Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority*

Eli Ben-Sasson[1], Serge Fehr[2], and Rafail Ostrovsky[3]

[1] Department of Computer Science, Technion, Haifa, Israel,
and Microsoft Research New-England, Cambridge, MA.
eli@cs.technion.ac.il

[2] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands.
serge.fehr@cwi.nl

[3] Department of Computer Science and Department of Mathematics, UCLA.
rafail@cs.ucla.edu

Abstract. In the setting of unconditionally-secure MPC, where dishonest players are unbounded and no cryptographic assumptions are used, it was known since the 1980's that an honest majority of players is both necessary and sufficient to achieve privacy and correctness, assuming secure point-to-point and broadcast channels. The main open question that was left is to establish the exact communication complexity.

We settle the above question by showing an unconditionally-secure MPC protocol, secure against a dishonest minority of malicious players, that matches the communication complexity of the best known MPC protocol in the honest-but-curious setting. More specifically, we present a new n-player MPC protocol that is secure against a computationally-unbounded malicious adversary that can adaptively corrupt up to $t < n/2$ of the players. For polynomially-large binary circuits that are not too unshaped, our protocol has an amortized communication complexity of $O(n \log n + \kappa/n^{const})$ bits per multiplication (i.e. AND) gate, where κ denotes the security parameter and $const \in \mathbb{Z}$ is an arbitrary non-negative constant. This improves on the previously most efficient protocol with the same security guarantee, which offers an amortized communication complexity of $O(n^2\kappa)$ bits per multiplication gate. For any κ polynomial in n, the amortized communication complexity of our protocol matches the $O(n \log n)$ bit communication complexity of the best known MPC protocol with passive security.

We introduce several novel techniques that are of independent interest and we believe will have wider applicability. One is a novel idea of computing authentication tags by means of a *mini MPC*, which allows us to avoid expensive double-sharings; the other is a *batch-wise multiplication verification* that allows us to speedup Beaver's "multiplication triples".

1 Introduction

Background. In secure multiparty computation (MPC), a set of n players wish to evaluate an arbitrary but fixed function F on private inputs. The function F is known to all the players and it is typically given as an arithmetic circuit $\mathcal{C}$ over some finite field $\mathbb{F}$. It should be guaranteed that the inputs remain private and at the same time that the output of the computation is correct, even in the presence of an *adversary* that can *corrupt* a certain number t of the players. In case of a *passive* adversary, corrupt players simply reveal all their information to the adversary but otherwise keep following the protocol specification; in case of an *active* adversary, a corrupt player is under full control of the adversary and may arbitrarily misbehave during the protocol execution. By default, the goal is to obtain security against an active adversary.

The problem of MPC was initially introduced by Yao [23], with the first generic solutions presented in [17, 9]. These first protocols offered cryptographic (aka. computational) security, meaning that the adversary is assumed to be computationally bounded, and can tolerate up to $t < n/2$ corrupt players. Subsequently, it was shown in [8, 5] that in a setting with perfectly-secure point-to-point communication and with up to $t < n/3$ corrupt players, MPC is possible with unconditional and even perfect security.[1] Finally, in [21, 1] it was shown that if a secure broadcast primitive is given — in addition to the secure point-to-point communication — then unconditionally (but not perfectly) secure MPC is possible against up to $t < n/2$ corrupt players.

* This is the full version of [6].

[1] Unconditional/perfect security means a computationally unbounded adversary and negligible/zero failure probability.

These early results showed that MPC is possible in principle (in different settings), but they perform rather poorly in terms of communication complexity, i.e., the number of bits that the players need to communicate throughout the protocol. Over the years, a lot of effort has been put into improving the communication complexity of MPC protocols. The table in Figure 1 shows recent achievements and the state of the art in the settings $t < n/2$ (cryptographic or with broadcast) and $t < n/3$ (perfect or unconditional, without broadcast). Additional efficiency improvements are possible if one is willing to sacrifice on the resilience and lower the corruption threshold t by a small constant fraction, as shown in [13, 15, 14]. Indeed, lowering t enables to apply several powerful tools, like *packed secret sharing* or *committee selection*. We do not consider this option here, but aim for optimal resilience.

Adv	**Resilience**	**Security**	**Communication**	**Ref**
passive	$t < n/2$	perfect	$O(c_M n \log n + n^2 \log n)$	[16]
active	$t < n/2$	cryptographic	$O(c_M n^2 \kappa + n^3 \kappa)$	[19]
active	$t < n/2$	cryptographic	$O(c_M n \kappa + n^3 \kappa)$	[20]
active	$t < n/2$	cryptographic	$O(c_M n \log n) + poly(n\kappa)$	[16]
active	$t < n/3$	unconditional	$O(c_M n^2 \kappa) + poly(n\kappa)$	[18]
active	$t < n/3$	unconditional	$O(c_M n \log n + d_M n^2 \log n) + poly(n\kappa)$	[16]
active	$t < n/3$	perfect	$O(c_M n \log n + d_M n^2 \log n + n^3 \log n)$	[4]
active	$t < n/2$	unconditional	$O(c_M n^5 \kappa + n^4 \kappa) + O(c_M n^5 \kappa)\mathcal{BC}$	[10]
active	$t < n/2$	unconditional	$O(c_M n^2 \kappa + n^5 \kappa^2) + O(n^3 \kappa)\mathcal{BC}$	[3]

Fig. 1. Comparison of recent MPC protocols for binary circuits. n denotes the number of players, κ the security parameter (which we assume to be $\geq \log n$), c_M the number of multiplication gates in the circuit (which we assume dominates the number of in- and outputs), and d_M the multiplicative depth of the circuit. The communication complexity counts the number of bits that are communicated in total in an execution, plus, in the setting where a broadcast primitive is needed, the number of bits broadcasted. For circuits over a larger field $\mathbb{F}$, the $\log n$-terms should be replaced by $\log(\max\{n, |\mathbb{F}|\})$.

We can see from Figure 1 that there is a significant discrepancy between the cryptographic setting with $t < n/2$, or, similarly, the unconditional/perfect setting with $t < n/3$, versus the unconditional setting with $t < n/2$. In the former, MPC is possible for binary circuits with a near-linear amortized communication complexity of $O(n \log n)$ bits per multiplication gate.[2] In the latter, the best known protocol has an amortized communication complexity of $O(n^2\kappa)$ bits per multiplication gate. This is not very surprising, since it is probably fair to say that the unconditional setting with $t < n/2$ is the most difficult one to deal with. The reason is that no cryptographic tools can be used, like commitments or signatures, as in the cryptographic setting, nor can we use techniques from error correcting codes, as in the case $t < n/3$. Therefore, achieving near-linear amortized communication complexity for the setting of unconditional security and $t < n/2$ has remained a challenging open problem.

We note that, in any of the three settings, $O(n \log n)$ bits per multiplication gate seems to be hard to beat, since not even the best known protocol with *passive* security [16] does better than that.

Our Result. For an arbitrary arithmetic circuit over a finite field $\mathbb{F}$, we show a novel MPC protocol with unconditional security and corruption threshold $t < n/2$, which has a communication complexity of $O(c_M(n\phi + \kappa) + d_M n^2 \kappa + n^7 \kappa)$ bits plus $O(n^3\kappa)$ broadcasts, where $\phi = \max\{\log n, \log |\mathbb{F}|\}$. Hence, for binary circuits that are not too "narrow" (meaning that the multiplicative depth d_M is sufficiently smaller than the number of multiplication gates), our protocol achieves an amortized communication complexity of $O(n \log n + \kappa)$ bits per multiplication gate. Furthermore, for any non-negative constant $const \in \mathbb{Z}$, a small modification to our protocol gives $O(n \log n + \kappa/n^{const})$ bits per multiplication gate, so that if $\kappa = O(n^{const+1})$, i.e., κ is at most polynomial in n, we obtain an amortized communication complexity of $O(n \log n)$ bits. Thus, our results show that even in the challenging setting of unconditional security with $t < n/2$, near-linear MPC is possible. Unless there is an additional improvement in the passive setting, this pretty much settles the question of the asymptotic complexity of unconditionally-secure MPC.

[2] By *amortized* communication complexity we mean under the assumption that the circuit is large enough so that the terms that are independent of the size of the circuit are irrelevant.

We would like to point out that the restriction on the multiplicative depth of the circuit, necessary for the claimed near-linear communication complexity per multiplication gate to hold, is also present in the easier $t < n/3$ setting for the protocols with near-linear communication complexity [16, 4]; whether it is an inherent restriction is not known.

Techniques. We borrow several techniques from previous constructions of efficient MPC protocols. For instance, we make use of the *dispute control* technique introduced in [3], and the (near) linear passively-secure multiplication technique from [16]. However, our new protocol and its near-linear amortized communication complexity is to a great extent due to two new techniques, which we briefly discuss here. More details will be given in Section 2.7 and Section 3.2.

Efficient batch verification of multiplication triples. The first technique allows to efficiently verify that a large list of N shared multiplication-triples are correct, i.e., satisfy the required multiplicative relation. These multiplication triples are used in order to implement Beaver's method of evaluating multiplication gates, and our new protocol allows us to guarantee all N triples in one shot using communication complexity that is (nearly) independent of N.

Our new technique is inspired by a method that plays an important role in the construction of PCP proofs. Given oracle access to three sequences of bits, or elements from a "small" finite field, $a^1, \dots, a^N$, $b^1, \dots, b^N$ and $c^1, \dots, c^N$, we wish to verify that $a^i \cdot b^i = c^i$ for all $i = 1, \dots, N$. The procedure should be query-efficient, i.e., (much) more efficient than when querying and verifying all triples. Suppose the triples are encoded as low-degree polynomials. This means, we are given oracle access to evaluations of polynomials f and g of degree $< N$ and h of degree $< 2N - 1$, with $f(x_i) = a^i$, $g(x_i) = b^i$ and $h(x_i) = c^i$ for all $i \in \{1, \dots, N\}$, where $x_1, \dots, x_N$ are fixed disjoint points and h is supposed to be $h = f \cdot g$. The key observation is this: by the fundamental theorem of algebra, if $f \cdot g \neq h$ then $f(\sigma) \cdot g(\sigma) \neq h(\sigma)$ except with probability at most $\frac{2N-1}{|\mathbb{K}|}$ for a randomly chosen $\sigma \in \mathbb{K}$, and for any suitably large extension field $\mathbb{K}$.

In our setting, it will turn out that we can indeed enforce the shared multiplication triples to be encoded via low-degree polynomials as above. So, by the above technique, it is possible to verify N multiplication triples with just *one* (random) query to f, g and h, and thus with a communication complexity that essentially only depends on the aspired error probability.

In independent work [12], Cramer *et al.* propose a 2-party batch zero-knowledge proof for committed multiplication triples. The techniques used there show some resemblance, but there are also differences due to the fact that in our setting, the a^i, b^i and c^i's are not known to any party.

Multiparty-computing the authentication tags Our other technique is a new way to "commit" the players to their shares, so that dishonest players who lie about their shares during reconstruction are caught. This is necessary in the setting $t < n/2$, where plain Shamir shares do not carry enough redundancy to reconstruct in the presence of incorrect shares.

The way we "commit" player P_i to his share σ_i is by attaching an *authentication tag* τ to σ_i, where the corresponding *authentication key* is held by some other player V, acting as *verifier*.[3] The reader may think of τ as $\tau = \mu \cdot \sigma_i + \nu$ over some large finite field, where (μ, ν) forms the key. It is well known and easy to see that if P_i does not know the key (μ, ν), then he is not able to come up with $\sigma_i' \neq \sigma_i$ and τ' such that $\tau' = \mu \cdot \sigma_i' + \nu$, except with small probability. Thus, incorrect shares can be detected and filtered out.

This idea is not new, and actually goes back to [21], but in all previous work the tag τ is *locally* computed by some party, usually the dealer that prepared the share σ_i. Obviously, this requires that the dealer *knows* the key (μ, ν); otherwise, he cannot compute $\tau = \mu \cdot \sigma_i + \nu$. As a consequence, if the dealer is dishonest, the authentication tag τ is useless, because with the knowledge of the key, an authentication tag τ' for an incorrect share σ_i' can easily be forged. In previous work, as in [21, 10, 3], this problem was overcome by means of a *double* sharing, where every share σ_i is again shared, and the authentication tags are attached to the second-level shares. However, such a double sharing obviously leads to a (at least) quadratic communication complexity.

Instead, here we propose to compute the tag τ by means of a *mini MPC*, to which P_i provides his share σ_i as input, and V his key (μ, ν), and the tag τ is securely computed jointly by all the players. This way, no one beyond V learns the key (μ, ν), and forging a tag remains hard, and no expensive double sharing is necessary.

[3] Actually, σ_i comes along with n tags, one for each player acting as verifier V.

At first glance this may look hopeless since MPC typically is very expensive, and we cannot expect to increase the efficiency of MPC by using an expensive MPC as subprotocol. What saves us is that our mini MPC is for a very specific function in a very specific setting. We use several tricks, like re-using parts of the authentication key, batching etc., to obtain a *tailored* mini MPC for computing the tag τ, with an amortized communication complexity that has no significant impact. One of the crucial new tricks is to make use of the fact that Shamir's secret sharing scheme is "symmetric" in terms of what is the shared secret and what are the shares; this allows us to avoid having to re-share the share σ_i for the mini MPC, but instead we can use the other shares σ_j as shares of σ_i.

2 Near-Linear MPC: Our Result and Approach

2.1 Communication and Corruption Model

We consider a set of $n = 2t + 1$ players $P_1, \dots, P_n$, which are connected by means of a complete network of secure synchronous communication channels. Additionally, we assume a broadcast channel, available to all the players. For simplicity, we assume the broadcast channel to broadcast single bits; longer messages are broadcasted bit-wise. For a protocol that instructs the players to communicate (in total) X bits and to broadcast Y bits, we say that the protocol has communication complexity $X + Y \cdot \mathcal{BC}$.

