Efficient Multi-Party Computation with Dispute Control*

Zuzana Beerliová-Trubíniová and Martin Hirt

ETH Zurich, Department of Computer Science, CH-8092 Zurich
{bzuzana,hirt}@inf.ethz.ch

Abstract. Secure multi-party computation (MPC) allows a set of n players to securely compute an agreed function of their inputs, even when up to t players are under the control of an (active or passive) adversary. In the information-theoretic model MPC is possible if and only if $t < n/2$ (where active security with $t \geq n/3$ requires a trusted key setup).

Known passive MPC protocols require a communication of $\mathcal{O}(n^2)$ field elements per multiplication. Recently, the same communication complexity was achieved for active security with $t < n/3$. It remained an open question whether $\mathcal{O}(n^2)$ complexity is achievable for $n/3 \leq t < n/2$.

We answer this question in the affirmative by presenting an active MPC protocol that provides optimal ($t < n/2$) security and communicates only $\mathcal{O}(n^2)$ field elements per multiplication. Additionally the protocol broadcasts $\mathcal{O}(n^3)$ field elements *overall*, for the whole computation.

The communication complexity of the new protocol is to be compared with the most efficient previously known protocol for the same model, which requires *broadcasting* $\Omega(n^5)$ field elements per multiplication. This substantial reduction in communication is mainly achieved by applying a new technique called *dispute control*: During the course of the protocol, the players keep track of disputes that arise among them, and the ongoing computation is adjusted such that known disputes cannot arise again. Dispute control is inspired by the player-elimination framework. However, player elimination is not suited for models with $t \geq n/3$.

1 Introduction

1.1 Background

Secure multi-party computation (MPC) enables a set of n players to securely evaluate an agreed function of their inputs even when t of the players are corrupted by a central adversary. A *passive adversary* can read the internal state of the corrupted players, trying to obtain information about the honest players' inputs. An *active adversary* can additionally make the corrupted players deviate from the protocol, trying to falsify the outcome of the computation.

* This work was partially supported by the Zurich Information Security Center. It represents the views of the authors.

S. Halevi and T. Rabin (Eds.): TCC 2006, LNCS 3876, pp. 305-328, 2006.

The MPC problem dates back to Yao [Yao82]. The first generic solutions presented in [GMW87,CDG87,GHY87] were based on cryptographic intractability assumptions. Later, MPC protocols with information-theoretic security were developed [BGW88,CCD88,RB89,Bea91b], which is the focus of this work.

Information-theoretic security against a passive or active adversary is possible if and only if $t < n/2$. The protocols with active security require broadcast channels, which can be simulated from scratch for $t < n/3$ [PSL80,BGP92,CW92], and can be simulated when a trusted key setup is available for $t < n$ [DS82,PW92].[1]

The communication complexity of MPC is measured in bits sent by honest parties. The function to be computed is represented as an arithmetic circuit over a finite field (with additions and multiplications). The classical MPC protocol with passive security (for $t < n/2$) requires a communication of $\mathcal{O}(n^2)$ field elements per multiplication [BGW88]. Recently, the same communication complexity was achieved for active security, including the costs for simulating the broadcast channels [HM01]; however, this protocol is only suitable for $t < n/3$. The most efficient actively secure MPC protocol for $t < n/2$ requires *broadcasting* $\Omega(n^5)$ field elements per multiplication [CDD+99], and each of these broadcasts must be simulated with an expensive broadcast protocol [PW92].

1.2 Contributions

In this work, we show that information-theoretic MPC with adaptive active security for $t < n/2$ is achievable with sending $\mathcal{O}(n^2)$ field elements per multiplication, and broadcasting $\mathcal{O}(n^3)$ field elements *overall*, for the whole computation. This improves on previous protocols which require *broadcasting* $\Omega(n^5)$ field elements per multiplication [CDD+99].

This result is of particular theoretical interest, as it shows that for all t for which information-theoretic MPC is possible, i.e., $t < n/2$, (adaptive) active security is achievable at essentially the same costs as passive security. This extends the result of [HM01], where only the range $t < n/3$ could be solved. The achieved communication complexity might well be optimal, as even in the passive model it seems unavoidable that for each multiplication gate, every player sends a value to every other player.

The following table summarizes the communication complexities of known and new MPC protocols, where κ denotes the security parameter (i.e., the bit-length of a field element), $\mathcal{BC}(\cdot)$ the number of broadcasted bits, and c_M the number of multiplication gates in the circuit. For simplicity, we assume that the function takes n inputs and gives n outputs.

Thresh.	Adv.	Communication	References
$t < n/2$	passive	$\mathcal{O}\left(c_M n^2 + n^2\right)\kappa$	[BGW88]
$t < n/3$	active	$\mathcal{O}\left(c_M n^2 + n^4\right)\kappa + \mathcal{O}\left(n^3\right)\mathcal{BC}(\kappa)$	[HM01]
$t < n/2$	active	$\mathcal{O}\left(c_M n^5 + n^4\right)\kappa + \mathcal{O}\left(c_M n^5 + n^4\right)\mathcal{BC}(\kappa)$	[CDD+99]
$t < n/2$	active	$\mathcal{O}\left(c_M n^2 + n^5\kappa\right)\kappa + \mathcal{O}\left(n^3\right)\mathcal{BC}(\kappa)$	this paper

[1] Even cryptographically secure broadcast and MPC require a trusted key setup for $t \geq n/3$.

Technically, the new protocol improves the approach of [CDD$^+$99], which requires $\Omega(n^5)$ broadcasts per multiplication. We introduce a new concept, so-called *dispute control*, that allows to substantially reduce the communication complexity. The goal of dispute control is to reduce the frequency of faults that the adversary can provoke by identifying a pair of disputing players (at least one of them corrupted) whenever a fault is observed and preventing this pair from getting into dispute ever again. Hence, the number of faults that can occur during the whole protocol is limited to $t(t+1)$. This technique is inspired by the player-elimination framework [HMP00], and shares many advantages with it. However, player elimination is not to be suited for models with $t \geq n/3$. Furthermore, player elimination is not applicable in the input stage, which results in our protocol being more efficient than the protocol in [HM01] when the number of inputs is large ($n^2\kappa$ bits per input in our protocol versus $n^4\kappa$ bits in [HM01]).

2 Protocol Overview

2.1 Model

We consider a set $\mathcal{P}$ of n players, $\mathcal{P} = \{P_1, \ldots, P_n\}$, which are connected with a complete network of secure synchronous channels. Furthermore, we assume the availability of broadcast channels. These can be simulated when a trusted setup is available [PW92]. The adversary corrupts up to t players for any fixed t with $t < n/2$, and makes them deviate from the protocol in any desired manner. The adversary is computationally unbounded, active, adaptive and rushing. The security of our protocols is information-theoretic with a negligible error probability of $2^{-\mathcal{O}(\kappa)}$ for some security parameter κ.

For the ease of presentation, we always assume that the messages sent through the channels are from the right domain — if a player receives a message which is not in the right domain (e.g., no message at all), he replaces it with an arbitrary message from the specified domain.

The function to be computed is specified as an arithmetic circuit over a finite field $\mathcal{F} = GF(2^\kappa)$, with input, addition, multiplication, random, and output gates. We denote the number of gates of each type by c_I, c_A, c_M, c_R, and c_O.

2.2 Dispute Control

In the active model, the adversary can provoke inconsistencies among the honest players, who therefore regularly have to check their views and, in case of inconsistencies, invoke some fault-recovery procedure. These checks tend to be very expensive (they require invocations to a Byzantine agreement primitive), and must be performed even when no player deviates from the protocol.

The goal of *dispute control* is to reduce the frequency of faults by publicly identifying (localizing) a pair of disputing players (at least one of them corrupted) whenever a fault is observed and preventing this pair from getting into dispute ever again. Hence, the number of faults that can occur during the whole protocol is limited to $t(t+1)$.

The localized disputes are filed in a publicly known *dispute set* $\Delta \subseteq \mathcal{P} \times \mathcal{P}$, a set of unordered pairs of players that are in dispute with each other. A pair $\{P_i, P_j\} \in \Delta$ means that there is a dispute between P_i and P_j, hence either P_i or P_j (or both) are corrupted. Note that from the point of view of P_i, the players $\{P_j \mid \{P_i, P_j\} \in \Delta\}$ are corrupted, and P_i doesn't care for them; in particular, he won't send or receive any private messages from them. As no honest player can be in dispute with more than t players, we automatically include the pairs $\{P_i, P_j\}$ for every $P_j \in \mathcal{P}$ once P_i is involved in more than t disputes. Furthermore, we define the set $\mathcal{X}$ to be the set of players who are undoubtedly detected to be corrupted, i.e., those players who are in dispute with more than t other players.

Once dispute control is in place, we can take advantage of the fact that the number of faults during the protocol is limited and reduce the number of expensive consistency checks: We divide the protocol into n^2 *segments*, run each segment without any consistency checks and only at the end of the segment check all operations of the segment in a single verification step. If the verification fails, a new dispute is localized, and the segment is repeated. At most $t(t+1)$ segments can fail, and the total number of segment evaluations (including repetitions) is at most $n^2 + t(t+1)$, hence the overhead for repeating failed segments is only a factor of 2. Formally the evaluation of each segment proceeds as follows:

1. **Private (dispute-aware) computation.** The effective protocol is computed very efficiently but non-robustly. This computation is adjusted to prevent faults due to disputes that are already registered in the dispute set Δ. In particular, players in dispute do not communicate with each other privately.
2. **Fault detection.** The players jointly find out whether or not a fault has occurred. This step typically requires each player to broadcast one bit indicating whether or not he observed an inconsistency within the current segment. If no fault is reported, then the computation of the segment is completed, and the next segment is evaluated. If at least one fault is reported, we say that the segment has failed, and the following step is performed.
3. **Fault localization and dispute control.** The players publicly identify a pair $\{P_i, P_j\}$ of players, where at least one of them is corrupted and has deviated from the protocol, and who are not yet registered in Δ. Then we set $\Delta \leftarrow \Delta \cup \{P_i, P_j\}$ and restart the current segment.

