isp$. For each P_m that accepted the message, $\{P_D, P_m\}$ is added to $\mathcal{D}isp$.

6.4 At this point, any P_i who was accused by P_D and who broadcast an incorrect share in step 1 will have been accused by all honest $P_m \notin \mathcal{D}isp_i$, meaning that P_i will be added to $\mathcal{C}orr$. Similarly, if P_i was accused by P_D but broadcast a correct share in step 1, then P_D will have been accused by all honest $P_m \notin \mathcal{D}isp_D$, meaning that P_D will be added to $\mathcal{C}orr$.

6.5 If the shares of players not in $\mathcal{C}orr$ (together with the Kudzu shares) form a consistent sharing, then those shares are used to reconstruct w. If those shares are inconsistent, then the dealer is added to $\mathcal{C}orr$, and the players proceed to step 7.

7. Dealer in $\mathcal{C}orr$

7.1 For all P_i that do not hold Kudzu-shares and for all $P_m \notin \mathcal{D}isp_i$, the players invoke Check-Message$(P_i, P_m, \mathbf{s}_i)$, where $\mathbf{s}_i$ is the vector defined in step 1.2 of the invocation of VSS in which $[w]$ was shared.

7.2 For any P_i who sent a share to P_m that was different than the share broadcast in step 1, P_m broadcasts $(\texttt{accuse}, i)$, and $\{P_m, P_i\}$ is added to $\mathcal{D}isp$.

7.3 At this point, any P_i who broadcast an incorrect share in step 1 will have been accused by all $P_m \notin \mathcal{D}isp_i$, meaning that P_i will be added to $\mathcal{C}orr$. The shares of players not in $\mathcal{C}orr$ are now used to reconstruct w.

VSS-Reconstruct	CB	BCB	CR	BCR
$WithoutDispute$	0	d	0	1
$PerDispute$	$n^2\ell + n^2\kappa \log d$	n^2	2	5