We consider a computationally-unbounded active adversary that can adaptively corrupt up to t of the players. Adaptivity means that the adversary can corrupt players during the execution of the protocol, and depending on the information gathered so far. Once a player is corrupted, the adversary learns the internal state of the player, which consists of the complete history of that player, and takes over full control of that player and can make him deviate from the protocol in any desired manner.

For any given arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$, the goal is to have a protocol that permits the n players to securely evaluate $\mathcal{C}$ on their private inputs. For simplicity, we assume that all the players should learn the entire result. Security means that the adversary cannot influence the result of the computation more than by selecting the inputs for the corrupt players, and the adversary should learn nothing about the uncorrupt players' inputs beyond what can be deduced from the result of the computation. This should hold unconditionally, meaning without any computational restrictions on the adversary, and up to a negligible failure probability ε.

2.2 Main Result

For an arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$, we denote the respective numbers of input, output, addition, and multiplication gates in $\mathcal{C}$ by c_I, c_O, c_A, and c_M, and we write $c_{tot} = c_I + c_O + c_M$ (not counting c_A). Furthermore, we write d_M to denote its multiplicative depth, i.e., the maximal number of multiplication gates on any path from an input gate to an output gate.

Theorem 1. *For every $n, \kappa \in \mathbb{N}$, and for every arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$ with $|\mathbb{F}| \leq 2^{\kappa+n}$, there exists an n-party MPC protocol that securely computes $\mathcal{C}$ against an unbounded active adaptive adversary corrupting up to $t < n/2$ players, with failure probability $\varepsilon \leq O(c_{tot} n)/2^\kappa$ and communication complexity $O(c_{tot} \cdot (n\phi + \kappa) + d_M n^2 \kappa + n^7 \kappa) + O(n^3 \kappa) \cdot \mathcal{BC}$, where $\phi = \max\{\log |\mathbb{F}|, \log n\}$.*
More generally, for any $const \in \mathbb{Z}$, there exists such an n-party MPC protocol with communication complexity $O(c_{tot} \cdot (n\phi + \kappa/n^{const}) + d_M n^2 \kappa + n^7 \kappa) + O(n^3 \kappa) \cdot \mathcal{BC}$.

Theorem 1 guarantees that for large enough circuits that are not too "narrow", meaning that the multiplicative depth d_M is significantly smaller than the number c_M of multiplication gates (e.g. $d_M \leq c_M/(n\kappa)$ is good enough), the communication complexity per multiplication gate (assuming that c_M dominates c_I, c_O and c_R) is $O(n\phi + \kappa/n^{const})$ bits, i.e., $O(n \log n + \kappa/n^{const})$ for binary circuits, for an arbitrary non-negative $const \in \mathbb{Z}$. Recall, the best previous MPC scheme in this setting [3] required $O(n^2\kappa)$ bits per multiplication gate. For simplicity, we focus on the case $const = 0$ and merely give some indication on how to adapt the same for larger $const$.

2.3 The Set Up

We are given positive integers n and κ, and an arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$. We assume that $|\mathbb{F}| \geq 2n^2$ (or $|\mathbb{F}| \geq 2n^{2+const}$ for an arbitrary *const*) — otherwise we consider $\mathcal{C}$ over an appropriate extension field[4] — and we write $\phi = \log(|\mathbb{F}|)$, i.e., ϕ denotes the number of bits needed to represent an element in $\mathbb{F}$. We may assume that $\kappa \geq n$ (otherwise, we set $\kappa = n$) and thus that κ is an integer multiple of n. We fix an extension field $\mathbb{K}$ of $\mathbb{F}$ such that $|\mathbb{K}| \geq 2^{2(\kappa+n)}$. Finally, we set $M = 2(c_M + c_O + c_I)$.

As convention, we write elements in $\mathbb{F}$ as Roman letters, and elements in $\mathbb{K}$ as Greek letters. Note that $\mathbb{F}$ is naturally a subset of $\mathbb{K}$, and thus for $s \in \mathbb{F}$ and $\lambda \in \mathbb{K}$, the product $\lambda \cdot s$ is a well defined element in $\mathbb{K}$. Also note that by fixing an $\mathbb{F}$-linear bijection $\mathbb{F}^e \to \mathbb{K}$, where e is the extension degree $e = [\mathbb{K}:\mathbb{F}]$ we can understand a vector $(s^1, \ldots, s^e) \in \mathbb{F}^e$ as a field element $\sigma \in \mathbb{K}$, and a vector $(s^1, \ldots, s^{q \cdot e}) \in \mathbb{F}^{q \cdot e}$ for $q \in \mathbb{N}$ as a vector $\boldsymbol{\sigma} = (\sigma^1, \ldots, \sigma^q) \in \mathbb{K}^q$ of q field elements in $\mathbb{K}$.

2.4 Dispute Control

We make use of the *dispute control* framework due to Beerliová-Trubíniová and Hirt. The idea of dispute control is to divide (the different phases of) the MPC protocol into n^2 *segments* (of equal "size"), and to execute the segments sequentially. If the execution of a segment should fail due to malicious behavior of some corrupt parties, then two players are identified that are in dispute and of which at least one must be corrupt. Then, the failed segment is freshly re-executed, but now in such a way that the two players in dispute will *not* be able to get into dispute anymore, during this segment and during all the remaining segments. This ensures that overall there can be at most n^2 disputes (actually fewer, because two uncorrupt players will never get into a dispute), and therefore at most n^2 times a segment needs to be re-executed. This means that overall there are at most $2n^2$ executions of a segment.

We will show that (if d_M is small enough) any segment of size $m = M/n^2$ can be executed with bit communication complexity $O\big(m(n\phi + \kappa) + n^5\kappa\big) + O(n\kappa) \cdot \mathcal{BC}$; it thus follows that the communication complexity of the overall scheme is $2n^2 \cdot O\big(m(n\phi + \kappa) + n^5\kappa\big) = O\big(M(n\phi + \kappa) + n^7\kappa\big)$ bits plus $O(n^3\kappa) \cdot \mathcal{BC}$, which amounts to $O(n\phi + \kappa)$ bits per multiplication gate for large enough circuits.

A dispute between two players P_i and P_j typically arises when player P_j claims to have received message *msg* from P_i whereas P_i claims that he had actually sent $msg' \neq msg$ to P_j. In order to ensure that two players P_i and P_j in dispute will not get into a new dispute again, they will not communicate anymore with each other. This is achieved by means of the following two means:

(1) If P_i is supposed to share a secret w and distribute the shares to the players, then he chooses the sharing polynomial so that P_j's share w_j vanishes, and thus there is no need to communicate the share, P_j just takes $w_j = 0$ as his share. Using the terminology from [3], we call such a share that is enforced to be 0 a *Kudzu* share (see also Section 2.5).
(2) For other messages that P_i needs to communicate to P_j, he sends to P_j via a *relay*: the first player P_r that is not in dispute with P_i and not with P_j.

In order to keep track of the disputes and the players that were caught cheating, the players maintain two sets, $\mathcal{C}orr$ and $\mathcal{D}isp$, which at the beginning of the execution are both initialized to be empty. Whenever the players jointly identify a player P_i to be corrupt, then P_i is added to $\mathcal{C}orr$. Additionally, $\{P_i, P_j\}$ will be added to $\mathcal{D}isp$ for every $j \in \{1, \ldots, n\}$. Whenever there is a dispute between two players P_i and P_j, so that one of them must be corrupt but it cannot be resolved which of the two, then $\{P_i, P_j\}$ is added to $\mathcal{D}isp$. Whenever a player P_i is in dispute with more than t players, then he must be corrupt and is added to $\mathcal{C}orr$ (and $\mathcal{D}isp$ is updated accordingly). We write $\mathcal{D}isp_i$ for the set of all players P_j with $\{P_i, P_j\} \in \mathcal{D}isp$. Players that are in dispute (with some other players) still take part in the protocol, but they do not communicate anymore with each other. Players in $\mathcal{C}orr$, i.e., players that have been identified to be corrupt, are excluded from (the remainder of) the protocol execution. We do not always make this explicit in the description of the protocol when we quantify over all players but actually mean all players not in $\mathcal{C}orr$. Also, we do not make it always explicit but understand it as clear that whenever a new dispute is found, the remainder of the execution of the current segment is skipped, and the segment is freshly executed with the updated $\mathcal{D}isp$ (and $\mathcal{C}orr$).

[4] In this case one has to make sure that the inputs provided by the players belong to the original base field; this can easily be taken care of by means of our techniques, without increasing the asymptotic communication complexity.

2.5 The Different Sharings

We will be using different variants and extensions of Shamir's secret sharing scheme [22]. We introduce here these different versions and the notation that we will be using for the remainder of the paper. We consider the field $\mathbb{F}$ from Section 2.3, and fix distinct elements $x_0, x_1, \dots, x_n \in \mathbb{F}$ with $x_0 = 0$. We also fix an additional $2n^2 - n - 1$ elements $x_{n+1}, \dots, x_{2n^2-1}$ with the property that every pair x_i, x_j with $i \neq j \in \{0, \dots, 2n^2 - 1\}$ is disjoint; these additional elements will be used later on. It may be convenient to view the different kinds of sharings we introduce below as different *data structures* for representing an element $w \in \mathbb{F}$ by data held among the players.

- A *degree-t (Shamir) sharing* of $w \in \mathbb{F}$ consists of n shares $w_1, \dots, w_n \in \mathbb{F}$ of the following form: there exists a sharing polynomial $f(X) \in \mathbb{F}[X]$ of degree at most t such that $w = f(0)$ and $w_j = f(x_j)$ for $j \in \{1, \dots, n\}$. Furthermore, share w_j is held by player P_j for $j \in \{1, \dots, n\}$. We denote such a sharing as $[w]$. If a designated player P_d (e.g. the dealer) knows all the shares, and thus also w, we indicate this by denoting the sharing as $[w]_d$.
- A *degree-*$2t$ *(Shamir) sharing* of $w \in \mathbb{F}$ is defined as the degree-t sharing above, except that the degree of the sharing polynomial f is at most $2t$. We write $\langle w \rangle$ for such a sharing, and $\langle w \rangle_d$ for such a sharing when P_d knows all the shares.
- A *twisted* degree-t sharing of $w \in \mathbb{F}$ with respect to player P_i, denoted as $\lceil w \rfloor^i$, consists of $n-1$ shares $w_1, ..., w_{i-1}, w_{i+1}, ..., w_n \in \mathbb{F}$, of the following form: there exists a sharing polynomial $f(X) \in \mathbb{F}[X]$ of degree at most t such that $w = f(x_i)$, $f(0) = 0$, and $w_j = f(x_j)$ for $j \in \{1, \dots, n\} \setminus \{i\}$.[5] Share w_j for $j \in \{1, \dots, n\} \setminus \{i\}$ is known to player P_j. We write $\lceil w \rfloor^i_d$ for such a sharing when P_d knows all the shares.
- A *twisted* degree-$2t$ sharing of $w \in \mathbb{F}$ with respect to P_i, denoted as $\langle w \rangle^i$ respectively $\langle w \rangle^i_d$ when P_d knows all the shares, is defined as the twisted degree-t sharing above, except that the degree of the sharing polynomial f is at most $2t$.
- A *two-level (degree-t/sum) sharing* $[\![w]\!]$ consists of n degree-t Shamir sharings $[w(1)]_1, ..., [w(n)]_n$ with $w = \sum_d w(d)$.[6] The shares $w_1(d), \dots, w_n(d)$ given by $[w(d)]_d$ for $d \in \{1, \dots, n\}$ then define a degree-t sharing $[w]$ of w by means of $w_j = \sum_d w_j(d)$ for $j \in \{1, \dots, n\}$ (see Figure 2, center left). We point out that the second level shares $w_i(d)$ can be understood as Shamir shares of the sum-shares $w(d)$ of w, as well as sum-shares of the Shamir shares w_i of w.
- A *two-level (degree-2t/sum) sharing* $\langle\!\langle w \rangle\!\rangle$ is defined similar to above as $\langle\!\langle w \rangle\!\rangle = (\langle w(1) \rangle_1, ..., \langle w(n) \rangle_n)$ with $w = \sum_d w(d)$.

The above list merely specifies the structures of the different sharings, but does not address privacy. In our scheme, the different sharings will be prepared in such a way that the standard privacy requirement holds: the shares of any t players reveals no information on the shared secret. In the case of a *twisted* sharing $\lceil w \rfloor^i$, privacy is slightly more subtle. Because player P_i is given no share, but, on the other hand, the sharing polynomial vanishes at 0, privacy will only hold in case P_i is (or gets) corrupted, so that the t corrupted players miss one polynomial evaluation; this will be good enough for our purpose.

We note that the players can, by means of local computations, perform certain computations on the sharings. For instance, by linearity of Shamir's secret sharing scheme, it follows that if the players locally add their shares of a degree-t sharing $[v]$ of v to their shares of a degree-t sharing $[w]$ of w, then they obtain a degree-t sharing $[v + w]$ of $v + w$. We denote this computation as $[v] + [w] = [v + w]$. Also, multiplication with a known constant: $c[w] = [cw]$, or adding a known constant: $[w] + d = [w + d]$, can be performed by means of local computations. This holds for all the different sharings discussed above: $\langle v \rangle + c\langle w \rangle + d = \langle v + cw + d \rangle$, $[\![v]\!] + c[\![w]\!] + d = [\![v + cw + d]\!]$ etc. Furthermore, locally multiplying the shares of two degree-t shared secrets results in a degree-$2t$ sharing of the product: $[v] \cdot [w] = \langle v \cdot w \rangle$. Finally, locally multiplying the shares $[v]$ of an ordinarily degree-t shared secret with the shares $\lceil w \rfloor^i$ of a twisted degree-t shared secret results in a twisted degree-$2t$ sharing of the product of P_i's share v_i of $[v]$ and w: $[v] \cdot \lceil w \rfloor^i = \langle v_i \cdot w \rangle^i$. This property of a twisted sharing is of crucial importance to us; thus, we encourage the reader to verify this claim.