2.3 Three-Level Secret-Sharing

We use three different levels of secret-shadings, all based on Shamir's sharing [Sha79], ameliorated with dispute control. The weakest level, called *1D-sharing*, is a polynomial sharing scheme, where the shares of players who are in dispute with the dealer (implicitly) receive a fixed-0 share, called *Kudzu-share*. In order to *1D-share* a value s, the dealer P_D selects a random degree-t polynomial $f(x)$ with $f(0) = s$ and $f(i) = 0$ for every $\{P_D, P_i\} \in \Delta$, and sends the shares $s_i = f(i)$ to every $P_i \in \mathcal{P}$ (the Kudzu-shares are not really sent; instead, the receiver sets his share to 0). A protocol $\mathsf{VSS1D}$ for verifiably 1D-share a bunch of values will be given in Section 3.2. Note that 1D-sharings are not robust; reconstruction requires that all players (except those with Kudzu-shares) cooperate.

However, they are detectable in the sense that it can be decided whether or not the reconstruction was successful.

The middle level of secret sharing, called *2D-sharing*, is a two-level polynomial sharings scheme: The share s_i of each player $P_i \in \mathcal{P}$ is 1D-shared among the players (for dealer P_i). More precisely, a value s is 2D-shared when there exists degree-t polynomials $f, f_1, \ldots, f_n$ with $f(0) = s$ and, for $i = 1, \ldots, n$, $f_i(0) = f(i)$ and $\forall P_j \in \mathcal{P} : \{P_i, P_j\} \in \Delta \rightarrow f_i(j) = 0$. Every player $P_i \in \mathcal{P}$ holds a share $s_i = f(i)$ of s, the polynomial $f_i(x)$ for sharing s_i, and a share-share $s_{ji} = f_j(i)$ of the share s_j of every player $P_j \in \mathcal{P}$. We say that P_i *owns* the 1D-sharing of s_i, which means in particular that players who are in dispute with P_i hold 0 as share-share of s_i. We will never have a dealer 2D-share a value; instead, we will upgrade 1D-sharings (or rather sums of 1D-sharings) to 2D-sharings, using protocol Upgrade1Dto2D. Note that also 2D-sharings are not robust.

The strongest level of secret sharing, called *2D*-sharing*, is a 2D-sharing, where in addition, the share-shares are secured with information checking (see Section 3.5). More precisely, for each share-share s_{ij} (which is not a Kudzu-share, i.e., $\{P_i, P_j\} \notin \Delta$), the owner P_i of the sharing has provided authentication tags for every verifier $P_V \in \mathcal{P}$ who is neither in dispute with the owner P_i nor the recipient P_j, i.e., $\{P_V, P_i\} \notin \Delta$ and $\{P_V, P_j\} \notin \Delta$. These authentication tags allow P_V in the reconstruction to verify the correctness of the received share-shares; hence, 2D*-sharings are robust. Actually, P_i does not distribute authentication tags for every single share-share s_{ij}, but rather for huge collections of many share-shares $s_{ij}^{(1)}, \ldots, s_{ij}^{(\ell)}$, and P_V can only verify the correctness of all share-shares at once. Also 2D*-sharings are never distributed by a dealer; instead, we will upgrade collections of 2D-sharings to 2D*-sharings, using protocol Upgrade2Dto2D*.

2.4 Main Protocol

The main protocol proceeds in three phases (each making use of segmentation and dispute control):

Preparation phase: The preparation phase uses the circuit-randomization technique of Beaver [Bea91a]: A number of so-called multiplication triples (a, b, c) with $c = ab$ are generated and shared among the players. These triples will then be used in the computation phase for efficiently multiplying shared values. Furthermore, a number of random values are generated and shared, which will be used as outputs of random gates.

Input phase: In the input phase, every player with input shares his input among the players.

Computation phase: In the computation phase, the circuit is evaluated gate by gate (level by level), with help of the prepared multiplication triples and the random values. Given the sharings of the multiplication triples, the random values, and the inputs, the computation phase is fully deterministic. Indeed, the computation phase can be seen as a sequence of reconstructions of known linear combinations of shared values.

Each phase uses dispute control. We initialize the dispute set $\Delta = \{\}$ and enter the first segment of the preparation phase. Then we evaluate segment by segment, and with each segment that fails and is to be repeated, the dispute set Δ grows. Once all segments of the preparation phase have succeeded, the players move on to the first segment of the input phase. Also in this phase, segments can fail and have to be repeated. This allows corrupted players to change their inputs. However, as the adversary obtains no information about whatsoever in the input phase, this does not affect the independence of the inputs. Once all input segments have succeeded, the players move on to the first segment of the computation phase. In this phase, the players (and hence also the adversary) do obtain information about their outputs; however, the computation stage is fully deterministic. Even when a segment fails (and is repeated) *after* the adversary has learned some output, he cannot influence the outputs of the honest players anymore.

In the preparation phase and in the input phase, the private computation is highly parallelized. All proposed sub-protocols process many inputs at once, producing many outputs. This helps reducing the costs for the fault detection and localization, as for all parallel instances, only one single fault-handling procedure is executed. Often, instead of verifying single instances of some test data, we will verify a random linear combination of many instances. Note that the protocols themselves do not use broadcast, but fault handling does.

3 Sub-Protocols

All sub-protocols have a private (dispute aware) computation, a fault detection and a fault localization. They can succeed or fail and the players always agree (using broadcast) on what is the case. In case of a failure the public output of the sub-protocol is a (new) pair of players $E = \{P_i, P_j\} \notin \Delta$ such as either P_i or P_j (or both) are corrupted. If some invoked sub-protocol fails with $E = \{P_i, P_j\}$ then the invoking sub-protocol fails with $E = \{P_i, P_j\}$ and is aborted (this abort will be handled in the main protocol).

3.1 Dispute-Control Broadcast

The protocol DC-Broadcast allows every sender $P_S \in \mathcal{P} \setminus \mathcal{X}$ to distribute a vector of ℓ values $s^{(1,S)}, \ldots, s^{(\ell,S)}$ among the players in $\mathcal{P} \setminus \mathcal{X}$, such that it is guaranteed that all honest recipients receive the same vectors (or the protocol fails).

This protocol is rather simple: Every sender directly transmits his vector to the players he is not in dispute with, and via another player to those players he is in dispute with. Then the players pairwisely compare their vectors by using universal hash functions [CW79]. As universal hash with key $k \in \mathcal{F}$, we use the function $U_k : \mathcal{F}^\ell \to \mathcal{F}, (s^{(1)}, \ldots, s^{(\ell)}) \mapsto s^{(1)} + s^{(2)}k + \ldots + s^{(\ell)}k^{\ell-1}$. The probability that two different vectors map to the same hash value for a uniformly chosen key is at most $\ell/|\mathcal{F}|$, which is negligible in our setting with $\mathcal{F} = GF(2^\kappa)$.

Protocol DC-Broadcast.

1. Private Computation: The following steps are executed in parallel for every sender $P_S \in \mathcal{P} \setminus \mathcal{X}$:
   1.1 P_S sends $s^{(1,S)}, \ldots, s^{(\ell,S)}$ to every P_i with $\{P_S, P_i\} \notin \Delta$.
   1.2 For every P_i with $\{P_S, P_i\} \in \Delta$ (but $P_i \notin \mathcal{X}$), the smallest player $P_{i'}$ with $\{P_S, P_{i'}\} \notin \Delta$ and $\{P_{i'}, P_i\} \notin \Delta$ forwards $s^{(1,S)}, \ldots, s^{(\ell,S)}$ to P_i.[2] We call $P_{i'}$ the *proxy* of P_i.
2. Fault Detection: The following steps are executed in parallel for every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$:
   2.1 P_V selects a key $k_V \in_R \mathcal{F}$ for a universal hash function U_k and sends it to every P_i with $\{P_V, P_i\} \notin \Delta$.
   2.2 Every P_i with $\{P_V, P_i\} \notin \Delta$ sends the values $h_{S,i} = U_{k_V}(s^{(1,S)}, \ldots, s^{(\ell,S)})$ for every P_S to P_V.
   2.3 P_V broadcasts a bit "accept" or "reject", indicating whether for every $P_S \in \mathcal{P} \setminus \mathcal{X}$, the hash values $h_{S,i}$ of each P_i with $\{P_V, P_i\} \notin \Delta$ are equal.

   If every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$ broadcasts "accept" in Step 2.3, then the protocol succeeds and terminates.
3. Fault Localization: The following steps are executed for the smallest $P_V \in \mathcal{P} \setminus \mathcal{X}$ reporting a fault.
   3.1 P_V selects S, i, j such that $P_S \notin \mathcal{X}$, $\{P_V, P_i\} \notin \Delta$, and $\{P_V, P_j\} \notin \Delta$, and $h_{S,i} \neq h_{S,j}$, and broadcasts $S, i, j, h_{S,i}, h_{S,j}$, and $k = k_V$.
   3.2 We denote the proxies of P_i and P_j by $P_{i'}$ and $P_{j'}$, respectively (if no proxy exists, we set $i' = i$, respectively $j' = j$). The players $P_S, P_i, P_j, P_{i'}, P_{j'}$ all compute and broadcast a hash value with key k of their vector $s^{(1,S)}, \ldots, s^{(\ell,S)}$, denoted as $h_S, h_i, h_j, h_{i'}, h_{j'}$, respectively. The protocol fails with E being the first pair (P_V, P_i), $(P_i, P_{i'})$, $(P_{i'}, P_S)$, $(P_S, P_{j'})$, $(P_{j'}, P_j)$, or (P_j, P_V), where $h_{S,i} \neq h_i$, $h_i \neq h_{i'}$, $h_{i'} \neq h_S$, $h_S \neq h_{j'}$, $h_{j'} \neq h_j$, or $h_j \neq h_{S,j}$, respectively.