[5] Thus, instead of plugging the secret into the evaluation at 0 (i.e. into the constant coefficient of f), we pug it into the evaluation at x_i, and require $f(0)$ to vanish and give player P_i no share.

[6] We point out that $w(1), ..., w(n)$ are simply n elements in $\mathbb{F}$, indexed by $d = 1, \dots, n$, that add up to w, and they should not be understood as function evaluations. Our convention is to write $w(1), ..., w(n)$ as sum-shares of w, and $w_1, \dots, w_n$ as Shamir shares of w, and $w_1(d), \dots, w_n(d)$ as Shamir shares of $w(d)$, etc.

We point out that opening such a product of sharings, like $\langle v \cdot w \rangle = [v] \cdot [w]$, reveals more information on v and w than just their product. This will be of no concern to us, because in our scheme, such sharings will only be opened in the form of $\langle u + v \cdot w \rangle = \langle u \rangle + [v] \cdot [w]$, i.e., when masked with a random degree-$2t$ sharing, which ensures that no information on u, v, w is revealed beyond $u + v \cdot w$.

Borrowing the terminology from [3], we say that a sharing $[s]_d$ has *Kudzu* shares, if the share s_j of every player P_j that currently is in $\mathcal{D}isp_d$ is set to $s_j = 0$, i.e., the sharing polynomial $f(x)$ is such that $f(x_j) = 0$ for every $P_j \in \mathcal{D}isp_d$. The same terminology correspondingly applies to sharings $\langle s \rangle_d$, $\lceil s \rfloor_d^i$ and $\langle s \rangle_d^i$. Furthermore, a two-level sharing $[\![s]\!]$ is said to have Kudzu shares if $[s(d)]_d$ has Kudzu shares for all $P_d \notin \mathcal{C}orr$, and $[s(d)]_d$ consist of all-0 shares for all $P_d \in \mathcal{C}orr$, and similarly for $\langle\!\langle s \rangle\!\rangle$.

Finally, we would like to point out that due to the linearity, e sharings $[s^1], \ldots, [s^e]$ of secrets $s^1, \ldots, s^e \in \mathbb{F}$ can also be understood and treated as a sharing $[\sigma]$ of $\sigma = (s^1, \ldots, s^e)$, viewed as an element in $\mathbb{K}$ and with shares $\sigma_i \in \mathbb{K}$, by means of a sharing polynomial $f(X) \in \mathbb{K}[X]$, but with the same interpolation points $x_1, \ldots, x_n \in \mathbb{F} \subseteq \mathbb{K}$.

2.6 Protocol Overview

The protocol consists of three phases: the *preparation phase*, the *input phase*, and the *computation phase*. We briefly discuss (the goal of) these three phases here. As discussed in Section 2.4, every phase will be performed in segments; and whenever a segment fails, then a new dispute is found and added to $\mathcal{D}isp$, and the segment is re-executed.

Preparation Phase. In this phase, the following data structure is prepared.

Two-level shared multiplication triples: A list $\mathcal{M}$ of M correctly two-level shared triples $([\![a]\!], [\![b]\!], [\![c]\!])$, where for every[7] $([\![a]\!], [\![b]\!], [\![c]\!]) \in \mathcal{M}$, the values a and b are uniformly distributed in $\mathbb{F}$ (and independent of each other and of the other triples in $\mathcal{M}$) and unknown to the adversary, and $c = a \cdot b$. We write $\cup\mathcal{M}$ for the list of $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ sharings contained in $\mathcal{M}$, i.e., $\cup\mathcal{M} = \bigcup_{\mathcal{M}} \{[\![a]\!], [\![b]\!], [\![c]\!]\}$, where the union is over all $([\![a]\!], [\![b]\!], [\![c]\!]) \in \mathcal{M}$

Local base sharings: The two-level sharings of the multiplication triples are not fully independent. Instead, for every player P_d there exists a list $\mathcal{S}(d)$ of $L = O(M/n)$ so-called *local base sharings* $[s(d)]_d$ with $s(d) \in \mathbb{F}$, such that for every $[\![w]\!] \in \cup\mathcal{M}$, the sharing $[w(d)]_d$ (which is part of $[\![w]\!]$) is a linear combination (with known coefficients) of the local base sharings:[8]

$$[w(d)]_d = \sum_{s(d) \in \mathcal{S}(d)} u_{s(d)} [s(d)]_d + u_\circ .$$

Although there are dependencies among the second-level shares of different $[\![w]\!] \in \cup\mathcal{M}$ (which means we have to pay special attention when revealing those, or the local base sharings), it will be the case that the first-level Shamir sharings $[w]$ are independent among all $[\![w]\!] \in \cup\mathcal{M}$.

For every P_d, the list $\mathcal{S}(d)$ will be divided into n^3 blocks, each block containing L/n^3 sharings $[s(d)]_d$ from $\mathcal{S}(d)$. Each such block, we can write as $[\boldsymbol{\sigma}(d)]_d$ with $\boldsymbol{\sigma}(d) \in \mathbb{K}^q$, and understand it as a list of $q = L/(n^3 e)$ sharings $[\sigma(d)]_d$ of elements $\sigma(d) \in \mathbb{K}$, where $e = [\mathbb{K} : \mathbb{F}]$. As such, $\mathcal{S}(d)$ can now be understood as a list of n^3 sharings $[\boldsymbol{\sigma}(d)]_d$.[9]

Authentication tags: For every player P_d, every block $[\boldsymbol{\sigma}(d)]_d \in \mathcal{S}(d)$, every player P_i holding the shares $\boldsymbol{\sigma}_i(d) \in \mathbb{K}^q$ of block $[\boldsymbol{\sigma}(d)]_d$, and every player P_V (acting as verifier), the following holds. P_V holds a random *long-term authentication key* $\boldsymbol{\mu} \in \mathbb{K}^q$ and a random *one-time authentication key* $\nu \in \mathbb{K}$, and P_i holds the *(one-time) authentication tag*

$$\tau = \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i(d) + \nu \in \mathbb{K} ,$$

[7] We will often use set-notation for lists: for a list $\mathcal{L} = (\ell_1, \ldots, \ell_m)$, the expression $\ell \in \mathcal{L}$ is to be understood as ℓ_i for $i \in \{1, \ldots, m\}$. Also, $\sum_{\ell \in \mathcal{L}} u_\ell \ell$ should be understood as $\sum_{i=1}^m u_i \ell_i$.

[8] As a consequence, even though every player implicitly holds in total $3Mn$ subshares of the $[\![w]\!] \in \cup\mathcal{M}$, he only needs to explicitly store $n \cdot L = O(M)$ values. Thus, to communicate all these subshares (for all the players), only $O(Mn)$ elements in $\mathbb{F}$ need to be communicated, i.e., a linear number per multiplication triple.

[9] We silently assume here that the fraction $L/(n^3 e)$ is an integer, and we will similarly do so for a few other fractions later. We may always do so without loss of generality.

where $\odot$ denotes the standard inner product over $\mathbb{K}$. We stress that ν and, consequently, τ are fresh for every P_d, every block $[\boldsymbol{\sigma}(d)]_d \in \mathcal{S}(d)$, and every P_i and P_V, but $\boldsymbol{\mu}$ is somewhat re-used: P_V uses the same $\boldsymbol{\mu}$ for every P_d (but fresh $\boldsymbol{\mu}$'s for different P_i's) and for n out of the n^3 blocks $[\boldsymbol{\sigma}(d)]_d \in \mathcal{S}(d)$.[10]

This data structure is illustrated in Figure 2.

$$
a \cdot b = c \qquad
\begin{array}{ccccccc}
w & \to & w(1) & w(2) & \cdots & w(n) \\
\Downarrow & & \Downarrow & \Downarrow & \cdots & \Downarrow \\
w_1 & \to & w_1(1) & w_1(2) & \cdots & w_1(n) \\
w_2 & \to & w_2(1) & w_2(2) & \cdots & w_2(n) \\
\vdots & & \vdots & \vdots & & \vdots \\
w_n & \to & w_n(1) & w_n(2) & \cdots & w_2(n)
\end{array}
\qquad
\begin{array}{c}
w(d) \\ \Downarrow \\ w_1(d) \\ w_2(d) \\ \vdots \\ w_n(d)
\end{array}
\in span \left\{
\begin{array}{c}
s(d) \\ \Downarrow \\ s_1(d) \\ s_2(d) \\ \vdots \\ s_n(d)
\end{array}
\right\}
\qquad
\tau = \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i(d) + \nu
$$

Fig. 2. The data structure: multiplication triples (a, b, c) (left), $w \in \{a, b, c\}$ is two-level shared as $[\![w]\!]$ (center left), $[w(d)]_d$ is a linear combination of P_d's local base sharings (center right), and $s_i(d)$ is authenticated within a batch $\boldsymbol{\sigma}_i(d)$ (right).

The purpose of the *authentication tags* (and *keys*) is to be able to identify an incorrect share $\boldsymbol{\sigma}_i(d)$ claimed by a corrupt player P_i. Indeed, it is well known (and goes back to Carter and Wegman [7]) that if the adversary has no information on $\boldsymbol{\mu}$ beyond knowing the tags τ for several $\boldsymbol{\sigma}_i(d)$ with fresh one-time keys ν, then the probability for the adversary to produce $\boldsymbol{\sigma}'_i(d) \neq \boldsymbol{\sigma}_i(d)$ and τ' with $\tau' = \boldsymbol{\mu} \odot \boldsymbol{\sigma}'_i(d) + \nu$ is at most $1/|\mathbb{K}| \leq 2^{-2(\kappa+n)}$. Informally, this means that with the given data structure, a dishonest player P_i will not be able to lie about his share $\boldsymbol{\sigma}_i(d)$ without being caught.

The use of authentication tags to (try to) commit players to their (sub)share is not new. What distinguishes our approach from previous work is that here the tag τ will be computed in a *multi-party fashion* so that no one beyond the verifier P_V knows the corresponding key. This gives us the decisive advantage over previous work.

Input Phase. For every player P_i, and for every input $x \in \mathbb{F}$ of that player to the circuit, a fresh multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$ is chosen from $\mathcal{M}$, and a is reconstructed towards P_i. Then P_i announces $d = x - a$, and the players compute the sharing $[\![x]\!] = d + [\![a]\!]$. The used triple $([\![a]\!], [\![b]\!], [\![c]\!])$ is then removed from $\mathcal{M}$.

Essentially the only thing corrupt players can do to disrupt the computation phase, is to provide incorrect shares when P_i is supposed to reconstruct some shared a. However, because every $[a(d)]_d$ is a linear combination of the local base sharings $[s(d)]_d$, and because players are committed to their local base sharings (block-wise) by means of the authentication tags, players that hand in incorrect shares can be caught.

Computation Phase. The actual computation is done in a gate-by-gate fashion. To start with, we say that the input values are *computed*. Then, inductively, for every gate in the circuit whose input values have already been computed, the corresponding output value of the gate is computed. This is done as follows. Let $[\![x]\!]$ and $[\![y]\!]$ be the sharings of the input values to the gate. If the gate is an addition gate, then the output value is computed locally as $[\![z]\!] = [\![x + y]\!] = [\![x]\!] + [\![y]\!]$. If the gate is a multiplication gate, then the output value is computed by using Beavers technique [2] as follows. A fresh multiplication triple $[\![a]\!], [\![b]\!], [\![c]\!]$ is selected and the differences $[\![x - a]\!] = [\![x]\!] - [\![a]\!]$ and $[\![y - b]\!] = [\![y]\!] - [\![b]\!]$ are reconstructed. Then, the output value of the gate is computed locally as $[\![z]\!] = [\![x \cdot y]\!] = (x - a)(y - b) + (x - a)[\![b]\!] + (y - b)[\![a]\!] - [\![c]\!]$. In the end, once the output values of the circuit have been computed, they are reconstructed.[11]

Essentially the only thing corrupt players can do to disrupt the computation phase, is to provide incorrect shares when the players (try to) reconstruct a shared value $[\![w]\!]$. Since the latter is a linear combination of sharings in $\cup\mathcal{M}$ so that every $[w(d)]_d$ is a linear combination of the local base sharings $[s(d)]_d$, and because players are committed to their local base sharings (block-wise) by the authentication tags, players that hand in incorrect shares can be caught.

[10] As a consequence, the total number of fresh one-time keys $\boldsymbol{\mu}$ equals the total number of $\boldsymbol{\sigma}(d)$'s (over all d's), and thus sharing them (which will be needed) does not increase the overall asymptotic communication complexity.

[11] For simplicity we assume that all the players are supposed to learn all output values of the circuit. It is straightforward to adjust our scheme so that different players learn different output values.

2.7 Two New Essential Ingredients

We present here the two main new components that enable our improved communication complexity.

Batch-wise Multiplication Verification. Assume we have two sharings $[a]$ and $[b]$ (over $\mathbb{F}$), and the players have computed a sharing $[c]$, which is supposed to be $c = a \cdot b$, using an *optimistic* multiplication protocol (i.e., one that assumes that players behave). And now the players want to verify that indeed $c = a \cdot b$, without revealing anything beyond about a, b, c. The standard way of doing so (see e.g. [11] or [3]) has a failure probability of $1/|\mathbb{F}|$, which is too large for us, or when performed over the bigger field $\mathbb{K}$, has a sufficiently small failure probability of $1/|\mathbb{K}|$, but requires to share an element from $\mathbb{K}$ *for every triple* to be verified. This means we get a communication complexity of at least $O(n\kappa)$ bits per multiplication gate, whereas we want $O(n\phi + \kappa)$.