Lemma 1. *If* DC-Broadcast *succeeds, then with overwhelming probability, for each sender $P_S \in \mathcal{P} \setminus \mathcal{X}$, all honest players in $\mathcal{P}$ hold the same vector $s^{(1,S)}, \ldots, s^{(\ell,S)}$, which is the vector of P_S if honest. If the protocol fails, a new dispute pair E is localized. The protocol communicates $\mathcal{O}(\ell n^2 + n^3)$ and broadcasts $\mathcal{O}(n)$ field elements.*

Proof. In order to prove that all honest players output the same vector $s^{(1,S)}, \ldots, s^{(\ell,S)}$ when the protocol succeeds, consider two honest players P_i and P_j. As both P_i and P_j are honest, $\{P_i, P_j\} \notin \Delta$ holds, and P_i and P_j have mutually exchanged universal hash values in Step 2. Hence, with overwhelming probability, a difference in the vectors would have been detected and the protocol would have failed. It follows immediately from the protocol that when P_S is

[2] The existence of such a player $P_{i'}$ for $P_{S'} \notin \mathcal{X}$ and $P_i \notin \mathcal{X}$ follows by a counting argument.

honest and the protocol succeeds, then all honest players receive the vector directly from P_S. When the protocol fails with dispute pair E, then one can verify by inspection that the two players in E disagree on a value they have privately exchanged, hence either of the players must be faulty. And as players in dispute do not communicate with each other, the localized dispute pair is new. $\square$

3.2 Verifiable Secret-Sharing

The protocol VSS1D allows every dealer $P_D \in \mathcal{P} \setminus \mathcal{X}$ to verifiably 1D-share ℓ values $s^{(1,D)}, \ldots, s^{(\ell,D)}$ resulting in each player $P_i \in \mathcal{P} \setminus \mathcal{X}$ holding the shares $s_i^{(1,D)}, \ldots, s_i^{(\ell,D)}$ for each dealer P_D. The correctness of these sharings is verified by letting every player take on the role of a verifier P_V and inspect a random linear combination of the sharings of each dealer P_D. For privacy reasons, each such random linear combination is blinded with a random 1D-sharing, i.e., every dealer P_D 1D-shares additional n blinding values $s^{(\ell+1,D)}, \ldots, s^{(\ell+n,D)}$.

Protocol VSS1D.

1. Private Computation: Every dealer $P_D \in \mathcal{P} \setminus \mathcal{X}$ selects n random blindings $s^{(\ell+1,D)}, \ldots, s^{(\ell+n,D)}$. Then, P_D 1D-shares $s^{(1,D)}, \ldots, s^{(\ell+n,D)}$, i.e., for every $m = 1, \ldots, \ell+n$, P_D picks a random polynomial $f^{(m,D)}(x)$ with $f^{(m,D)}(0) = s^{(m,D)}$ and $f^{(m,D)}(i) = 0$ for every i with $\{P_D, P_i\} \in \Delta$ (the Kudzu-shares), and sends the share $f^{(m,D)}(i)$ to every player P_i with $\{P_D, P_i\} \notin \Delta$; every player P_i with $\{P_D, P_i\} \in \Delta$ sets his share $s_i^{(m,D)} = 0$.
2. Fault Detection: Every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$ selects a random challenge vector $(r^{(1,V)}, \ldots, r^{(\ell,V)})$. Then, DC-Broadcast is invoked to let every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$ distribute his vector among the players $P_i \in \mathcal{P} \setminus \mathcal{X}$. Then the following steps are executed for every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$ (we suppress the index V and denote the challenge vector $(r^{(1)}, \ldots, r^{(\ell)})$):
   2.1 For every dealer P_D, the random linear combination $f^{(*,D)}(x)$ of his 1D-sharings is defined as $f^{(*,D)}(x) = \sum_{m=1}^{\ell} r^{(m)} f^{(m,D)}(x) + f^{(\ell+V,D)}(x)$. Accordingly, for every dealer P_D, every player P_i with $\{P_i, P_D\} \notin \Delta$ and $\{P_i, P_V\} \notin \Delta$ sends to P_V his share $s_i^{(*,D)}$ on $f^{(*,D)}(x)$, i.e., $s_i^{(*,D)} = \sum_{m=1}^{\ell} r^{(m)} s_i^{(m,D)} + s_i^{(\ell+V,D)}$.
   2.2 For each dealer $P_D \in \mathcal{P} \setminus \mathcal{X}$, the verifier P_V checks whether the received shares $s_i^{(*,D)}$ define a correct 1D-sharing for P_D, i.e., whether there exists a degree-t polynomial $\widetilde{f}^{(*,D)}(x)$ with $\widetilde{f}^{(*,D)}(i) = s_i^{(*,D)}$ for every i with $\{P_V, P_i\} \notin \Delta$ and $\{P_D, P_i\} \notin \Delta$, and $\widetilde{f}^{(*,D)}(i) = 0$ for every i with $\{P_D, P_i\} \in \Delta$ (Kudzu).[3] P_V broadcasts a bit "accept" or "reject", indicating whether or not the the above checks succeed for all dealers.

   If all verifiers $P_V \in \mathcal{P} \setminus \mathcal{X}$ broadcast "accept" the protocol succeeded and terminates.
3. Fault Localization: The following steps are executed for the smallest P_V reporting a fault in Step 2.2.

[3] Note that any linear combination of Kudzu-shares is Kudzu.

3.1 P_V broadcasts the index D of P_D whose polynomial $\widetilde{f}^{(*,D)}(x)$ does not define a correct 1D-sharing.

3.2 Every player P_i with $\{P_i, P_D\} \notin \Delta$ and $\{P_i, P_V\} \notin \Delta$ broadcasts his share $s_i^{(*,D)}$.

3.3 If the broadcasted shares define a 1D-sharing for dealer P_D, then P_V broadcasts the index i of a player P_i with $\{P_i, P_V\} \notin \Delta$ and $\{P_i, P_D\} \notin \Delta$ who has broadcasted a different share $s_i^{(*,D)}$ in Step 3.2 than he has privately sent to P_V in Step 2.1, and the protocol fails with $E = \{P_V, P_i\}$. Otherwise, when the broadcasted shares do not define a correct 1D-sharing for dealer P_D, then the dealer broadcasts the index i of a player P_i with $\{P_i, P_D\} \notin \Delta$ who has broadcasted a wrong share $s_i^{(*,D)}$ and the protocol fails with $E = \{P_D, P_i\}$.

Lemma 2. *If* VSS1D *succeeds, then with overwhelming probability, the values* $s^{(1,D)}, \ldots, s^{(\ell,D)}$ *of each dealer* $P_D \in \mathcal{P}\setminus\mathcal{X}$ *are correctly 1D-shared. If the protocol fails, then the localized pair* $E = \{P_i, P_j\}$ *is new (i.e.,* $E \notin \Delta$*) and either* P_i *or* P_j *(or both) are corrupted. The privacy of the inputs of the honest players is guaranteed through the whole protocol (even if the protocol fails). The protocol communicates* $\mathcal{O}(\ell n^2 + n^3)$ *and broadcasts* $\mathcal{O}(n)$ *field elements.*

Proof. In order to prove the correctness, first consider a dealer P_D, an honest verifier P_V, the (by P_D supposedly correct 1D-shared) values $s^{(1,D)}, \ldots, s^{(\ell,D)}$ and the blinding value $s^{(\ell+V,D)}$. Assume that the sharing of one of the values is not a correct 1D-sharing, i.e., the shares of the honest players (including the Kudzu shares) lie on a polynomial of degree higher than t. Then there are at most $2^{\kappa(\ell-1)}$ (out of $2^{\kappa\ell}$) challenge vectors $(r^{(1)}, \ldots, r^{(\ell)}) \in \mathcal{F}^\ell$ such that the sharing of $s^{(*,D)} = \sum_{m=1}^{\ell} r^{(m)} s^{(m,D)} + s^{(\ell+V,D)}$ is a correct 1D-sharing, i.e. the polynomial defined by the shares of the honest players is of degree t. As the verifier P_V chooses his challenge vector uniformly at random and gets the correctly linearly combined shares from all honest players (an honest verifier is in dispute with no honest player), the probability of him not detecting the fault is at most $2^{\kappa(\ell-1)}/2^{\kappa\ell} = 1/2^\kappa$. Thus the probability that the protocol succeeds in case of at least one faulty sharing (from any dealer) is negligible.

The privacy of the inputs of the honest players follows from the fact that up to t shares give no information about the secret and from the fact that the reconstructed linear combinations are blinded with a random value chosen by the dealer himself (for every verifier a different one) and are so (for every honest dealer) statistically independent from the dealers secret.

If the protocol fails, then the localized dispute pair consists of two players who have publicly disagreed on a value they have privately exchanged in some previous step (or a value computed from such values), therefore it is obvious, that at least one of them is corrupted. As only players who are not in dispute with each other communicate privately, the localized dispute is a new one. □

We present a protocol for reconstructing sums of correct 1D-sharings. Consider a set $\mathcal{P}_D \subseteq \mathcal{P} \setminus \mathcal{X}$ of dealers and a set $\mathcal{P}_R \subseteq \mathcal{P} \setminus \mathcal{X}$ of recipients and the actual dispute set Δ. Every dealer $P_D \in \mathcal{P}_D$ has verifiably 1D-shared (with the actual Δ) ℓ summands $s^{(1,D)}, \ldots, s^{(\ell,D)}$ with the polynomials $f^{(1,D)}(x), \ldots, f^{(\ell,D)}(x)$. We denote the share of $f^{(m,D)}(x)$ for player $P_i \in \mathcal{P}$ by $s_i^{(m,D)}$. Note that $s_i^{(m,D)} = 0$ when $\{P_D, P_i\} \in \Delta$ (Kudzu). The values $s^{(1)}, \ldots, s^{(\ell)}$ to be reconstructed are defined as the sums of the above summands, i.e., $s^{(m)} = \sum_{P_D \in \mathcal{P}_D} s^{(m,D)}$. Each of these values is implicitly shared (as Shamir-sharing, not as 1D-sharing) with the polynomial $f^{(m)}(x) = \sum_{P_D \in \mathcal{P}_D} f^{(m,D)}(x)$; we denote the (implicitly defined) share of each player $P_i \in \mathcal{P}$ by $s_i^{(m)} = f^{(m)}(i)$.

Protocol Reconstruct1D.

1. Private Computation: For every $m = 1, \ldots, \ell$, every player $P_i \in \mathcal{P}$ computes his sum share $s_i^{(m)} = \sum_{P_D \in \mathcal{P}_D} s_i^{(m,D)}$, and sends it to every $P_R \in \mathcal{P}_R$ with $\{P_i, P_R\} \notin \Delta$. Every $P_R \in \mathcal{P}_R$ checks for each $m = 1, \ldots, \ell$ whether the received shares lie on a polynomial $\widetilde{f}^{(m)}(x)$ of degree t. If so, it follows that $\widetilde{f}^{(m)}(x) = f^{(m)}(x)$, and P_R reconstructs $s^{(m)} = \widetilde{f}^{(m)}(0)$.
2. Fault Detection: Every $P_R \in \mathcal{P}_R$ broadcasts "accept" or "reject", indicating whether he could reconstruct all values $s^{(m)}$ for $m = 1, \ldots, \ell$ in Step 1. If all recipients broadcast "accept", then the protocol succeeds and terminates.
3. Fault Localization: The following steps are executed for the smallest complaining recipient $P_R \in \mathcal{P}_R$.
   - 3.1 P_R broadcasts the index m of the polynomial $\widetilde{f}^{(m)}(x)$ he could not reconstruct.
   - 3.2 Every player P_i with $\{P_i, P_R\} \notin \Delta$ sends to P_R his summand shares $s_i^{(m,D)}$ for every dealer $P_D \in \mathcal{P}_D$ with $\{P_i, P_D\} \notin \Delta$.
   - 3.3 P_R verifies for every P_i with $\{P_i, P_R\} \notin \Delta$ that the provided summand shares add up to the previously provided sum share, i.e., $\sum_{P_D : \{P_i, P_D\} \notin \Delta} s_i^{(m,D)} = s_i^{(m)}$.[4] In case of a fault, P_R broadcasts the index i of the bad player P_i, and the protocol fails with $E = \{P_i, P_R\}$.
   - 3.4 P_R broadcasts the index D of a dealer $P_D \in \mathcal{P}_D$ such that the received shares $s_i^{(m,D)}$ do not define a correct 1D-sharing for dealer P_D, i.e., there is no degree-t polynomial $f(x)$ with $f(i) = s_i^{(m,D)}$ for every i with $\{P_i, P_R\} \notin \Delta$ and $\{P_i, P_D\} \notin \Delta$, and $f(i) = 0$ (Kudzu) for every i with $\{P_i, P_R\} \notin \Delta$ and $\{P_i, P_D\} \in \Delta$.
   - 3.5 Every player P_i with $\{P_i, P_R\} \notin \Delta$ and $\{P_i, P_D\} \notin \Delta$ broadcasts his summand share $s_i^{(m,D)}$.
   - 3.6 If the broadcasted summand shares define a correct 1D-sharing for dealer P_D, then P_R broadcasts the index i of a player P_i who has broadcasted a different value $s_i^{(m,D)}$ in Step 3.5 than he has privately sent to P_R in Step 3.2, and the protocol fails with $E = \{P_i, P_R\}$. Otherwise, when the broadcasted summand shares do not define a cor-

[4] Note that the Kudzu-shares $s_i^{(*,D)}$ with $\{P_i, P_D\} \in \Delta$ are 0 and do not contribute to the sum.

rect 1D-sharing for P_D, then P_D broadcasts the index i of a player P_i who has broadcasted a wrong share $s_i^{(m,D)}$, and the protocol fails with $E = \{P_i, P_D\}$.

Lemma 3. *If the values $s^{(1,D)}, \ldots, s^{(\ell,D)}$ of each $P_D \in \mathcal{P}_D$ are correctly 1D-shared (for the actual Δ), then the following holds: If* Reconstruct1D *succeeds, then the privacy is guaranteed and every value reconstructed towards an honest recipient lies on the degree t polynomial defined by the (at least $t+1$) shares of the honest players. If the protocol fails then the localized pair $E = \{P_i, P_j\}$ is new and contains at least one corrupted player. The protocol communicates $\mathcal{O}(\ell n^2)$ and broadcasts $\mathcal{O}(n)$ field elements.*

Proof. As an honest verifier is not in dispute with any other honest player, he will receive at least $t+1$ shares of the honest players, which uniquely define a degree t polynomial. If the shares received from the corrupted players lie on this polynomial, he will reconstruct the right secret, otherwise the interpolated polynomial will be of degree higher then t and the protocol will fail. The rest follows (along the lines of proof of Lemma 2) from inspection of the protocol. □

3.3 Generating Random Challenges

The following protocol allows the players to generate a publicly known (i.e., to the players in $\mathcal{P} \setminus \mathcal{X}$) challenge vector $s^{(1)}, \ldots, s^{(\ell)}$, or the protocol fails, if one of the sub-protocols fails, and outputs a new dispute pair $E = \{P_i, P_j\}$:

Protocol GenerateChallenges**.**

1. Every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ selects a random summand vector $s^{(1,k)}, \ldots, s^{(\ell,k)}$.
2. Invoke VSS1D to let every P_k verifiably 1D-share his summand vector.
3. Invoke the protocol Reconstruct1D (with $\mathcal{P}_D = \mathcal{P}_R = \mathcal{P} \setminus \mathcal{X}$) to reconstruct the sum sharings $\sum_{P_k \in \mathcal{P}_D} s^{(1,k)}, \ldots, \sum_{P_k \in \mathcal{P}_D} s^{(\ell,k)}$ towards every $P_j \in \mathcal{P}_R$.

Lemma 4. *If* GenerateChallenges *succeeds, then with overwhelming probability, the generated values are uniformly distributed. If the protocol fails, then the localized dispute pair $E = \{P_i, P_j\}$ is new and contains at least one corrupted player. The protocol communicates $\mathcal{O}(\ell n^2 + n^3)$ and broadcasts $\mathcal{O}(n)$ field elements.*

3.4 Upgrading 1D-Sharings to 2D-Sharings

We present a protocol for upgrading sums of 1D-sharings to 2D-sharings. The given 1D-sharings must be for the actual Δ; the correctness of these sharings is implicitly verified in the upgrade protocol and must not be a priori guaranteed. The protocol outputs correct 2D-sharings or it fails with a new dispute pair E.

Formally, we consider a set $\mathcal{P}_D \subseteq \mathcal{P} \setminus \mathcal{X}$ of dealers, where each dealer $P_D \in \mathcal{P}_D$ has (for the actual Δ) 1D-shared ℓ summands $s^{(1,D)}, \ldots, s^{(\ell,D)}$ with the polynomials $f^{(1,D)}(x), \ldots, f^{(\ell,D)}(x)$. We denote the share of $f^{(m,D)}(x)$ for player

$P_i \in \mathcal{P}$ by $s_i^{(m,D)}$. Note that $s_i^{(m,D)} = 0$ when $\{P_D, P_i\} \in \Delta$ (Kudzu). The values $s^{(1)}, \ldots, s^{(\ell)}$ to be 2D-shared are defined as the sums of the above summands, i.e., $s^{(m)} = \sum_{P_D \in \mathcal{P}_D} s^{(m,D)}$. Each of these values is implicitly shared (as Shamir-sharing, not as 1D-sharing) with the polynomial $f^{(m)}(x) = \sum_{P_D \in \mathcal{P}_D} f^{(m,D)}(x)$; we denote the (implicitly defined) share of each player $P_i \in \mathcal{P}$ by $s_i^{(m)} = f^{(m)}(i)$.

Protocol Upgrade1Dto2D**.**

1. Private Computation: The players first jointly generate a sharing of an additional randomly chosen value $s^{(\ell+1)}$. Then, all $\ell+1$ sharings are upgraded to 2D-sharings, and the correctness is verified with destroying the privacy of this blinding value.
   1.1 Every dealer $P_D \in \mathcal{P}_D$ picks a random summand $s^{(\ell+1,D)}$ and 1D-shares it among the players with polynomial $f^{(\ell+1,D)}(x)$, resulting in every player P_i holding a share $s_i^{(\ell+1,D)}$.
   1.2 For every $m = 1, \ldots, \ell+1$, every player $P_i \in \mathcal{P} \setminus \mathcal{X}$ computes his sum share $s_i^{(m)} = \sum_{P_D \in \mathcal{P}_D} s_i^{(m,D)}$, and 1D-shares it with the polynomial $f_i^{(m)}(x)$, such that $f_i^{(m)}(j) = 0$ for $\{P_i, P_j\} \in \Delta$ (Kudzu). We denote the share-shares as $s_{ij}^{(m)}$. The 1D-sharing of detected players $P_i \in \mathcal{X}$ is the constant-0 sharing (all share-shares are Kudzu).
2. Fault Detection: In order to verify the correctness of the resulting sharings, the players jointly generate a random challenge vector $(r^{(1)}, \ldots, r^{(\ell)}) \in \mathcal{F}^\ell$ using the protocol GenerateChallenges. Then, the correctness of the 2D-sharing of the random linear combination $\sum_{m=1}^{\ell} r^{(m)} s^{(m)} + s^{(\ell+1)}$ will be verified (in parallel) by every player $P_V \in \mathcal{P} \setminus \mathcal{X}$. We denote the linearly combined polynomials by $f(x) = \sum_{m=1}^{\ell} r^{(m)} f^{(m)}(x) + f^{(\ell+1)}(x)$, respectively $f_i(x) = \sum_{m=1}^{\ell} r^{(m)} f_i^{(m)}(x) + f_i^{(\ell+1)}(x)$.
   The following steps are performed in parallel for every verifier $P_V \in \mathcal{P} \setminus \mathcal{X}$:
   2.1 Every P_j with $\{P_V, P_j\} \notin \Delta$ computes and sends to P_V the following linear combinations of his share-shares for every $i = 1, \ldots, n$ with $\{P_i, P_j\} \notin \Delta$: $\quad s_{ij} = \sum_{m=1}^{\ell} r^{(m)} s_{ij}^{(m)} + s_{ij}^{(\ell+1)}$.
   2.2 P_V checks for each $i = 1, \ldots, n$, whether the received share-shares s_{ij} define a valid 1D-sharing for dealer P_i, i.e., there exists a polynomial $\widetilde{f}_i(x)$ with $\widetilde{f}_i(j) = s_{ij}$ for every j with $\{P_V, P_j\} \notin \Delta$ and $\{P_i, P_j\} \notin \Delta$, and $\widetilde{f}_i(j) = 0$ (i.e., Kudzu) for every j with $\{P_i, P_j\} \in \Delta$,[5] and broadcasts a bit "accept" or "reject".
   2.3. P_V checks that the first-level sharing $\widetilde{f}_1(0), \ldots, \widetilde{f}_n(0)$ is a valid Shamir-sharing of degree t and broadcasts "accept" or "reject".