We achieve the latter by verifying $c = a \cdot b$ *batch-wise*. This is done by means of the following method. Let $([a^1], [b^1], [c^1]), \dots, ([a^N], [b^N], [c^N])$ be $N = n^2$ multiplication triples that need to be verified. Consider the degree-$(N-1)$ polynomials f and g with $f(x_k) = a^k$ and $g(x_k) = b^k$ for all $k \in \{1, \dots, N\}$, and set $a^k := f(x_k)$ and $b^k := g(x_k)$ for $k \in \{N+1, \dots, 2N-1\}$. The players can locally compute sharings $[a^k]$ and $[b^k]$ of these a^k and b^k with $k \in \{N+1, \dots, 2N-1\}$. Furthermore, by using the optimistic multiplication protocol, we let them compute $[c^{N+1}], \dots, [c^{2N-1}]$ where c^k is supposed to be $a^k \cdot b^k$. Let h be the degree-$(2N-2)$ polynomial with $h(x_k) = c^k$ for all $k \in \{1, \dots, 2N-1\}$. It now holds that all the multiplication triples are correct — i.e., that $c^k = a^k \cdot b^k$ for all $k \in \{1, \dots, 2N-1\}$ — if and only if $h = f \cdot g$ as polynomials. In order to test if $h = f \cdot g$ or not, the players can simply choose a random challenge $\sigma \in \mathbb{K}$ and see if $h(\sigma) = f(\sigma) \cdot g(\sigma)$ or not. For the latter, the players locally compute their shares of $[f(\sigma)]$, $[g(\sigma)]$ and $[h(\sigma)]$ — each is a linear combination of the shares of f, g, h that the player holds — and apply the "expensive" standard multiplication verification to $[f(\sigma)]$, $[g(\sigma)]$ and $[h(\sigma)]$.

Multiparty Computation of the Tags. As mentioned before, the tags τ should be computed in a multi-party fashion, without blowing up the asymptotic communication complexity. To simplify the exposition here, we assume for the moment that each tag τ is computed as $\tau = \mu \cdot \sigma_i(d) + \nu$ for $\mu \in \mathbb{K}$, and where $\sigma_i(d) \in \mathbb{K}$ is the i-th share of $[\sigma(d)]$. A first step in a multi-party computation usually is to share the inputs; here: μ, $\sigma_i(d)$ and ν. However, this blows up the communication complexity by a factor n, which we cannot afford. Note that sharing μ is actually okay, since the μ's are (partly) re-used, and thus we can also re-use their sharings. Also, sharing ν is okay, since in the actual authentication scheme we are using (not the simplified version we are discussing here), there is only one ν for many $\sigma_i(d)$'s. What *is* problematic, however, is the sharing of $\sigma_i(d)$. And this is where our second new method comes into play. We make use of the fact that $\sigma_i(d)$ is not an arbitrary input to the multi-party computation, but that it is actually a share of a shared secret $\sigma(d)$. Due to the symmetry of Shamir's secret sharing scheme, we may then view $\sigma_i(d)$ as the *secret* and the remaining shares $\sigma_j(d)$ as a sharing of $\sigma_i(d)$. Indeed, any $t+1$ of the shares $\sigma_j(d)$ can be used to recover $\sigma_i(d)$. Thus, in that sense, $\sigma_i(d)$ *is* already shared, and there is no need to share it once more.

Using this idea, the players can compute τ in a multi-party way as follows.[12] Player P_V, holding μ and ν, shares μ as a twisted degree-t sharing $\lceil \mu \rfloor^i_V$, and ν as a twisted degree-$2t$ sharing $\langle \nu \rangle^i_V$. The players now locally compute $\lceil \mu \rfloor^i_V \cdot [\sigma(d)] + \langle \nu \rangle^i_V$, which results in a twisted degree-$2t$ sharing $\langle \mu \cdot \sigma_i(d) + \nu \rangle^i$ of $\tau = \mu \cdot \sigma_i(d) + \nu$, as explained at the end of Section 2.5. These shares can now be sent to P_i for reconstruction (and correctness of τ will be verified by a cut-and-choose technique).

We point out that by corrupting t players P_j that do not include P_V or P_i, the adversary can learn μ from the (twisted) shares of the players in P_j. However, in that case, the adversary *cannot* anymore corrupt player P_i, and thus knowledge of μ is of no use. What is important is that the adversary does not learn μ in case it corrupts P_i, and this we will show to hold (in the final version of the scheme).

Adapting the above to $\tau = \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i(d) + \nu$, and re-using $\boldsymbol{\mu}$ and its twisted sharing, gives the players the means to compute their tags with a communication complexity that is negligible for large enough circuits.

2.8 A High Level Sketch of Our Construction

For the preparation phase, every player, acting as dealer P_d, produces many sharings $[s(d)]_d$. Correctness is verified batch-wise by means of a standard cut-and-choose technique. Every list of sharings $[s(1)]_1, \dots, [s(n)]_n$ then gives rise to $t+1$ two-level sharings $[\![a]\!]$ by setting $a = \sum_{d=1}^{n} s(d) x_j^d$ for $t+1$ different choices of j. This way,

[12] The actual scheme will be slightly more complicated due to some issue that we ignore right now for simplicity.

preparing *one* $[\![a]\!] \in \cup\mathcal{M}$ (and the same for $[\![b]\!] \in \cup\mathcal{M}$) amounts to preparing *one* $[s(d)]_d$ (up to constant factors), which has linear amortized complexity (meaning: a linear number of elements in $\mathbb{F}$). This technique is borrowed from [16]. Then, $[\![c]\!]$, where c is supposed to be $a \cdot b$, is computed by means of the passively-secure multiplication protocol due to [16], which has linear communication complexity. In order to verify the correctness of the c's, we use the *batch-wise multiplication verification* described in Section 2.7. Using batches of size $N = n^2$, verifying the correctness of N multiplication triples essentially boils down to reconstructing a *constant* number of sharings over the big field $\mathbb{K}$, which consists of every player sending his share (in $\mathbb{K}$) to every other player. Per multiplication triple, this then amounts to $O(\kappa)$ bits. Using batches of size $N = n^{2+const}$ reduces this to $O(\kappa/n^{const})$.

It remains to compute the authentication tags. As explained in Section 2.7, for a tag of the form $\tau = \mu \cdot \sigma_i(d) + \nu$ (where $\sigma_i(d)$ consists of a collection of $s_i(d)$'s), this can be done by computing $\langle\tau\rangle^i_V = \lceil\mu\rfloor^i_V \cdot [\sigma(d)] + \langle\nu\rangle^i_V$. Since the μ's (and their twisted shares) are re-used to some extent, and since the $\sigma(d)$'s are already shared, the communication complexity is dominated by communicating the shares $\langle\nu\rangle^i_V$ and $\langle\tau\rangle^i$; this consists of a linear number of elements in $\mathbb{K}$ per $\sigma(d)$, per verifier P_V, and per shareholder P_i. To do better, the tags are actually of the form $\tau = \boldsymbol{\mu} \cdot \boldsymbol{\sigma}_i(d) + \nu$, where $\boldsymbol{\sigma}_i(d)$ is a "large" vector. Hence, only *one* set of shares $\langle\nu\rangle^i_V$ (and $\langle\tau\rangle^i_V$) needs to be communicated (per P_V and P_i) to compute the tag τ for a "*large*" list of $s(d)$'s, making the overall communication complexity per $s(d)$, and thus per multiplication triple, negligible. The correctness of the tags is verified by a standard cut-and-choose technique. The details are worked out in Section 3.2.

Once the data structure as described in Section 2.6 is prepared, we are in good shape. Essentially, the only thing that can cause problems during the input and the computation phase is that corrupt players hand in incorrect shares; but this will be detected (since the shares then do not lie on a degree-t polynomial), and the corrupt players will be found with the help of the authentication tags (on the local base sharings). The details are explained in Sections 3.3 and 3.4.

3 Detailed Protocol Description

We now present the full protocol description. We stress that the main ingredients to the protocol are the new batch verification for multiplication triples, and to use a mini MPC for computing authentication tags, as discussed earlier in the paper. Fine-tuning everything makes the protocol very involved and probably hard to follow at first look. We suggest to skip at first reading the *fault localization* subprotocols. They are in fact rather straightforward (the players essentially open up everything in order to locate where things went wrong) but tedious, often with several case distinctions in order to branch through all possibilities, and sometimes not done in the most direct way in order to keep the communication low.

3.1 Two Basic Subprotocols

We introduce here two subprotocols that we will use later on several times.

Generating a Challenge. The purpose of the subprotocol Challenge below is to generate a common *challenge* $\lambda \in \mathbb{K}$ with high min-entropy. It makes use of an arbitrary injective mapping *convert* from $\{0,1\}^{2(\kappa+n)/n} \times \ldots \times \{0,1\}^{2(\kappa+n)/n}$ (n times) into $\mathbb{K}$.[13] Its communication complexity is $O(\kappa) \cdot \mathcal{BC}$.

Protocol Challenge

Every player $P_i \notin \mathcal{C}orr$ chooses and broadcasts a random string $str_i \in \{0,1\}^{2(\kappa+n)/n}$; for $P_i \in \mathcal{C}orr$, str_i is set to the all-0 string in $\{0,1\}^{2(\kappa+n)/n}$. The field element $\lambda = convert(str_1, \ldots, str_n) \in \mathbb{K}$ is taken as the generated challenge.

The following is easy to see.

Fact 1 *For any* fixed *set of t corrupt players*[14] *and for any given subset $\mathcal{S} \subset \mathbb{K}$, the probability that a challenge generated by the subprotocol* Challenge *lies in $\mathcal{S}$ is at most $|\mathcal{S}|/2^{2(t+1)(\kappa+n)/n} \leq |\mathcal{S}|/2^{\kappa+n}$.*

[13] Recall that n divides κ.

[14] Recall that in the end the adversary may corrupt the players *adaptively*.

Verified Sharing The purpose of the next subprotocol, $\mathsf{VerShare}_d$, is for player P_d (the dealer) to generate, for some $\ell' \in \mathbb{N}$ specified later, sharings $[s^1]_d, \dots, [s^{\ell'}]_d$ (with Kudzu shares) for $s^1, \dots, s^{\ell'} \in \mathbb{F}$ (randomly) chosen by him, and for the remaining players to verify the correctness of the sharings.

Protocol $\mathsf{VerShare}_d$

P_d chooses ℓ' random polynomials $f^1, \dots, f^{\ell'}$ over $\mathbb{F}$, and a random polynomial f^0 over $\mathbb{K}$, all of degree at most t, subject to that $f^k(x_j) = 0$ for every player $P_j \in \mathcal{D}isp_d$, and he sets $s^k = f^k(0)$ for every $k \in \{0, 1, \dots, \ell'\}$. Then, P_d computes and sends share $s_j^k = f^k(x_j)$ to player P_j, for every $k \in \{0, \dots, \ell'\}$ and for every player not in $\mathcal{D}isp_d$. These shares then form sharings $[s^0]_d, [s^1]_d, \dots, [s^{\ell'}]_d$ (with Kudzu shares) of which the first will be dismissed at the end.

Verification: By means of $\mathsf{Challenge}$, the players generate a challenge $\lambda \in \mathbb{K}$. Then, for every player P_V (acting as verifier), the following is done. Every player $P_j \notin \mathcal{D}isp_V$ sends his share $\sigma_i = \sum_{k=0}^{\ell'} \lambda^k s_j^k \in \mathbb{K}$ of $[\sigma] = \sum_{k=0}^{\ell'} \lambda^k [s^k]_d$ to P_V. If $\sigma_1, \dots, \sigma_n$ is a correct degree-t Shamir sharing (not counting the shares of players in $\mathcal{D}isp_V$) and $\sigma_j = 0$ for all $P_j \in \mathcal{D}isp_d$, then P_V broadcasts "`ok`" and protocol $\mathsf{VerShare}_d$ halts with $[s^1]_d, \dots, [s^{\ell'}]_d$ as output. Else, P_V broadcasts "`fault`", and *fault localization* is performed for the first P_V that complains.

Fault localization: The dealer P_d and the players $P_j \notin \mathcal{D}isp_V$ send all shares of $[s^0]_d, \dots, [s^{\ell'}]_d$ to P_V. This means, every share s_j^k is sent by both P_d and P_j to P_V. We distinguish between the following three cases (one of which must occur):

Case 1: One of the sharings $[s^k]_d$ sent by P_d is not a correct degree-t Shamir sharing. In this case, P_V broadcasts "`accuse` P_d", and the pair $\{P_d, P_V\}$ is added to the set of disputes $\mathcal{D}isp$.

Case 2: Some player P_j sent shares $s_j^0, \dots, s_j^\ell$ for which $\sigma_j \neq \sum_k \lambda^k s_j^k$, where σ_j is the value sent for the verification. In this case, P_V broadcasts "`accuse` P_j", and the pair $\{P_j, P_V\}$ is added to $\mathcal{D}isp$.

Case 3: P_d and P_j sent a different share s_j^k for some k. In this case, P_V broadcasts "`open` (k, j)" upon which both P_d and P_j must broadcast s_j^k. If they broadcast different values, then $\{P_d, P_j\}$ is added to $\mathcal{D}isp$. Otherwise, P_V checks who broadcasted a different value than he had sent to him, and then accuses that player as in case 1 or 2.

Fact 2 *If P_d remains honest then the adversary learns no information on $s^1, \dots, s^{\ell'}$. If the sharings $[s^1]_d, \dots, [s^{\ell'}]_d$ handed out by the dealer P_d are not all correct degree-t Shamir sharings, then a new dispute is found except with probability $2^n \ell'/2^{\kappa+n} = \ell'/2^\kappa$.*

The 2^n-factor stems from the adaptiveness of the adversary, i.e., that he may corrupt players *after* having seen the challenge λ; thus, we argue for every possible set of t corrupt players and apply union bound. The claim that a *new* dispute is found is somewhat tedious but straightforward to verify.

The communication complexity of $\mathsf{VerShare}_d$ is $O(\ell' n \phi + n^2 \kappa) + O(\kappa) \cdot \mathcal{BC}$.

It is easy to see that the protocol allows for the following variants (with the same asymptotic communication complexity): to generate simultaneously correct degree-t sharings $[s^1]_d, \dots, [s^{\ell'}]_d$ and degree-$2t$ sharings $\langle s^1 \rangle_d, \dots, \langle s^{\ell'} \rangle_d$ for the *same* $s^1, \dots, s^{\ell'} \in \mathbb{F}$ randomly chosen by P_d, or to generate correct twisted sharings $[s^1]_d^i, \dots, [s^{\ell'}]_d^i$, or to generate a correct twisted degree-$2t$ sharing $\langle 0 \rangle_d^i$ of 0, etc.