   If all verifiers P_V broadcast "accept" in Steps 2.2 and 2.3, the protocol succeeded and terminates.
3. Fault Localization: The following steps are executed for the smallest complaining verifier P_V.

[5] Observe that in this case $\widetilde{f}_i(x) = f_i(x)$.

3.1 If the reported fault was in Step 2.2, i.e., P_V observed that one of the second-level sharings is not a correct 1D-sharing, the following steps are executed:

3.1.1 P_V broadcasts the index i of the invalid second-level sharing.

3.1.2 Every P_j with $\{P_j, P_V\} \notin \Delta$ and $\{P_j, P_i\} \notin \Delta$ broadcasts s_{ij}.

3.1.3 If the broadcasted shares define a correct 1D-sharing, then there exists a player P_j with $\{P_j, P_V\} \notin \Delta$ who has broadcasted a different value than he has privately sent to P_V in Step 2.1. P_V broadcasts his index j, and the protocol fails with $E = \{P_V, P_j\}$. If the broadcasted shares do not define a correct 1D-sharing, the owner P_i of this second-level sharing broadcasts the index j of a player P_j (with $\{P_i, P_j\} \notin \Delta$) who has broadcasted a wrong share $s_{ij} \neq f_i(j)$, and the protocol fails with $E = \{P_i, P_j\}$.

3.2 If the observed fault was in Step 2.3, i.e., P_V could correctly interpolate each second-level sharing $\widetilde{f}_1(x), \ldots, \widetilde{f}_n(x)$, but the interpolated values $\widetilde{f}_1(0), \ldots, \widetilde{f}_n(0)$ do not define a valid (first-level) Shamir-sharing of degree t,[6] then the following steps are executed.

3.2.1 For every dealer P_D, the random linear combination $f^{(*,D)}(x)$ of his 1D-sharings is defined as $f^{(*,D)}(x) = \sum_{m=1}^{\ell} r^{(m)} f^{(m,D)}(x) + f^{(\ell+1,D)}(x)$. Accordingly, for every dealer P_D, every player P_i with $\{P_i, P_D\} \notin \Delta$ and $\{P_i, P_V\} \notin \Delta$ sends to P_V his share $s_i^{(*,D)}$ on $f^{(*,D)}(x)$, i.e., $s_i^{(*,D)} = \sum_{m=1}^{\ell} r^{(m)} s_i^{(m,D)} + s_i^{(\ell+1,D)}$.

3.2.2 P_V checks for every player P_i with $\{P_V, P_i\} \notin \Delta$ that $\sum_{P_D : \{P_i, P_D\} \notin \Delta} s_i^{(*,D)} = \widetilde{f}_i(0)$.[7] If the check fails for some P_i, then P_V broadcasts i, and the protocol fails with $E = \{P_V, P_i\}$.

3.2.3 P_V broadcasts the index D of P_D such that the received shares $s_i^{(*,D)}$ (for every i with $\{P_i, P_D\} \notin \Delta$ and $\{P_i, P_V\} \notin \Delta$) do not define a correct 1D-sharing.

3.2.4 Every $P_i \in \mathcal{P}$ with $\{P_i, P_V\} \notin \Delta$ and $\{P_i, P_D\} \notin \Delta$ broadcasts his share $s_i^{(*,D)}$.

3.2.5 If the broadcasted shares define a correct 1D-sharing for dealer P_D, then P_V broadcasts the index i of the player P_i with $\{P_V, P_i\} \notin \Delta$ who has broadcast a different share $s_i^{(*,D)}$ than he has privately sent to P_V in Step 3.2.1, and the protocol fails with $E = \{P_V, P_i\}$. If the broadcasted shares do not define a correct 1D-sharing for dealer P_D, then P_D broadcasts the index i of a player P_i with $\{P_D, P_i\} \notin \Delta$ who broadcasted a wrong share $s_i^{(*,D)}$, and the protocol fails with $E = \{P_D, P_i\}$.

[6] Note that $\widetilde{f}_i(0) = f_i(0)$ for every i, i.e., $\widetilde{f}_i(0)$ is the linear combination of the values that P_i did indeed 1D-share as his shares $s_i^{(m)}$ in Step 1.

[7] Note that the Kudzu-shares $s_i^{(*,D)}$ with $\{P_i, P_D\} \in \Delta$ are 0 and do not contribute to the sum.

Lemma 5. *If* Upgrade1Dto2D *succeeds, then with overwhelming probability, the upgraded sharings are correct 2D-sharings. If the protocol fails, then the localized pair $E = \{P_i, P_j\}$ is new and contains at least one corrupted player. The privacy of the shared values is guaranteed through the whole protocol (even if it fails). The protocol communicates $\mathcal{O}(\ell n^2 + n^3)$ and broadcasts $\mathcal{O}(n)$ field elements.*

Proof. Along the lines of the proof of Lemma 2. □

3.5 Information Checking with Dispute Control

An *information-checking (IC)* scheme allows a sender to deliver a message to a recipient in such a way that the recipient can later forward the message and prove its authenticity to a designated verifier. More precisely, an IC-scheme for a sender P_S, recipient P_R, and verifier P_V, consists of two protocols:[8]

IC-Distr: The sender P_S delivers the message m and some authentication tag y to P_R and some checking tag z to P_V.

IC-Reveal: The recipient P_R forwards m and y to P_V, who uses z to verify the authenticity of m, and either accepts or rejects m.

Our information-checking protocol is a variant of the information-checking protocol of [CDD+99] with two modifications. First, our IC-Distr protocol may fail in case of a fault; then, a dispute among two of the three players is identified.[9] Second, our protocol supports authenticating long messages $m = (m_1, \ldots, m_\ell) \in \mathcal{F}^\ell$ without additional costs.[10]

For authenticating $m = (m_1, \ldots, m_\ell)$, a random degree-$\ell$ polynomial $f(x)$ with $f(i) = m_i$ for $i = 1, \ldots, \ell$ is chosen, then the authentication tag is $y = f(0)$ and the verification tag is a random point $z = (u, v)$ with $f(u) = v$ and $u \geq \ell$. One can easily verify that this approach satisfies completeness, secrecy, and correctness (with error probability $\ell/(|\mathcal{F}| - \ell - 1)$) as long as the tags are computed as indicated. In order to ensure that the sender computes the tags correctly, we use a cut-and-choose proof: The sender generates and distributes κ independent tags, and the verifier hands half of them to the recipient, who checks them. The concrete protocols are given in the sequel:

Protocol IC-Distr.

1. Private Computation: The sender P_S, holding message $m = (m_1, \ldots, m_\ell)$, selects uniformly at random κ authentication tags $y_1, \ldots, y_\kappa \in_R \mathcal{F}^\kappa$, κ elements $u_1, \ldots, u_\kappa \in_R (\mathcal{F} \setminus \{0, \ldots, \ell\})^\kappa$, and computes $v_1, \ldots, v_\kappa$ such that for each $i \in \{1, \ldots, \kappa\}$, the $\ell + 2$ points $(0, y_i), (1, m_1), \ldots, (\ell, m_\ell), (u_i, v_i)$ lie on a polynomial of degree ℓ. P_S sends the message m and the authentication tags $y_1, \ldots, y_\kappa$ to P_R and the verification tags $z_1 = (u_1, v_1), \ldots, z_\kappa = (u_\kappa, v_\kappa)$ to P_V.

[8] In [RB89,CDD+99], a different notation is used. They denote the sender as "dealer", the recipient as "intermediary", and the verifier as "receiver".

[9] In our context, the IC-scheme will be used only by triples of players with no a priori dispute among them, so the identified dispute will be a new one.

[10] The costs in the scheme of [CDD+99] grow linearly with the size of the message.

2. Fault Detection:
   2.1 P_V partitions the index set $\{1,\ldots,\kappa\}$ into two partitions I and $\overline{I}$ of (almost) equal size, and sends I, $\overline{I}$, and z_i for every $i \in I$ to P_R.
   2.2 P_R checks whether for *every* $i \in I$, the points $(0, y_i), (1, m_1), \ldots, (\ell, m_\ell), z_i$ lie on a polynomial of degree ℓ, and broadcasts either "accept" (and the protocol succeeded) or "reject".
3. Fault Localization: If P_R broadcasted "reject", the protocol fails and:
   3.1 P_R selects $i \in I$ such that the verification tag z_i received from P_V does not match with the message m and the authentication tag y_i received from P_S, and broadcasts i and z_i.
   3.2 P_S and P_V broadcast z_i.
   3.3 If the z_i-s broadcasted by P_S and P_V differ, then $E = \{P_S, P_V\}$. Otherwise, if the z_i-s broadcasted by P_R and P_V differ, then $E = \{P_R, P_V\}$. Otherwise, $E = \{P_S, P_R\}$.

Protocol IC-Reveal.
1. The recipient P_R sends the message m and the authentication tags y_i for $i \in \overline{I}$ to the verifier P_V.
2. The verifier with verification tags $z_1, \ldots, z_\ell$ accepts $m = (m_1, \ldots, m_\ell)$ if for *any* $i \in \overline{I}$, the points $(0, y_i), (1, m_1), \ldots, (\ell, m_\ell), z_i$ form a polynomial of degree ℓ; otherwise, he rejects m.

Lemma 6. *If* IC-Distr *succeeds and P_V, P_R are honest, then with overwhelming probability P_V accepts the message m in* IC-Reveal *(completeness). If* IC-Distr *fails, then the localized pair E contains at least one corrupted player. If P_S and P_V are honest, then with overwhelming probability, P_V rejects any fake message $m' \neq m$ in* IC-Reveal *(correctness). If P_S and P_R are honest, then P_V obtains no information about m in* IC-Distr *(even if it fails) (privacy).*

Proof. Completeness: If the cut-and-choose proof is successful, then the probability that at least one of the remaining authentication tags is valid is at least $1 - \kappa/2^\kappa$. Correctness: The probability that an corrupted receiver can produce at least one correct tag for a message $m' \neq m$ is equal to the probability, that he can guess at least one verification point z_i, which is less than $\kappa/(2^\kappa - \ell - 1)$. Privacy follows from the fact that the verification tag is statistically independent from the message. □

3.6 Upgrading 2D-Sharings to 2D*-Sharings

The following protocol upgrades ℓ 2D-sharings to 2D*-sharings. We denote the 2D-shared values by $s^{(m)}$ (for $m = 1, \ldots, \ell$), the shares of each player $P_i \in \mathcal{P}$ by $s_i^{(m)}$, and P_j's share-share of $s_i^{(m)}$ by $s_{ij}^{(m)}$.

Protocol Upgrade2Dto2D*.

1. For every triple of players $P_i, P_j, P_k \in \mathcal{P}$ with no dispute among them (i.e., $\{P_i, P_j\} \notin \Delta$, $\{P_i, P_k\} \notin \Delta$, $\{P_j, P_k\} \notin \Delta$), the protocol IC-Distr is invoked for the message $m = (s_{ij}^{(1)}, \ldots, s_{ij}^{(\ell)})$ with sender P_i, receiver P_j and verifier P_k. The message is not really sent, as P_j already holds it. Furthermore, these up to n^3 parallel invocations are merged when it comes to fault-detection and fault-localization: Every player P_j broadcasts one single bit in the fault-detection, indicating whether he observed a fault in one of the instances he acted as recipient. Then, the smallest player P_j that reported a fault, broadcasts i and k, indicating the instance i, j, k in which he observed the fault, and fault-localization is invoked only for this instance.

Lemma 7. *If the 2D-sharings to be upgraded are correct (for the actual Δ) and the protocol* Upgrade2Dto2D* *succeeds, then the upgraded 2D*-sharings are with overwhelming probability correct. If the protocol fails, then the output pair E is new and contains at least one corrupted player. The privacy of the shared values is guaranteed through the whole protocol (even if it fails). The protocol communicates $\mathcal{O}(n^3\kappa)$ and broadcasts $\mathcal{O}(n)$ field elements.*

3.7 ABC-Protocol

The following protocol allows every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ to prove that for every $m = 1, \ldots, \ell$, the (for the actual Δ correctly) 1D-shared value $c^{(m,k)}$ is the product of the (for the actual Δ correctly) 1D-shared values $a_k^{(m)}$ and $b_k^{(m)}$. This ABC-protocol is inspired by the corresponding protocol of [CDD+99].

The intuition of the ABC protocol is the following (where we denote the factors as a and b and the product as c): The prover shares a random $\overline{a}$ and $\overline{c} = \overline{a}b$, i.e., $(\overline{a}, b, \overline{c})$ is a multiplication triple, and proves for a random challenge r, that the shared triple $(ra+\overline{a}, b, rc+\overline{c})$ is a correct multiplication triple. This is achieved by first reconstructing $\widetilde{a} = ra+\overline{a}$, and then verifying that $z = \widetilde{a}b-rc-\overline{c}$ is a sharing of 0. For the sake of efficiency, we parallelize this ABC-proof for many triples and amortize the verification. Instead of reconstructing the sharing of each $\widetilde{a}$, we ask the prover to send the (alleged) values $\widetilde{a}$ to every player; who then verify that a random linear combination of these sharings reconstructs to the linear combination of the alleged values. Analogously, instead of verifying each z to be zero, the players reconstruct a random linear combination of these values, which must be zero.

Protocol ABC.

1. Every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ selects for each $m = 1, \ldots, \ell$ a random $\overline{a}_k^{(m)}$ and computes $\overline{c}^{(m,k)} = \overline{a}_k^{(m)} b_k^{(m)}$.
2. Invoke VSS1D to let every $P_k \in \mathcal{P} \setminus \mathcal{X}$ verifiably 1D-share $\overline{a}_k^{(m)}$ and $\overline{c}^{(m,k)}$ for $m = 1, \ldots, \ell$.
3. Invoke GenerateChallenges to generate one random challenge r.

4. Every $P_k \in \mathcal{P} \setminus \mathcal{X}$ sends $\widetilde{a}_k^{(m)} = ra_k^{(m)} + \overline{a}_k^{(m)}$ for $m = 1, \ldots, \ell$ to every $P_i \in \mathcal{P}$ with $\{P_k, P_i\} \notin \Delta$.
5. Invoke GenerateChallenges to generate ℓ challenges $r^{(1)}, \ldots, r^{(\ell)}$.
6. Invoke Reconstruct1D with $\mathcal{P}_R = \mathcal{P} \setminus \mathcal{X}$ to publicly reconstruct $\widehat{a}_k = \sum_{m=1}^{\ell} r^{(m)} \left(ra_k^{(m)} + \overline{a}_k^{(m)} \right)$ for $k = 1, \ldots, n$.[11]
7. Every $P_i \in \mathcal{P} \setminus \mathcal{X}$ checks for every P_k with $\{P_i, P_k\} \notin \Delta$ whether $\widehat{a}_k = \sum_{m=1}^{\ell} r^{(m)} \widetilde{a}_k^{(m)}$, and broadcasts the index k of a player P_k for whom the check failed, respectively $\perp$ if all checks succeed. If at least one player P_i broadcasts k with $\{P_i, P_k\} \notin \Delta$, then the protocol fails with $E = \{P_i, P_k\}$ for the smallest such P_i (and the accused P_k).
8. Invoke Reconstruct1D with $\mathcal{P}_R = \mathcal{P} \setminus \mathcal{X}$ to reconstruct $z^{(k)} = \sum_{m=1}^{\ell} r^{(m)} \left(\widetilde{a}_k^{(m)} b_k^{(m)} - rc^{(m,k)} - \overline{c}^{(m,k)} \right)$ for $k = 1, \ldots, n$. Note that $\widetilde{a}_k^{(m)}$ is a constant known to all players P_i with $\{P_i, P_k\} \notin \Delta$,[12] hence $z^{(k)}$ is a linear combination of 1D-shared values, as required by Reconstruct1D. Note that when this reconstruction succeeds, then every player $P_V \in \mathcal{P} \setminus \mathcal{X}$ reconstructs the same vector $(z^{(1)}, \ldots, z^{(n)})$.
9. Every player $P_V \in \mathcal{P} \setminus \mathcal{X}$ checks whether the reconstructed values $z^{(k)} = 0$ for every $P_k \in \mathcal{P} \setminus \mathcal{X}$. If this check fails, then P_k is corrupted, and the protocol fails with $E = \{P_i, P_k\}$ for all $P_i \in \mathcal{P}$ (i.e., P_k is in dispute with every player).

Lemma 8. *If all triples $(a_k^{(m)}, b_k^{(m)}, c^{(m,k)})$ are correctly 1D-shared for the actual Δ, then the following holds with overwhelming probability: If* ABC *succeeds, then the checked triples $(a_k^{(m)}, b_k^{(m)}, c^{(m,k)})$ are correct multiplication triples, i.e. $c^{(m,k)} = a_k^{(m)} b_k^{(m)}$ for every $m = 1, \ldots \ell$, and their privacy is preserved. If the protocol fails, then it localizes a new dispute pair E containing at least one corrupted player (respectively localizes single player who is corrupted). The protocol communicates $\mathcal{O}(\ell n^2 + n^3)$ and broadcasts $\mathcal{O}(n)$ field elements.*

Proof. In order to prove correctness, assume that there is at least one (incorrect) triple $(a_k^{(m)}, b_k^{(m)}, c^{(m,k)})$ (of player P_k) such that $c^{(m,k)} \neq a_k^{(m)} b_k^{(m)}$. Then there is at most one (out of 2^κ) challenge $r \in F$ such that $(ra_k^{(m)} + \overline{a}_k^{(m)}) b_k^{(m)} - rc^{(m,k)} - \overline{c}^{(m,k)} = 0$. If $(ra_k^{(m)} + \overline{a}_k^{(m)}) b_k^{(m)} - rc^{(m,k)} - \overline{c}^{(m,k)} \neq 0$ then there are at most $2^{\kappa(\ell-1)}$ (out of $2^{\kappa\ell}$) challenge vectors $(r^{(1)}, \ldots, r^{(\ell)}) \in \mathcal{F}^\ell$ such that the sum $z^{(k)} = \sum_{m=1}^{\ell} r^{(m)} \left(\left(ra_k^{(m)} + \overline{a}_k^{(m)} \right) b_k^{(m)} - rc^{(m,k)} - \overline{c}^{(m,k)} \right) = 0$. So provided that the values $a_k^{(m)}, b_k^{(m)}, c^{(m,k)}, \overline{a}_k^{(m)}, \overline{c}^{(m,k)}$ for $m = 1, \ldots, \ell$ are correctly 1D-shared, the challenges are random, and in Step 4., player P_k sent the correct

[11] Note that the 1D-sharing $\widehat{a}_k$ belongs to dealer P_k. Formally, Reconstruct1D requires every value to be reconstructed to be the *sum* of one 1D-sharing of each dealer in $\mathcal{P}_D$; hence, we implicitly assume constant-0 1D-sharings for the other dealers, and set $\mathcal{P}_D = \mathcal{P} \setminus \mathcal{X}$.