3.2 The Preparation Phase

The goal of the preparation phase is to prepare a data structure as discussed in Section 2.6. This is done by means of dividing the work into n^2 segments. In each segment $seg \in \{1, \dots, n^2\}$, a list $\mathcal{M}_{seg}$ of $m = M/n^2$ multiplication triples $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ is generated, and corresponding lists $\mathcal{S}_{seg}(d)$ of $\ell = L/n^2 = 3m/(t+1)$ local base sharings $[s(d)]_d$, with authenticated blocks $[\boldsymbol{\sigma}(d)]_d$. If a segment fails, then a new dispute is added to $\mathcal{D}isp$, the data of the segment is dismissed, and the players retry that segment. In the end, after at most $2n^2$ (possibly repeated) segments, the $\mathcal{M}_{seg}$ and $\mathcal{S}_{seg}(d)$'s (with the authenticated blocks) are combined to the full size data structure as described in Section 2.6.

In the remainder of this section, we describe how the data structure for a fixed segment seg is prepared. We take it as understood that as soon as a fault is detected, and as a consequence a new dispute is found (which may also mean that a player is identified to be corrupt), the execution of that segment is aborted and re-done.

Base Sharings. First, the players generate m two-level sharings $\llbracket a^1 \rrbracket, \dots, \llbracket a^m \rrbracket$ with Kudzu shares of random secret values $a^1, \dots, a^m \in \mathbb{F}$ with underlying *local base sharings*. This is done by means of the following procedure $\mathsf{Base\,Sharings}$. It makes use of the fact that the transpose of a Vandermonde matrix acts as a randomness extractor (see e.g. [16]).

Protocol $\mathsf{Base\,Sharings}$

Every player $P_d \notin \mathcal{C}orr$ performs $\mathsf{VerShare}_d$ to produce degree-t Shamir sharings $[s^1(d)]_d, \ldots, [s^{\ell/3}(d)]_d$ with Kudzu shares; for players $P_d \in \mathcal{C}orr$, $[s^1(d)]_d, \ldots, [s^{\ell/3}(d)]_d$ are all set to all-0 sharings. Then, for every $k \in \{1, \ldots, \ell/3\}$ and every $j \in \{0, \ldots, t\}$, the data structure $(x_1^j \cdot [s^k(1)]_1, x_2^j \cdot [s^k(2)]_2, \ldots, x_n^j \cdot [s^k(n)]_n)$ forms a two-level sharing $[\![\sum_i x_i^j s^k(i)]\!]$. The resulting $m = (t+1)\ell/3$ two-level sharings are set to be $[\![a^1]\!], \ldots, [\![a^m]\!]$.

Fact 3 *Except with probability at most $mn/2^\kappa$, the following holds: protocol* $\mathsf{Base\,Sharings}$ *succeeds and the uncorrupt players hold correct subshares of two-level sharings $[\![a^1]\!], \ldots, [\![a^m]\!]$, or it fails and a new dispute is found. Furthermore, if it succeeds, then $[a^1], \ldots, [a^m]$ are random and independent sharings of random and independent values $a^1, \ldots, a^m \in \mathbb{F}$.*

We point out that the second-level shares are *not* independent. As a consequence, we have to be careful with revealing those.

Similarly, the players produce m (verified) two-level sharings $[\![b^1]\!], \ldots, [\![b^m]\!]$ with Kudzu shares with underlying local base sharings $[s^{\ell/3+1}(d)]_d, \ldots, [s^{2\ell/3}(d)]_d$ are produced. Finally, using a straightforward variant of the above procedure, the players produce m (verified) two-level sharings $[\![r^1]\!], \ldots, [\![r^m]\!]$ and $\langle\!\langle r^1 \rangle\!\rangle, \ldots, \langle\!\langle r^m \rangle\!\rangle$ with Kudzu shares with respective underlying local base sharings $[s^{2\ell/3+1}(d)]_d, \ldots, [s^{\ell}(d)]_d$ and $\langle s^{2\ell/3'+1}(d)\rangle_d, \ldots, \langle s^{\ell}(d)\rangle_d$. $\mathcal{M}'_{seg}$ is set to be the list of m quadruples $([\![a^k]\!], [\![b^k]\!], [\![r^k]\!], \langle\!\langle r^k \rangle\!\rangle)$, and, for every d, $\mathcal{S}_{seg}(d)$ is set to be the list $[s^1(d)]_d, \ldots, [s^{\ell}(d)]_d$.

The communication complexity of preparing $\mathcal{M}'_{seg}$ (or failing but finding a new dispute) is $O(\ell n^2 \phi + n^3 \kappa) = O(mn\phi + n^3\kappa)$ bits plus $n \cdot O(\kappa) \cdot \mathcal{BC}$.[15]

Multiplication. Every quadruple $([\![a]\!], [\![b]\!], [\![r]\!], \langle\!\langle r \rangle\!\rangle) \in \mathcal{M}'_{seg}$ is extended to $([\![a]\!], [\![b]\!], [\![c]\!], [\![r]\!], \langle\!\langle r \rangle\!\rangle)$, where c is supposed to be $c = a \cdot b$, and we call the new (extended) list $\mathcal{M}''_{seg}$. This is done by means of the following procedure Mult, which is due to Damgård and Nielsen [16], and which has a communication complexity of $O(n\phi)$ bits per quadruple.

Protocol Mult

Let P_K (the "king") be the first player that is not in $\mathcal{C}orr$. Every player P_j sends his share of $\langle d \rangle = [a][b] + \langle r \rangle$ to P_K; if $P_j \in \mathcal{D}isp_K$, then P_j sends his share via a *relay*, i.e., via the first player that is neither in $\mathcal{D}isp_j$ nor in $\mathcal{D}isp_K$. P_K then reconstructs d by computing the unique degree-$2t$ polynomial defined by the n shares, and sends d to every player, via a relay for players in $\mathcal{D}isp_K$. $[\![c]\!]$ is then computed (by means of local computations) as $[\![c]\!] = d - [\![r]\!]$.

Fact 4 *The adversary learns no information on a and b from executing* Mult. *Furthermore, in case of no adversarial behavior, the players hold correct two-level shares of $c = a \cdot b$.*

In the end, $\mathcal{M}_{seg}$ is obtained by removing the $[\![r]\!]$ and $\langle\!\langle r \rangle\!\rangle$-components from the entries of $\mathcal{M}''_{seg}$, but beforehand, the players need to verify the correctness of the $[\![c]\!]$'s: that the sharings are correct, and that indeed $c = a \cdot b$. This is done as outlined below.

Auxiliary Structures. In order to be able to verify the multiplication triples, i.e., that indeed $c = a \cdot b$, the players need to produce some additional auxiliary data structures. One is an additional fresh list $\mathcal{R}^\circ_{seg}$ of pairs of verified two-level sharings $([\![r^1_\circ]\!], \langle\!\langle r^1_\circ \rangle\!\rangle), \ldots, ([\![r^m_\circ]\!], \langle\!\langle r^m_\circ \rangle\!\rangle)$ with Kudzu shares of random $r^1_\circ, \ldots, r^m_\circ \in \mathbb{F}$; this has clearly no effect on the asymptotic overall complexity. The other is a list $\mathcal{M}^\circ_{seg}$ of m/n^2 *dummy* multiplication triples $[\![\alpha_\circ]\!], [\![\beta_\circ]\!], [\![\gamma_\circ]\!]$ with the corresponding $[\![\rho_\circ]\!]$ and $\langle\!\langle \rho_\circ \rangle\!\rangle$-components, but over $\mathbb{K}$, i.e., $\alpha_\circ, \beta_\circ, \gamma_\circ, \rho_\circ \in \mathbb{K}$. This can be done by similar means as $\mathcal{M}''_{seg}$ is prepared, but using the field $\mathbb{K}$ instead of $\mathbb{F}$. The communication complexity of preparing $\mathcal{M}^\circ_{seg}$ is $O(m\kappa/n + n^3\kappa)$ bits plus $O(n\kappa)$ broadcasts.

[15] The broadcast complexity could be reduced to $O(\kappa)$ by re-using the challenge in $\mathsf{VerShare}$ for the different dealers. However, since the broadcast complexity will anyway be dominated by some other part of the protocol, this does not affect the overall asymptotic complexity.

Verifying the Multiplication. The standard procedure to verify the correctness of multiplication triples, as for instance used in [10, 11, 3] (see also Single Verify below), which verifies triples on a one-by-one basis, is too expensive for us. We verify the correctness of the multiplication triples in *batches* of size $N = n^2$ (or $N = n^{2+const}$ for a general *const*).

Protocol Verify

The players execute the following *batch verify* procedure in parallel for every (disjoint) batch $([\![a^1]\!], [\![b^1]\!], [\![c^1]\!], [\![r^1]\!], \langle\!\langle r^1\rangle\!\rangle), \dots, ([\![a^N]\!], [\![b^N]\!], [\![c^N]\!], [\![r^N]\!], \langle\!\langle r^N\rangle\!\rangle) \in \mathcal{M}''_{seg}$ of size $N = n^2$, using the same challenge σ in all the parallel executions, and the same challenge λ in all the parallel sub-calls to Single Verify.

Batch verify: For the considered batch $([\![a^1]\!], [\![b^1]\!]), \dots, ([\![a^N]\!], [\![b^N]\!])$, define degree-$(N-1)$ polynomials f and g with $f(x_k) = a^k$ and $g(x_k) = b^k$ for all $k \in \{1, \dots, N\}$. The players locally compute $[\![a^k]\!]$ and $[\![b^k]\!]$ with $f(x_k) = a^k$ and $g(x_k) = b^k$ for all $k \in \{N+1, \dots, 2N-1\}$. Furthermore, by using $N-1$ pairs $([\![r^{N+1}]\!], \langle\!\langle r^{N+1}\rangle\!\rangle), \dots, ([\![r^{2N-1}]\!], \langle\!\langle r^{2N-1}\rangle\!\rangle)$ from $\mathcal{R}^{\circ}_{seg}$ (which are then removed from $\mathcal{R}^{\circ}_{seg}$) and invoking Mult, the players compute $[\![c^{N+1}]\!], \dots, [\![c^{2N-1}]\!]$ where c^k is supposed to be $a^k \cdot b^k$. Let h be the degree-$(2\ell - 2)$ polynomial with $h(x_k) = c^k$ for all $k \in \{1, \dots, 2N-1\}$. The players generate a challenge $\sigma \in \mathbb{K}$ (one for all the calls to *batch verify*) by means of Challenge and compute $[\![\alpha]\!]$, $[\![\beta]\!]$ and $[\![\gamma]\!]$ for $\alpha = f(\sigma)$, $\beta = g(\sigma)$ and $\gamma = h(\sigma)$ in $\mathbb{K}$; this can be done by local computations since $f(\sigma)$, $g(\sigma)$ and $h(\sigma)$ are linear combinations of $a^1, \dots, a^N$, $b^1, \dots, b^N$ and $c^1, \dots, c^{2N-1}$, respectively (with coefficients in $\mathbb{K}$). Finally, they perform Single Verify to verify that $\alpha \cdot \beta = \gamma$.

Then, every player P_V broadcasts "ok" if he accepted all calls to Single Verify, else he broadcasts the number of the batch for which Single Verify failed for him. For the first batch $([\![a^1]\!], [\![b^1]\!]), \dots, ([\![a^N]\!], [\![b^N]\!])$ for which there was a complaint, and for the smallest player P_V that complained about that batch, *fault localization* of Single Verify is performed. Note that the value δ and the sharings $[\![\rho]\!]$ and $\langle\!\langle \rho\rangle\!\rangle$ (such that γ is supposed to be $\delta - [\![\rho]\!]$ with $\langle \delta\rangle = [\alpha][\beta] + \langle \rho\rangle$), required by the *fault localization* of Single Verify, can be computed as a linear combination of the d^k's and the $[\![r^k]\!]$ and $\langle\!\langle r^k\rangle\!\rangle$'s, respectively.

We observe that if $a^k \cdot b^k \neq c^k$ for some $k \in \{1, \dots, N\}$, then $f \cdot g \neq h$ as polynomials (of degree at most $2(N-1)$), and thus there are at most $2N-1$ values $\sigma \in \mathbb{K}$ with $f(\sigma) \cdot g(\sigma) = h(\sigma)$. Thus, from Fact 1, and by using union bound over all possible sets of corrupted players, we obtain the following.

Fact 5 *For every batch $([\![a^1]\!], [\![b^1]\!]), \dots, ([\![a^N]\!], [\![b^N]\!])$, if $[\![c^k]\!]$ is not a correct sharing or $a^k \cdot b^k \neq c^k$ for some $k \in \{1, \dots, N\}$, then the probability that $[\![\gamma]\!]$ is a correct sharing and $\alpha \cdot \beta = \gamma$ is at most $2(N-1)/2^{\kappa}$.*

It remains to show how the players verify the correctness of the multiplication triples $([\![\alpha]\!], [\![\beta]\!], [\![\gamma]\!])$. This is done by means of the following standard protocol.

Protocol Single Verify

The players choose a fresh dummy multiplication triple $([\![\alpha_\circ]\!], [\![\beta_\circ]\!], [\![\gamma_\circ]\!], [\![\rho_\circ]\!], \langle\!\langle \rho_\circ\rangle\!\rangle)$ from $\mathcal{M}^{\circ}_{seg}$ and remove it from the list. Then, the players invoke Challenge to generate a challenge (one for all the parallel calls to Single Verify) $\lambda \in \mathbb{K}$, and $[\alpha'] = [\alpha_\circ] + \lambda[\alpha]$ and $[o] = [\gamma_\circ] + \lambda[\gamma] - \alpha'[\beta]$ are sequentially opened. The latter is done by exchanging all shares of $[\alpha']$ (and of $[o]$), i.e., every player P_j sends his share α'_j of α' (and similar for o) to every $P_V \notin \mathcal{D}isp_j$. If one of the players P_V receives inconsistent shares or reconstructs $o \neq 0$, then he rejects, otherwise he accepts.