[12] Note that P_k is the owner of the 1D-sharing of $z^{(k)}$; hence, the share of every player P_i with $\{P_i, P_k\} \in \Delta$ is Kudzu, and he does not need to know the constant $\widetilde{a}_k^{(m)}$.

$\widetilde{a}_k^{(m)} = ra_k^{(m)} + \overline{a}_k^{(m)}$ for $m = 1, \ldots, \ell$ to every $P_i \in \mathcal{P}$ with $\{P_k, P_i\} \notin \varDelta$, the probability of the false triple not being detected is at most $2/2^\kappa$, which is negligible. As with overwhelming probability the values $a_k^{(m)}, b_k^{(m)}, c^{(m,k)}, \overline{a}_k^{(m)}, \overline{c}^{(m,k)}$ for $m = 1, \ldots, \ell$ are correctly 1D-shared and the challenges are random, it is now sufficient to show that the probability of P_k sending at least one false $\widetilde{a}_k^{(m)} \neq ra_k^{(m)} + \overline{a}_k^{(m)}$ to at least one honest verifier P_i in Step 4 and not being detected (by P_i) in Step 7 is negligible. This holds because for a false $\widetilde{a}_k^{(m)}$ there are at most $2^{\kappa(\ell-1)}$ (out of $2^{\kappa\ell}$) challenge vectors for which the check in Step 7 does not fail. □

4 Preparation Phase

The goal of this phase is to generate c_M random 2D*-shared multiplication triples (a, b, c) (one for each multiplication gate) and c_R random 2D*-shared values (one for each random gate). We wastefully generate $c_M + c_R$ random multiplication triples and use only the first factor for the random gates.

The generation of the $c_M + c_R$ multiplication triples is divided into n^2 segments, each of length $L = \lceil (c_M + c_R)/n^2 \rceil$. The computation is non-robust, and its correctness is verified at the end of the segment. In fact, the segment will consist of several stages, each with a private computation and fault-detection. As soon as a fault is reported in a fault-detection procedure, the corresponding fault-localization is used to localize a new dispute to be registered in $\varDelta$, and the whole segment has failed and is repeated.

Protocol PreparationPhase**.**
Set $\varDelta := \{\}$ and $\mathcal{X} = \{\}$, and for each segment (of length L) do the following steps. If any of the invoked sub-protocols fails, then include the localized pair $E = \{P_i, P_j\}$ in $\varDelta$, i.e., $\varDelta \leftarrow \varDelta \cup \{P_i, P_j\}$, and repeat the failed segment.

1. Generate $2L$ correct random 2D-sharings $\left(a^{(1)}, b^{(1)}\right), \ldots, \left(a^{(L)}, b^{(L)}\right)$:
   1.1. Every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ 1D-shares L randomly selected pairs $\left(a^{(1,k)}, b^{(1,k)}\right), \ldots, \left(a^{(L,k)}, b^{(L,k)}\right) \in \mathcal{F}^2$ among the players. We denote the distributed shares of $a^{(m,k)}$ by $a_1^{(m,k)}, \ldots, a_n^{(m,k)}$.
   1.2. Invoke Upgrade1Dto2D with $\mathcal{P}_D = \mathcal{P} \setminus \mathcal{X}$ and $\ell = L$ to upgrade the implicitly defined sum sharings of $\sum_{P_k \in \mathcal{P}_D} a^{(1,k)}, \ldots, \sum_{P_k \in \mathcal{P}_D} a^{(L,k)}$ to 2D-sharings, resulting in L correctly 2D-shared random values $a^{(1)}, \ldots, a^{(\ell)}$. The same for b.
2. Multiply the L pairs $\left(a^{(1)}, b^{(1)}\right), \ldots, \left(a^{(L)}, b^{(L)}\right)$, resulting in L correctly 2D-shared products $c^{(1)}, \ldots, c^{(L)}$:
   2.1. Every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ computes for every $m = 1, \ldots, L$ the product $c^{(m,k)}$ of his shares $a_k^{(m)}$ and $b_k^{(m)}$. Note that the product $c^{(m)} = a^{(m)} b^{(m)}$ can be computed as a weighted sum of these values $c^{(m,k)}$ (namely Lagrange interpolation); accordingly, we will compute a sharing of $c^{(m)}$ as weighted sum of sharings of $c^{(m,1)}, \ldots, c^{(m,n)}$.

2.2. Invoke VSS1D to let every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ verifiably 1D-share his values $c^{(1,k)}, \ldots, c^{(L,k)}$.

2.3. Invoke the protocol ABC to have every player $P_k \in \mathcal{P} \setminus \mathcal{X}$ prove that for every $m = 1, \ldots, L$, the value $c^{(m,k)}$ he shared in Step 2 is indeed the product of his shares $a_k^{(m)}$ and $b_k^{(m)}$, which are implicitly 1D-shared as part of the 2D-sharings of $a^{(m)}$ and $b^{(m)}$, respectively.

2.4. Invoke the protocol Upgrade1Dto2D with $\mathcal{P}_D = \mathcal{P} \setminus \mathcal{X}$ to upgrade the sharings of the weighted sums $\sum_{P_k \in \mathcal{P}_D} \lambda_k c^{(1,k)}, \ldots, \sum_{k=1}^{n} \lambda_k c^{(L,k)}$ to 2D-sharings, where λ_k denotes the Lagrange coefficients.[13]

3. Invoke Upgrade2Dto2D* to upgrade all $3L$ 2D-sharings to 2D*-sharings.

Lemma 9. *With overwhelming probability, the protocol* PreparationPhase *generates $c_M + c_R$ correctly 2D*-shared random multiplication triples (a, b, c) with $c = ab$; the secrecy of the triples is preserved. The protocol communicates $\mathcal{O}((c_M + c_R)n^2 + n^5\kappa)$ and broadcasts $\mathcal{O}(n^3)$ field elements.*

Proof. In order to show the correctness first consider one execution of the Steps 1.–3. for one segment of length L. (Note that the dispute set Δ remains unchanged through Steps 1.–3.) If the execution succeeds, then with overwhelming probability, the triples $\left(a^{(1)}, b^{(1)}, c^{(1)}\right), \ldots, \left(a^{(L)}, b^{(L)}, c^{(L)}\right)$ are correctly 2D*-shared (because of Lemma 2, 5, and 7), and $c = ab$ holds because of Lemma 8 for each triple (a, b, c). As there are n^2 segments and the adversary can provoke less than n^2 executions to fail (in total), he has less then $2n^2$ attempts to introduce a segment with a false triple. Because n is at most polynomial in κ, the probability that a false triple is not detected is negligible.

Privacy follows from the privacy of the invoked sub-protocols. Some of them do not guarantee privacy in case of a failure, but in such case all generated values are discarded and completely new shared values will be generated. □

5 Input Phase

The goal of the input phase is to provide 2D*-sharings of c_I inputs.

We set the upper bound on the number of input gates of a segment to $L = \lceil \frac{c_I}{n^2} \rceil$ and limit each segment to contain only input gates of the same player.

Protocol InputPhase**.**
For each segment, the following steps are executed to let the dealer $P_D \in \mathcal{P} \setminus \mathcal{X}$ verifiably 2D*-share his L inputs $s^{(1)}, \ldots, s^{(L)}$.[14] If any of the invoked sub-protocols fails, include the localized pair $E = \{P_i, P_j\}$ in Δ, i.e., $\Delta \leftarrow \Delta \cup \{P_i, P_j\}$, and repeat the segment.

[13] Note that the sharings of detected players $P_D \in \mathcal{X}$ are not considered in the Lagrange interpolation; however, as their shares are 0 (Kudzu), this omission does not falsify the outcome.

[14] If the dealer P_D is detected, i.e., $P_D \in \mathcal{X}$, then the players take the all-zero sharing of 0, i.e., every share is 0 and every share-share is 0 (Kudzu). Note that no authentication tags are needed because all share-shares are Kudzu.

1. P_D (unverifiably) 1D-shares the input values $s^{(1)}, \ldots, s^{(L)}$.
2. Invoke Upgrade1Dto2D with $\mathcal{P} = \{P_D\}$ to upgrade the 1D-sharings of $s^{(1)}, \ldots, s^{(L)}$ to 2D-sharings.
3. Invoke Upgrade2Dto2D* to upgrade the 2D-sharings of $s^{(1)}, \ldots, s^{(L)}$ to 2D*-sharings.

Lemma 10. *With overwhelming probability, the protocol* InputPhase *computes correct 2D*-sharings of c_I inputs, where the privacy of the inputs of the honest players is preserved. The protocol communicates $\mathcal{O}(c_I n^2 + n^5\kappa)$ and broadcasts $\mathcal{O}(n^3)$ field elements.*

Proof. In one execution of Steps 1.–3., the probability of success in spite of a false sharing is negligible. As there are at most n^2+n segments and less than n^2 repetitions, the adversary has at most $2n^2+n$ independent attempts to introduce a segment with a false sharing, hence his success probability is negligible. The privacy is guaranteed even in case of failure (and repetition) of some segment. □

6 Computation Phase

The computation of the circuit proceeds gate-by-gate. First, to every random and every multiplication gate, a prepared 2D*-shared random triple is assigned.

Given the 2D*-sharings of the multiplication triples and of the inputs, all values to be computed (and to be opened) in the computation stage are completely determined. We therefore call the values shared in the preparation phase and in the input phase the *base values* of the computation. All base values are robustly shared with 2D*-sharings.

It turns out that the value of each gate can be computed as linear combination of such base values. This is trivial as long as the circuit only consists of addition and random gates. For a multiplication gate, the players publicly reconstruct two sharings (both linear combinations of base values), such that the value of the multiplication gate is a linear combination of base values, where the coefficients of the linear combination depend on the two reconstructed values [Bea91a]. Hence, the whole computation phase consists only of a sequence of reconstructions of publicly known linear combinations of base sharings. More precisely, the gates are evaluated as follows:

Input Gate: Assign the corresponding 2D*-sharing of the input to the gate.

Random Gate: Assign the 2D*-sharing of a of the assigned multiplication triple (a, b, c) to the gate.

Addition Gate: To both summands, a linear combination of base sharings was assigned. Assign to the gate the sum of these two linear combinations (which is again a linear combination of base sharings).

Multiplication Gate: To both factors, a linear combination of base sharings was assigned. We denote the corresponding values by x and y, and denote the assigned multiplication triple by (a, b, c). The players reconstruct $d_x = x - a$ and $d_y = y - b$ towards every player in $\mathcal{P}$ (both d_x and d_y are represented as known linear combination of base sharings), and assign to the gate the linear combination $d_x d_y + d_x b + d_y a + c$ (i.e., a linear combination of the 2D*-sharings of a, b, and c, all three of them base sharings).

Output Gate: The players reconstruct the assigned linear combination of base sharings towards the designated output player.

Now, we are left with the problem of opening known linear combinations of base values towards designated players. For every multiplication gate, we need $2n$ reconstructions (one towards every player), and for every output gate, we need 1 reconstruction. Hence, in total we need to reconstruct $2nc_M + c_O$ linear combinations of 2D*-sharings. This job is, as usual, divided into n^2 segments, each with at most $L = \lceil (2nc_M + c_O)/n^2 \rceil$ reconstructions. Each reconstruction is processed non-robustly, and at the end of the segment, the players verify that no fault has occurred. In the non-robust reconstruction the receiver either obtains the right value, or he observes a fault, stops the further processing of this segment and only joins again in the fault handling procedure.

Protocol ComputationPhase.

For each segment with L reconstructions, the following steps are executed. If in a segment a fault is detected in Step 2., then Step 3 is executed to localize a new dispute pair E, which is included in Δ, i.e., $\Delta \leftarrow \Delta \cup \{E\}$, and the failed segment is repeated.

1. PRIVATE COMPUTATION: Execute the following for each output operation.[15] Denote the designated output player with P_k, the publicly known linear combination for the output operation with $\mathcal{L}$, and the 2D*-shared base values used in the linear combination with $s^{(1)}, s^{(2)}, \ldots$. Furthermore, we denote the share and shares-shares of P_i by $s_i^{(m)}, s_{1i}^{(m)}, \ldots, s_{ni}^{(m)}$, respectively, and the polynomial used for the second-level sharing of $s_i^{(m)}$ by $f_i^{(m)}(x)$.
   1.1 Every P_i with $\{P_i, P_k\} \notin \Delta$ sends his linearly combined share $s_i = \mathcal{L}(s_i^{(1)}, s_i^{(2)}, \ldots)$ to P_k, who receives a message in $\mathcal{F} \cup \{\epsilon\}$.[16]
   1.2 If P_k received *all* shares s_i he was supposed to get (i.e., there was no empty message ϵ), and the received shares lie on a polynomial $f(x)$ of degree t, he computes the output value as $s = f(0)$; otherwise P_k observes a fault and aborts the segment, i.e., for the rest of the segment, P_k only sends empty messages.
2. FAULT DETECTION: Every player $P_i \in \mathcal{P} \setminus \mathcal{X}$ broadcasts the index q_i of the first failed reconstruction operation, respectively $\perp$ if he successfully completed the segment. If all players broadcast $\perp$, then the evaluation of the current segment succeeded

[15] All output operations at the same level in the circuit can be executed in parallel.

[16] It is legal for an honest player P_i to send the empty message ϵ to P_k, namely when P_i has observed a fault in an earlier gate. Hence, P_k must accept the empty message as valid.

3. FAULT LOCALIZATION: Execute the following steps for the player P_k with the smallest q_k, for the failed reconstruction operation with index q_k:
   3.1 Every player P_i with $\{P_k, P_i\} \notin \Delta$ sends the polynomial $f_i(x) = \mathcal{L}(f_i^{(1)}, f_i^{(2)}, \ldots)$ and all share-shares $s_{ji}(x) = \mathcal{L}(s_{ji}^{(1)}, s_{ji}^{(2)}, \ldots)$ to P_k.
   3.2 If for some P_i with $\{P_k, P_i\} \notin \Delta$, P_k did not receive s_i in Step 1.1, or the provided polynomial $f_i(x)$ is inconsistent with s_i (i.e., $f_i(0) \neq s_i$), then P_k broadcasts i, and the fault localization terminates with $E = \{P_k, P_i\}$.
   3.3 P_k identifies two players P_i, P_j with $\{P_k, P_i\} \notin \Delta$ and $\{P_k, P_j\} \notin \Delta$, such that $f_i(j) \neq s_{ij}$,[17] and broadcasts $(i, j, s_{ij}, f_i(j))$.
   3.4 Both P_i and P_j broadcast a bit indicating whether or not they agree with the values broadcasted by P_k. If P_i (respectively P_j) disagrees, the fault localization terminates with $E = \{P_k, P_i\}$ (respectively $E = \{P_k, P_j\}$).
   3.5 As both P_i and P_j agree with s_{ij} respectively $f_i(j)$ as broadcasted by P_k, and as $f_i(j) \neq s_{ij}$, either P_i or P_j delivered a wrong value to P_k. P_j can use the information checking scheme to prove to P_k the correctness of s_{ij}. However, there are no authentication tags for s_{ij} itself, but s_{ij} is computed as a publicly known linear combination $\mathcal{L}$ of base sharings, for which authentication tags exist (one authentication tag for all share-shares x_{ij} of each segment), respectively which are Kudzu and hence publicly known. Hence, P_j executes the protocol IC-Reveal for revealing the provably correct share-shares x_{ij} of every base sharing x, and if P_k accepts all invocations and the linear combination on the share-shares yields s_{ij}, then P_k broadcasts i and $E = \{P_k, P_i\}$, otherwise, P_k broadcasts j and $E = \{P_k, P_j\}$.

Lemma 11. *If all base values are correctly 2D*-shared and all multiplication triples are correct and random, then with overwhelming probability, the circuit evaluation as described above is correct, robust and private. The protocol communicates* $\mathcal{O}((c_I n^2 + c_M n^2 + c_R n^2 + c_O n + n^4)\kappa)$ *and broadcasts* $\mathcal{O}(n^3)$ *field elements.*

Proof. Once the base values are correctly 2D*-shared, the computation phase is purely deterministic. An honest player will never reconstruct a wrong secret: He receives shares from all players he is not in dispute with (otherwise he does not reconstruct at all), hence there are at least $t + 1$ correct shares from the honest players which prevent him from reconstructing a wrong value. Hence, the adversary cannot falsify the outputs of honest players, he can only prevent them from reconstructing. In this case, a fault is detected, a new dispute is localized and included in Δ, and the segment is repeat till eventually all honest players reconstruct all their outputs.

In order to argue about the privacy of the protocol, we observe that share-shares x_{ij} are revealed only when P_i and P_j disagree on some value s_{ij}, hence either P_i or P_j is corrupted. By revealing these values, the adversary obtains no additional information. □

[17] The existence of such a pair (P_i, P_j) is guaranteed due to the correctness of the base 2D*-sharings.

7 The New MPC Protocol and Conclusions

The new MPC protocol consists of the three described phases:

Protocol MPC.
1. Invoke PreparationPhase to prepare $c_M + c_R$ random 2D*-shared multiplication triples.
2. Invoke InputPhase to provide 2D*-sharings of the c_I inputs.
3. Invoke ComputationPhase to compute and reconstruct the outputs towards the specified players.

Theorem 1. *A set of n players communicating over a secure synchronous network, can evaluate an agreed function of their inputs securely against an unbounded active adaptive adversary corrupting up to $t < n/2$ of the players with communicating $\mathcal{O}(c_I n^2 + c_M n^2 + c_R n^2 + c_O n + n^5 \kappa)$ field elements and broadcasting $\mathcal{O}(n^3)$ field elements, where c_I, c_M, c_R, c_O denote the number of input gates, multiplication gates, random gates, and output gates, respectively.*

Note that for large enough circuits, the costs for simulating the $\mathcal{O}(n^3)$ broadcast invocations are dominated by the normal communication costs, such that the overall communication complexity is (up to a constant factor) the same as the one of passively secure MPC protocols [BGW88].

However, for very small circuits, the $\mathcal{O}(n^3)$ broadcasts are dominating the overall costs. Note that even in this case, our protocol is substantially more efficient than the most efficient previously known protocol for the same model [CDD+99], which broadcasts $\Omega(n^5)$ field elements *per multiplication*.

Acknowledgments

We would like to thank Micha Riser for the fruitful discussions, and the anonymous referees for their helpful comments.

References

[Bea91a] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO '91*, LNCS 576, pp. 420–432, 1991.

[Bea91b] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, pp. 75–122, 1991.

[BGP92] P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer Science Research*, pp. 313–322, 1992. Preliminary version in Proc. 21st STOC, 1989.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pp. 1–10, 1988.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th STOC*, pp. 11–19, 1988.

[CDD+99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT '99*, LNCS 1592, pp. 311–326, 1999.

[CDG87] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO '87*, LNCS 293, pp. 87–119, 1987.

[CW79] L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(4):143–154, 1979. Preliminary version in Proc. 9st STOC, 1977.

[CW92] B. A. Coan and J. L. Welch. Modular construction of a Byzantine agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992. Preliminary version in Proc. 8th PODC, 1989.

[DS82] D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. In *Proc. 14th STOC*, pp. 401–407, 1982.

[GHY87] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *CRYPTO '87*, LNCS 293, pp. 135–155, 1987.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th STOC*, pp. 218–229, 1987.

[HM01] M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *CRYPTO '01*, LNCS 2139, pp. 101–118, 2001.

[HMP00] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In *ASIACRYPT '00*, LNCS 1976, pp. 143–161, 2000.

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

[PW92] B. Pfitzmann and M. Waidner. Unconditional Byzantine agreement for any number of faulty processors. In *Proc. 9th STACS*, LNCS 577, 1992.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pp. 73–85, 1989.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. 23rd FOCS*, pp. 160–164, 1982.