Fault localization: The players broadcast their shares of $[\alpha], [\beta], [\rho]$ and $\langle \rho\rangle$, and of $[\alpha_\circ], [\rho_\circ]$ and $\langle \rho_\circ\rangle$. Additionally, they broadcast the values δ and $\delta_\circ$.

Case 1: If the value δ broadcasted by some player P_i is different to the one broadcasted by P_K (the "king" in Mult), or to the one broadcasted by his relay P_r in case $P_j \in \mathcal{D}isp_K$, then $\{P_i, P_K\}$, or $\{P_i, P_r\}$ respectively, is added to $\mathcal{D}isp$ and Single Verify halts. Similarly in case there is an inconsistency for $\delta_\circ$.

Case 2: If some sharing, say $[\alpha]$ for concreteness, is inconsistent, then every player $P_j \notin \mathcal{D}isp_V$ send his subshares $\alpha_j(1), \dots, \alpha_j(n)$ of $[\![\alpha]\!]$ to P_V. If the subshares of a player P_j are inconsistent with the share he broadcasted earlier, then P_V broadcasts "accuse P_j", and $\{P_V, P_j\}$ is added to $\mathcal{D}isp$ and Single Verify halts. Else, there exists $P_d \notin \mathcal{C}orr$ so that $\alpha_1(d), \dots, \alpha_n(d)$ (not considering the shares of players $P_j \in \mathcal{D}isp_V$) do not form a correct degree-t sharing, and player P_V broadcasts "open d", upon which the players $P_j \notin \mathcal{D}isp_V \cup \mathcal{D}isp_d$ have to broadcast $\alpha_j(d)$. If a player P_j broadcasts a different value than he had sent to P_V before, then P_V broadcasts "accuse P_j", and $\{P_V, P_j\}$ is added to $\mathcal{D}isp$ and Single Verify halts. Otherwise, P_d broadcasts "accuse P_j", where P_j is a player that broadcasted an incorrect share, and $\{P_d, P_j\}$ is added to $\mathcal{D}isp$ and Single Verify halts.

Case 3: If $\delta \neq \alpha \cdot \beta + \rho$ or $\delta_\circ \neq \alpha_\circ \cdot \beta_\circ + \rho_\circ$ then P_K broadcasts "accuse P_j", where P_j is a player who has broadcasted shares that are inconsistent with what he had sent him during Mult, and $\{P_K, P_j\}$ is added to $\mathcal{D}isp$ and Single Verify halts.

Fact 6 $\mathsf{Single\,Verify}$ *reveals no information on α, β and γ beyond the multiplicative relation $\gamma = \alpha \cdot \beta$.*

The communication complexity of one call to *batch verify* of Verify (not counting the calls to $\mathsf{Challenge}$ and $\mathsf{Single\,Verify}$) is $O(Nn\phi)$ bits. The communication complexity of one call to $\mathsf{Single\,Verify}$, not counting the sub-call to $\mathsf{Challenge}$ and *fault localization*, is $O(n^2\kappa)$ bits. The communication complexity of *fault localization* is $O(n^2\kappa) + O(n\kappa) \cdot \mathcal{BC}$. It follows that the communication complexity of Verify (including the calls to $\mathsf{Single\,Verify}$) amounts to $m/N \cdot O(Nn\phi + n^2\kappa) = O(m(n\phi+\kappa))$ bits plus $O(n\kappa) \cdot \mathcal{BC}$.

Fact 7 *The probability that all uncorrupt players $P_V \notin \mathcal{D}isp_K$ accept* $\mathsf{Single\,Verify}$ *yet either $[\![\gamma]\!]$ is not a correct sharing or $\gamma \neq \alpha \cdot \beta$ (or both) is at most $2^{-\kappa}$. If a player $P_V \notin \mathcal{D}isp_K$ rejects, and fault localization is performed, then a new dispute is found.*

If for all the batches $\mathsf{Batch\,Verify}$ succeeds, then the players remove the $[\![r]\!]$ and $\langle\!\langle r \rangle\!\rangle$-components from the entries of $\mathcal{M}''_{seg}$ to obtain $\mathcal{M}_{seg}$.

Computing the Tags. Finally, we need to "commit" the players to their shares of the local base sharings $[s(d)] \in \mathcal{S}_{seg}(d)$ (for every P_d) by giving them authentication tags. Specifically, the following needs to be achieved. For every player P_V (acting as verifier) and every player P_i (the player that needs to be committed to his shares), player P_V should obtain a random long-term key $\boldsymbol{\mu} \in \mathbb{K}^{3m'/ne}$, and for every player P_d and every block $[\boldsymbol{\sigma}(d)]_d \in \mathcal{S}_{seg}(d)$ (where $\boldsymbol{\sigma}(d) \in \mathbb{K}^q$), player P_V should additionally obtain a random one-time key $\nu \in \mathbb{K}$ and player P_i should be given the tag

$$\tau = \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i(d) + \nu\,.$$

Recall that $\boldsymbol{\sigma}(d)$ is obtained as follows. We parse the ℓ sharings $[s(d)]_d \in \mathcal{S}_{seg}(d)$ over $\mathbb{F}$ as ℓ/e sharings $[\sigma^1(d)]_d, \ldots, [\sigma^{\ell/e}(d)]_d$ over $\mathbb{K}$, and collect the $[\sigma^k(d)]$'s into n blocks $[\boldsymbol{\sigma}^1(d)], \ldots, [\boldsymbol{\sigma}^n(d)]$ with $\boldsymbol{\sigma}^1(d), \ldots, \boldsymbol{\sigma}^n(d) \in \mathbb{K}^q$ with $q = \ell/(ne)$.

We stress that per segment, every P_V has one *fixed* long-term key $\boldsymbol{\mu}$ per player P_i (and uses that very same key for the different players P_d and the different blocks $[\boldsymbol{\sigma}(d)]_d \in \mathcal{S}_{seg}(d)$). The short-term keys and the tags, on the other hand, are fresh for every P_V, P_d, P_i and block $[\boldsymbol{\sigma}(d)]_d$.

What makes the computation of these tags non-trivial, is the fact that no-one beyond P_V should learn $\boldsymbol{\mu}$ and ν, and no-one beyond P_i and P_d knows (and may know) $\boldsymbol{\sigma}_i(d)$. This means, no single player can perform the computation, but τ needs to be computed jointly by the players in a multi-party fashion.

For the computation (or, actually, the verification) of the tags, the players will need, for every P_d, a random *dummy* sharing $[\boldsymbol{\sigma}^\circ(d)]_d$. Preparing these does not increase the asymptotic overall communication complexity.

Protocol $\mathsf{TagComp}$

For every player P_V and P_i with $\{P_V, P_i\} \notin \mathcal{D}isp$, the following is performed.

Key generation: By means of (a straightforward variant of) $\mathsf{VerShare}_V$, player P_V generates q verified twisted sharings $\lceil \mu^1 \rfloor^i_V, \ldots, \lceil \mu^q \rfloor^i_V$ with Kudzu shares of randomly chosen $\boldsymbol{\mu} = (\mu^1, \ldots, \mu^q) \in \mathbb{K}^q$.

Then, for every P_V, P_i, P_d with $P_i \notin \mathcal{D}isp_V \cup \mathcal{D}isp_d$, and for every of the $(n+1)$ blocks $[\boldsymbol{\sigma}^k(d)]_d \in \mathcal{S}_{seg}(d) \cup \{[\boldsymbol{\sigma}^\circ(d)]_d\}$, the players perform

Tag computation: The players execute $\mathsf{SingleTagComp}_{V,i,d}$ (given below); as a result, P_V obtains ν^k and P_i obtains τ^k (supposed to be $\tau^k = \boldsymbol{\mu} \odot \boldsymbol{\sigma}^k_i(d) + \nu^k$).

Then, the players produce a challenge $\lambda \in \mathbb{K}$ by means of $\mathsf{Challenge}$ and perform *batch verification* for every P_V, P_i, P_d with $P_i \notin \mathcal{D}isp_V \cup \mathcal{D}isp_d$.

Batch verification: P_i computes and sends $\boldsymbol{\sigma}'_i(d) = \sum_{k=1}^n \lambda^k \boldsymbol{\sigma}^k_i(d) + \boldsymbol{\sigma}^\circ_i(d)$ and $\tau' = \sum_{k=1}^n \lambda^k \tau^k + \tau^\circ$ to P_V. P_V accepts if $\tau' = \boldsymbol{\mu} \odot \boldsymbol{\sigma}'_i(d) + \nu'$ where $\nu' = \sum_{k=1}^n \lambda^k \nu^k + \nu^\circ$, else, P_V rejects.

For every P_V, if he accepted *batch verification* for every $P_i \notin \mathcal{D}isp_V$ then he broadcasts "`ok`", otherwise he broadcasts "`fault`" together the smallest P_i that he did not accept. For the smallest P_V that did not broadcast "`ok`" and the corresponding P_i, *fault localization* is performed.

Fault localization: P_i sends $\tau^1, \ldots, \tau^n$ and $\boldsymbol{\sigma}^1_i(d), \ldots, \boldsymbol{\sigma}^n_i(d)$ to P_V, and P_V computes $\tau^\circ = \tau' - \sum_{k=1}^n \lambda^k \tau^k$ and $\boldsymbol{\sigma}^\circ_i(d) = \boldsymbol{\sigma}'_i(d) - \sum_{k=1}^n \lambda^k \boldsymbol{\sigma}^k_i(d)$. Then, P_V finds and broadcasts the smallest index $k \in \{\circ, 1, \ldots, n\}$ for which $\tau^k \neq \boldsymbol{\mu} \odot \boldsymbol{\sigma}^k_i(d) + \nu^k$, and then *fault localization for a single tag* of $\mathsf{SingleTagComp}_{V,i,d}$ is invoked.

Fact 8 *If $\tau^k \neq \boldsymbol{\mu}\odot\boldsymbol{\sigma}_i^k(d)+\nu^k$ for some player P_d, some block $[\boldsymbol{\sigma}^k(d)]$, and some uncorrupt players P_i and P_V with $P_i \notin \mathcal{D}isp_d$, then, except with probability $n/2^\kappa$, the following holds: P_V broadcasts "*`fault`*", or P_i or P_V becomes corrupted. If some P_V broadcasts "*`fault`*", then fault localization for a single tag of* $\mathsf{SingleTagComp}_{V,i,d}$ *will be invoked for some player P_d, some block $[\boldsymbol{\sigma}^k(d)]$, and some player $P_i \notin \mathcal{D}isp_d\cup\mathcal{D}isp_V$ for which $\tau^k \neq \boldsymbol{\mu}\odot\boldsymbol{\sigma}_i^k(d)+\nu^k$ or for which P_V or P_i is corrupt.*

The communication complexity to run *key generation* for every P_V and P_i is $n^2\cdot O(qn\kappa+n^2\kappa) = O(mn\phi+n^4\kappa)$ bits plus $O(n^2\kappa)\cdot\mathcal{BC}$. By doing the $O(n^2)$ executions of *key generations* in parallel, and using the *same* challenge in all instances of $\mathsf{VerShare}$, permits to reduce the broadcast complexity at least to $O(n\kappa)\cdot\mathcal{BC}$. The communication complexity of *batch verification* for every i, d and V, including *fault localization* (which is executed at most once), but not counting the call to $\mathsf{SingleTagComp}$, is $O(n^3q\kappa) = O(mn\phi)$ bits plus $O(n)\cdot\mathcal{BC}$.

It remains to show how the tags are jointly computed by the players. Our description below is for a fixed choice of P_V, P_i, P_d and block $[\boldsymbol{\sigma}(d)]_d = [\boldsymbol{\sigma}^k(d)]_d \in \mathcal{S}_{seg}(d)$. For simplicity, we omit the argument d and write $[\boldsymbol{\sigma}]_d$ instead of $[\boldsymbol{\sigma}(d)]_d$, and we write $(\sigma^1,\dots,\sigma^q)$ for the coordinates of $\boldsymbol{\sigma}$ and $(\sigma_i^1,\dots,\sigma_i^q)$ for the coordinates of $\boldsymbol{\sigma}_i$.

The one-time key ν and the tag τ are chosen/computed by means of the following subprotocol, unless P_i is in dispute with P_d or with P_V. In the former case, his shares are fixed to 0 anyway, and μ and τ are simply both set to 0, and in the latter, P_i and P_V accuse each other anyway.

Protocol $\mathsf{SingleTagComp}_{V,i,d}$

Player P_V chooses a random $\nu\in\mathbb{K}$ and shares it (non-verifiably) as $\langle\nu\rangle_V^i$ with Kudzu shares. Similarly, player P_d shares $o = 0$ (i.e., zero) over $\mathbb{K}$ as $\langle o\rangle_V^i$ with Kudzu shares. The players locally compute $\langle\tau\rangle^i = \sum_{k=1}^q[\sigma^k]_d\lceil\mu^k\rfloor_V^i + \langle\nu\rangle_V^i + \langle o\rangle_d^i$ and send their shares to P_i. If $P_j \in \mathcal{D}isp_i$ then P_j sends his share of $\langle\tau\rangle^i$ to P_i via a *relay*, i.e., via the first player that is not in dispute with both P_i and P_j; for any player $P_j\in\mathcal{C}orr$, P_i takes 0 as this player's share. P_i can now compute the unique degree-$2t$ polynomial that fits these shares and obtains τ as the evaluation at x_i.

It is easy to verify that if all players follow the protocol, then P_i obtains $\tau = \boldsymbol{\mu}\odot\boldsymbol{\sigma}_i+\nu$ (where $\boldsymbol{\sigma}_i$ is determined by $[\boldsymbol{\sigma}]_d$ and ν by $\langle\nu\rangle_V^i$). The communication complexity to run $\mathsf{SingleTagComp}_{V,i,d}$ for every P_V, P_i, P_d and for every of the n blocks $[\boldsymbol{\sigma}] = [\boldsymbol{\sigma}(d)]\in\mathcal{S}_{seg}(d)$ is $O(n^5\kappa)$ bits.

Proposition 1 (Privacy of the keys). *If P_V remains honest and the adversary corrupts at most $t-1$ players different to P_i (which is e.g. satisfied if he eventually corrupts P_i), then the adversary learns no information on $\boldsymbol{\mu} = (\mu^1,\dots,\mu^q)$ and ν, beyond $\tau = \sum_k \sigma_i^k\mu^k+\nu$ (for the* correct *shares σ_i^k, defined by the shares of the uncorrupt players).*

By the security of the underlying authentication scheme, this guarantees that if at some later point player P_i lies about his shares, then he will be caught by P_V except with probability $1/|\mathbb{K}|$. Interestingly, if the adversary corrupts t players not including P_i (nor P_V) then he actually learns player P_V's long-term key $\boldsymbol{\mu}$ (that P_V uses to verify P_i's shares); however, in this case, P_i is guaranteed to remain honest and provide correct shares. So, this does not help the adversary.

Proof. It is sufficient to prove the claim in case of a corrupt dealer P_d and a corrupt player P_i, and thus we may assume that the adversary learns the shares of $\langle\tau\rangle^i = \sum_k[\sigma^k]\lceil\mu^k\rfloor_V^i + \langle\nu\rangle_V^i$, i.e., we may assume that all the shares of o are 0. We understand $[\sigma^k]$ as the *correct* sharing of some σ^k, determined by the shares of the uncorrupt players. As such, the data structure $\langle\tau\rangle^i = \sum_k[\sigma^k]\lceil\mu^k\rfloor_V^i + \langle\nu\rangle_V^i$, and in particular τ, is well defined, even though the corrupt players may perform additional computations on their shares of μ^k and ν. First note that (by assumption) there are at most $t-1$ corrupt players P_j that hold a (twisted) share of μ^k; thus, the $\lceil\mu^k\rfloor_V^i$'s give away no information on the μ^k's to the adversary. However, this is not sufficient to argue privacy, since the adversary also learns all shares of $\langle\tau\rangle^i = \sum_k[\sigma^k]\lceil\mu^k\rfloor_V^i + \langle\nu\rangle_V^i$, which potentially may leak additional information on the μ^k's and on ν (beyond τ). To argue privacy, consider a twisted sharing $\lceil\delta^1\rfloor_V^i$ of an arbitrary $\delta^1\in\mathbb{K}$, but with the additional property that the shares of all corrupt players are 0. Thus, the adversary cannot distinguish the sharing $\lceil\mu^1\rfloor_V^i$ from $\lceil\tilde{\mu}^1\rfloor_V^i = \lceil\mu^1+\delta^1\rfloor_V^i = \lceil\mu^1\rfloor_V^i + \lceil\delta^1\rfloor_V^i$. Furthermore, the adversary cannot distinguish the sharing $\langle\nu\rangle^i_V$ from $\langle\tilde{\nu}\rangle^i_V = \langle\nu - \sigma^1\delta^1\rangle^i_V = \langle\nu\rangle^i_V - [\sigma^1]_d\lceil\delta^1\rfloor^i_V$. But now, since

$$[\sigma^1]_d\lceil\tilde{\mu}^1\rfloor^i_V + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor^i_V + \langle\tilde{\nu}\rangle^i_V$$
$$= [\sigma^1]_d\lceil\mu^1\rfloor + [\sigma^1]_d\lceil\delta^1\rfloor^i_V + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor^i_V + \langle\nu\rangle^i_V - [\sigma^1]_d\lceil\delta^1\rfloor^i_V = \langle\tau\rangle^i$$

it holds that the adversary has no information on whether μ^1 and ν had been shared (even when given the remaining μ^k's), or $\tilde{\mu}^1$ and $\tilde{\nu}$. This means that every pair (μ^1, ν) with $\sum_k \sigma^k_i\mu^k + \nu = \tau$ is equally likely for the adversary, and similarly one can argue for the other μ^k's. □

Proposition 2 (Privacy of the shares). *If P_d remains honest, then the adversary learns no information on $\boldsymbol{\sigma} = (\sigma^1, \ldots, \sigma^q)$.*

Proof. The proof goes similar to the proof of Proposition 1. It is sufficient to prove the claim in case both P_i and P_V become corrupt, and thus we may assume that P_i learns the shares of $\langle\tau\rangle^i = \sum_k[\sigma^k]\lceil\mu^k\rfloor^i_V + \langle o\rangle^i_d$. Consider a twisted sharing $[\delta^1]_d$ of an arbitrary δ^1, but with the additional property that the shares of all corrupt players are 0. Thus, the adversary cannot distinguish the sharing $[\sigma^1]_d$ from $[\tilde{\sigma}^1]_d = [\sigma^1 + \delta^1]_d = [\sigma^1]_d + [\delta^1]_d$. Furthermore, the adversary cannot distinguish the sharing $\langle o\rangle^i_d$ from $\langle\tilde{o}\rangle^i_d = \langle o\rangle^i_d - [\delta^1]^i_d\lceil\mu^1\rfloor^i_V$, as both are (twisted) degree-$2t$ sharings of 0 with identical shares for the corrupt players (here we are using that the shares of $[\mu^1]^i_V$ correctly lie on a degree-t polynomial that vanishes at x_i). But now, since

$$[\tilde{\sigma}^1]_d\lceil\mu^1\rfloor^i_V + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor^i_V + \langle\tilde{o}\rangle^i_d$$
$$= [\sigma^1]_d\lceil\mu^1\rfloor + [\delta^1]_d\lceil\mu^1\rfloor^i_V + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor^i_V + \langle o\rangle^i_d - [\delta^1]^i_d\lceil\mu^1\rfloor^i_V = \langle\tau\rangle^i$$

it holds that the adversary has no information on whether σ^1 had been shared (even when given the remaining σ^k's), or $\tilde{\sigma}^1$. This means that every pair σ^1 is equally likely for the adversary, and similarly one can argue for the other σ^k's. □

Note that the *correctness* of τ is verified within the *batch verification* of TagComp. In case a tag is detected to be incorrect, the following *fault localization for a single tag* is performed.

Protocol SingleTagComp$_{V,i,d}$

Fault localization for a single tag: The shares of $\lceil\mu^1\rfloor^i_V, \ldots, \lceil\mu^q\rfloor^i_V$, $\langle\nu\rangle^i_V$ and $\langle o\rangle^i_d$ are sent to P_i, for each share by all players that know it, e.g., the players (via relays for players $P_j \in \mathcal{D}isp_i$) and P_V for the shares of $\lceil\mu^k\rfloor^i_V$ and P_d for the shares of $\langle o\rangle^i_d$.

Case 1: If P_V sent an incorrect sharing $\lceil\mu^k\rfloor^i_V$, then P_i broadcasts "`accuse` P_V", and $\{P_i, P_V\}$ is added to $\mathcal{D}isp$. Similarly, if $\langle o\rangle^i_d$ sent by P_d is not a correct sharing of 0 then P_i broadcasts "`accuse` P_d", and $\{P_i, P_d\}$ is added to $\mathcal{D}isp$.

Case 2: If two players that do not involve a relay sent different values for the same share (e.g. P_j sent a different value for μ^k_j than P_V), then P_i broadcasts an "`open`" command upon which the two players have to broadcast this value. If the two players broadcast different values, then these two players are added to $\mathcal{D}isp$. Else, P_i accuses the player that broadcasted a different value than he had sent earlier, and this player is added to $\mathcal{D}isp$ along with P_i.

Case 3: If two players, one via a relay, sent different values for the same share (e.g. P_j sent a different value for μ^k_j than P_V), then P_i broadcasts an "`open`" command upon which the two players and the *relay* have to broadcast this value. If two of these three players broadcast different values, then these two players are added to $\mathcal{D}isp$. Else, P_i accuses the player not in $\mathcal{D}isp_i$ that broadcasted a different value than he had sent earlier, and this player is added to $\mathcal{D}isp$ along with P_i.

Case 4: If $\tau_j \neq \sum_k \sigma^k_j\mu^k_j + \nu_j + o_j$ for some player P_j then the following is done. If $P_j \notin \mathcal{D}isp_i$ then P_i broadcasts "`accuse` P_j", and $\{P_i, P_j\}$ is added to $\mathcal{D}isp$. If $P_j \in \mathcal{D}isp_i$ then P_i broadcasts a request to have the *relay* resolve the issue, and the *relay* then broadcasts "`accuse` P_j" if $\tau_j \neq \sum_k \sigma^k_j\mu^k_j + \nu_j + o_j$ for the values received from P_j, and else "`accuse` P_V". The *relay* together with the accused player is then added to $\mathcal{D}isp$.

Case 5: If none of the above applies, P_i concludes that τ is actually correct and broadcasts "`accuse` P_V", and $\{P_i, P_V\}$ is added to $\mathcal{D}isp$.

Fact 9 *If $\tau \neq \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i + \nu^k$, or P_V or P_i is corrupt, and $P_i \notin \mathcal{D}isp_d \cup \mathcal{D}isp_V$, then fault localization for a single tag of* $\mathsf{TagComp}_{V,i,d}$ *finds a new dispute.*

The communication complexity of the above *fault localization*, which is invoked at most once (per segment), is $O(qn\kappa) = O(\ell\phi) = O(m\phi/n)$ bits plus $O(\kappa) \cdot \mathcal{BC}$. Adding up, the communication complexity of performing TagComp is $O(mn\phi + n^5\kappa) + O(n\kappa) \cdot \mathcal{BC}$.

One More Auxiliary Structure. In order to deal with the dependencies of the second-level shares of the two-level sharings $[\![w]\!] \in \cup\mathcal{M}$, the players also produce t random and *fully independent* two-level sharings of zero in the preparation phase. This is done by means of (a straightforward modification of) $\mathsf{VerShare}_d$: every player P_d produces t random and independent sharings $[o^1(d)], \ldots, [o^t(d)]$ of zero, and $[\![o^i]\!]$ is set to be the two-level sharing $[o^i(1)], \ldots, [o^i(n)]$ with $o^i = \sum_d o^i(d)$ for $i \in \{1, \ldots, n\}$. The $[o^i(d)]$'s are considered as part of $\mathcal{S}(d)$ and are authenticated together with the other local base sharings, as part of a block $[\boldsymbol{\sigma}(d)]_d$. This can be done without increasing the asymptotic communication complexity. The list of the two-level sharings $[\![o^1]\!], \ldots, [\![o^t]\!]$ is denoted by $\mathcal{O}$.

3.3 The Input Phase

For every player $P_i \notin \mathcal{C}orr$, and for every input $x \in \mathbb{F}$ of that player to the circuit, a sharing $[\![x]\!]$ needs to be prepared. This job is divided into n^2 segments, where in each segment, $m = c_I/n^2$ such inputs $x^1, \ldots, x^m$ are taken care of. We assume for simplicity that for each segment, the corresponding inputs $x^1, \ldots, x^m$ belong to one player P_i. For the preparation of $[\![x^1]\!], \ldots, [\![x^m]\!]$, the players make use of m multiplication triples $([\![a^1]\!], [\![b^1]\!], [\![c^1]\!]) \in \mathcal{M}$, which are then removed from $\mathcal{M}$.

Protocol Prepare Inputs

Every player not in $\mathcal{D}isp_i$ sends his shares of $[a^1], \ldots, [a^m]$ to P_i. If not all the sharings (not counting shares of players in $\mathcal{D}isp_i$) form correct degree-t sharings, then P_i broadcasts "`fault`" and *fault localization 2* is performed. Otherwise, P_i reconstructs and sends $d^1 = x^1 - a^1, \ldots, d^m = x^m - a^m$ to all the players not in $\mathcal{C}orr$, using a *relay* for the players in $\mathcal{D}isp_i$, broadcasts "`ok`", and *verification* is performed.

Verification: The players generate a challenge λ by means of Challenge, and every player (including P_i) not in $\mathcal{C}orr$ broadcasts $\delta = \sum_k \lambda^k d^k$. If a player P_j broadcasts a different value than P_i, then *fault localization 1* is performed for the smallest such P_j. Otherwise, the players compute $[\![x^k]\!]$ locally as $[\![x^k]\!] = d^k + [\![a^k]\!]$ for $k = 1, \ldots, m$ and halt.

Fault localization 1: If $\{P_i, P_j\} \notin \mathcal{D}isp$ then $\{P_i, P_j\}$ is added to $\mathcal{D}isp$ and Prepare Inputs halts. Otherwise, the *relay* broadcasts δ, and the *relay* together with the player that broadcasted a different value are added to $\mathcal{D}isp$, and Prepare Inputs halts.

Fault localization 2: P_i broadcasts the smallest index k for which he had received inconsistent shares. The players not in $\mathcal{C}orr$ then broadcast their respective shares of $[a^k]$. If these shares form a correct sharing, then P_i broadcasts "`accuse` P_j", where P_j is the smallest player not in $\mathcal{D}isp_i$ that broadcasted a different share than he had initially sent to P_i, and $\{P_i, P_j\}$ is added to $\mathcal{D}isp$ and Prepare Inputs halts. Otherwise, the players take a new $[\![o]\!]$ from $\mathcal{O}$ and broadcast their shares of $[a^k] + [o]$. If these shares are inconsistent then the players apply the procedure AnalyzeSharing below to the sharing $[\![w]\!] = [\![a^k]\!] + [\![o]\!]$. Else, the (implicitly announced) shares of $[o]$ are inconsistent, and the players apply AnalyzeSharing to the sharing $[\![w]\!] = [\![o]\!]$. As a result, a new corrupt player is identified.

The reason for involving the zero-sharing $[\![o]\!] \in \mathcal{O}$ in *fault localization 2* is the following. In order to deal with the inconsistent shares of $[a^k]$, the players need to look at the second-level shares. However, the second-level shares of different secrets are not fully independent, and thus revealing the second-level shares of $[\![a^k]\!]$ would leak information on other secrets. Therefore, we pad $[\![a^k]\!]$ with $[\![o]\!]$, a sharing of zero with fully independent second-level shares, and analyze the second-level shares of either $[\![a^k]\!] + [\![o]\!]$ or $[\![o]\!]$.

The communication complexity of Prepare Inputs (not counting the possible call to AnalyzeSharing) is of the order $O(mn\phi) + O(n\kappa) \cdot \mathcal{BC}$.

Fact 10 *Except with probability $m/2^\kappa$, after* Prepare Inputs *the players hold correct sharings $[\![x^1]\!], \ldots, [\![x^m]\!]$ (of player P_i's input if he remains honest) or a new dispute is found or* AnalyzeSharing *is performed on a sharing $[\![w]\!]$ for which the players not in $\mathcal{C}orr$ have broadcast inconsistent shares $[w]$. Also, if P_i remains honest then no information on x is leaked to the adversary.*

The procedure AnalyzeSharing below allows the players to resolve the following problem: how to find a dishonest player in case the players not in $\mathcal{C}orr$ have broadcasted inconsistent Shamir shares of a two-level shared value $[\![w]\!] = ([w(1)]_1, \ldots, [w(n)]_n)$ that is a known linear combination of the sharings in $\cup\mathcal{M}$ plus a zero-sharing $[\![o]\!] \in \mathcal{O}$, and thus where every $[w(d)]_d$ is a known linear combination of the local base sharings $[s(d)]_d \in \mathcal{S}(d)$, i.e., $[w(d)]_d = \sum_{s(d)\in\mathcal{S}(d)} u_{s(d)}[s(d)]_d$. Recall that $\mathcal{S}(d)$ can be partitioned into n^3 blocks $[\boldsymbol{\sigma}(d)]_d$ with $\boldsymbol{\sigma}(d) \in \mathbb{K}^q$, with each such block being authenticated (for every player P_V). For such a block $[\boldsymbol{\sigma}(d)]_d$, we write $[w(d)|_{\sigma(d)}]_d$ for $\sum_{s(d)\in\sigma(d)} u_{s(d)}[s(d)]_d$, i.e., for the restriction of the linear combination to the sharings in block $\boldsymbol{\sigma}(d)$.

Protocol AnalyzeSharing

Every player $P_j \notin \mathcal{C}orr$ broadcasts the subshares $w_j(1), \ldots, w_j(n)$ he holds of $[\![w]\!]$. If for some player P_j they do not add up to the share w_j, then P_i is declared dishonest and AnalyzeSharing halts. Otherwise, the players proceed as follows. Let d be the smallest number such that the subshares $w_j(d)$ do not form a correct Shamir sharing for the players $P_j \notin \mathcal{C}orr$. If $P_d \notin \mathcal{C}orr$ then P_d broadcasts the index i of a player $P_i \notin \mathcal{C}orr$ who broadcasted an incorrect subshare $w_i(d)$ (note that it may be that $\{P_d, P_i\} \in \mathcal{D}isp$). Then, the players check the correctness of $w_i(d)$, and thus can identify P_d or P_i to be dishonest, by means of *fault localization 1*, given below. Otherwise, i.e. if $P_d \in \mathcal{C}orr$, the players engage into *fault localization 2* to identify a dishonest player P_i that announced an incorrect share $w_i(d)$.

Fault localization 1: By means of a three-round search (with P_i broadcasting n values in each round), the players identify a block $\boldsymbol{\sigma}$ of $\mathcal{S}(d)$, such that the share $z_i(d)$ of $[z(d)]_d = [w(d)|_{\sigma(d)}]_d$ claimed by P_i is claimed to be incorrect by P_d. Note that by definition, $z_i(d)$ is determined by player P_i's share $\boldsymbol{\sigma}_i(d)$. Thus, the players can now verify the correctness of $z_i(d)$ as follows.

Check share: For every player $P_V \notin \mathcal{C}orr$, player P_i sends $\boldsymbol{\sigma}_i(d)$ to P_V, together with the corresponding tag τ. If $z_i(d)$ is consistent with $\boldsymbol{\sigma}_i(d)$, and if $\tau = \boldsymbol{\mu} \odot \boldsymbol{\sigma}_i(d) + \nu$ for the corresponding $\boldsymbol{\mu}$ and ν held by P_V, then P_V broadcasts "`correct`"; else, P_V broadcasts "`incorrect`".

If $t+1$ or more players P_V broadcast "`incorrect`", then P_i is declared corrupt and added to $\mathcal{C}orr$; else, P_d is declared corrupt and added to $\mathcal{C}orr$. Then, AnalyzeSharing halts.

Fault localization 2: By means of a three-round search (with every player broadcasting n values in each round), the players identify a block $\boldsymbol{\sigma}$ of $\mathcal{S}(d)$, such that the players not in $\mathcal{C}orr$ claim an inconsistent sharing of $[z(d)]_d = [w(d)|_{\sigma(d)}]_d$. Now, for every share $z_i(d)$ (with $P_i \notin \mathcal{C}orr$), its correctness is verified as in *fault localization 1* by means of *check share*. The first player whose share is claimed incorrect by $t+1$ or more players P_V, is declared corrupt and added to $\mathcal{C}orr$.

Fact 11 *When* AnalyzeSharing *is performed on a sharing $[\![w]\!]$ for which the players not in $\mathcal{C}orr$ have broadcast inconsistent shares $[w]$, then a new corrupt player is identified except with probability $n/2^\kappa$. Also, the adversary learns no information beyond what he already knows and/or can simulate himself.*

The most expensive part of AnalyzeSharing is *fault localization 2*. The search involves broadcasting $3n^2$ elements in $\mathbb{F}$. Then, invoking *check share* n times requires $O(n^2 \cdot \ell/n) = O(M/n^2)$ elements in $\mathbb{F}$ to be communicated and n^2 bits to be broadcasted. Since AnalyzeSharing finds a new corrupt player, and not just a dispute, it will be invoked at most t times during the whole multi-party computation, and thus contributes $O\big(M\phi/n\big) + O(n^3\phi)\cdot\mathcal{BC}$ to the *total* communication complexity.

3.4 The Computation Phase

During the computation phase, values $[\![w]\!]$ need to be reconstructed, where each $[\![w]\!]$ is a linear combination of the global base sharings. This job is divided into n^2 segments, where in each segment, $m = (c_M + c_O)/n^2$ such values $[\![w]\!]$ need to be reconstructed. These values $[\![w]\!]$ are in general determined adaptively, i.e., depend on previously reconstructed values, with the exact dependencies determined by the circuit. We assume here that the circuit is not too "narrow" so that in each segment *seg*, the set $\mathcal{W}_{seg}$ of shared values to be reconstructed can be divided into m/N blocks $\mathcal{W}^1, \ldots, \mathcal{W}^{m/N}$ of average size $N = n\kappa$ each, so that the values of each block only depend on the values of the previous blocks, and thus can be reconstructed *simultaneously*.

The $[\![w]\!]$'s are reconstructed block-wise by some designated player, and after each block, it is verified that he reconstructed correctly. If some fault is detected, then the players try to detect the cause of the fault; this is tedious but in the end rather straightforward. If this leads to a new dispute between two players, then these two players are added to $\mathcal{D}isp$ and the segment is repeated.

Protocol Rec

Let P_K (the "king") be the first player not in $\mathcal{C}orr$. Sequentially, for each block $\mathcal{W}^k$ (where $k = 1, \ldots, m/N$), consisting of sharings $[\![w^1]\!], \ldots, [\![w^N]\!]$ to be reconstructed, the following *block-reconstruction* and *block-verification* sub-protocols are performed. If one of these blocks *fails* for some player P_i (as specified below), then in all the subsequent blocks, he sends some default symbol $\perp$ for every message he is supposed to send (instead of the actual message).

Block-reconstruction: Every player not in $\mathcal{D}isp_K$ sends his shares of $[w^1], \ldots, [w^N]$ to P_K. If all the sharings (not counting shares of players in $\mathcal{D}isp_K$) form correct degree-t Shamir sharings (this includes receiving no $\perp$ as share), then P_K reconstructs and sends $w^1, \ldots, w^N$ to all the players not in $\mathcal{C}orr$, using a *relay* for the players in $\mathcal{D}isp_K$. Otherwise, P_K sends $\perp$ instead, and the block *fails* for him.

Block-verification: The following is done for every player P_V (acting as verifier). P_V chooses a random $\lambda_V \in \mathbb{K}$ and sends it to every player not in $\mathcal{D}isp_V$, and every player not in $\mathcal{D}isp_V$ sends his share of $[\omega^V] = \sum_k \lambda_V^k [w^k]$ to V. If these shares lie on a degree-t polynomial and reconstruct to $\omega^V = \sum_k \lambda_V^k w^k$, then P_V accepts the reconstruction of this block; otherwise, this block *fails* for P_V.

At then end, every player P_V broadcasts "`ok`" if no block failed for him, or else broadcasts "`fault`". In case all players broadcast "`ok`", Rec halts. Otherwise, the following *fault localization* is performed.

Fault localization: Every player P_V that broadcasted "`fault`", broadcasts the index k for the smallest block $\mathcal{W}^k$ that failed for him, together with an identifier that specifies the cause of the failure, i.e., whether (1) he received $\perp$ from P_K instead of $w^1, \ldots, w^N$, (2) he received $\perp$ from some player P_i instead of the share ω_i^V, (3) the shares do not form a correct sharing, or (4) the shares do not reconstruct to $\omega^V = \sum_k \lambda_V^k w^k$. Now, a fixed verifying player P_V is chosen among all $P_V \notin \mathcal{C}orr$ that broadcasted the smallest value for k. If there is such a P_V with $P_V \notin \mathcal{D}isp_K$ among those, then the first with this property is selected; otherwise, the first among all (that broadcasted the smallest value for k). Depending on the identifier this P_V broadcasted, one of the following is performed (where the case number corresponds to the above enumeration) for the smallest block k P_V complained about.

Case 1: If $P_V \in \mathcal{D}isp_K$, and hence he had received $\perp$ from his *relay* (which by choice of P_V has not complained about this block), P_V and his *relay* are added to $\mathcal{D}isp$ and Rec halts. Otherwise, i.e. if $P_V \notin \mathcal{D}isp_K$, the players proceed as follows. If P_K had not sent $\perp$ to P_V (but $w^1, \ldots, w^N$), then P_K broadcasts "`accuse` P_V", and $\{P_K, P_V\}$ is added to $\mathcal{D}isp$ and Rec halts. Otherwise, P_K broadcasts the index k of the first sharing $[w^k]$ for which the reconstruction failed. The players not in $\mathcal{C}orr$ then need to broadcast their shares of $[w^k]$. If these shares form a correct sharing, then P_K broadcasts "`accuse` P_i", where $P_i \notin \mathcal{D}isp_K$ is a player that had sent a different share to P_K during the *block-reconstruction* procedure, and $\{P_K, P_i\}$ is added to $\mathcal{D}isp$ and Rec halts. Otherwise, i.e. if the shares do not form a correct sharing, the players identify a corrupt player $P_j \notin \mathcal{C}orr$, and P_j is added to $\mathcal{C}orr$ and Rec halts. Identifying P_j is done by taking a new $[\![o]\!]$ from $\mathcal{O}$, broadcasting the shares of $[w^k] + [o]$, and by applying $\mathsf{AnalyzeSharing}$ to $[\![w]\!] = [\![w^k]\!] + [\![o]\!]$ or $[\![w]\!] = [\![o]\!]$, depending on which is incorrect.

Case 2: P_V broadcasts "`accuse` P_i", where $P_i \notin \mathcal{D}isp_V$ is a player that had sent $\perp$ to P_V instead of a share, and $\{P_V, P_i\}$ is added to $\mathcal{D}isp$ and Rec halts.

Case 3: P_V broadcasts λ_V. If a player $P_i \notin \mathcal{D}isp_V$ had received a different value for λ_V during *block-verification*, then he broadcasts "`accuse` P_V", and $\{P_i, P_V\}$ is added to $\mathcal{D}isp$ and Rec halts. Otherwise, every player $P_i \notin \mathcal{D}isp_V$ sends his shares $w_i^1, \ldots, w_i^N$ of $[w^1], \ldots, [w^N]$ to P_V. If $\omega_i^V \neq \sum_k \lambda_V^k w_i^k$ for some player P_i, then P_V broadcasts "`accuse` P_i", and $\{P_V, P_i\}$ is added to $\mathcal{D}isp$ and Rec halts. Else, P_V broadcasts the smallest index k for which the shares w_i^k do not form a correct sharing. The players not in $\mathcal{C}orr$ are then required to broadcast their shares of $[w^k]$. If the broadcast shares do form a correct sharing, then P_V broadcasts "`accuse` P_i", where $P_i \notin \mathcal{D}isp_K$ is a player that had sent a different share w_i^k to P_V before, and $\{P_V, P_i\}$ is added to $\mathcal{D}isp$ and Rec halts. Otherwise, the players take a new $[\![o]\!]$ from $\mathcal{O}$ and broadcast their shares of $[w_i^k] + [o]$, and, as in *case 1*, use $\mathsf{AnalyzeSharing}$ to identify a corrupt player $P_j \notin \mathcal{C}orr$, and P_j is added to $\mathcal{C}orr$ and Rec halts.

Case 4: If $P_V \in \mathcal{D}isp_K$, and hence he had received $w^1, \ldots, w^N$ from his *relay* (which by choice of P_V has not complained about this block), P_V and his *relay* are added to $\mathcal{D}isp$ and Rec halts. Otherwise, $\{P_V, P_K\}$ is added to $\mathcal{D}isp$ and Rec halts.

Protocol Rec (including a possible call to *fault localization*, given below) requires $m/N \cdot O(Nn\phi + n^2\kappa) = O(mn\phi)$ bits of communication, plus $O(n)$ broadcasts. In case the circuit is "narrow" so that the blocks $\mathcal{W}^k$ need to be chosen smaller than specified above, then the communication amounts to $O(mn\phi)$ bits plus an additional $O(n^2\kappa)$ bits per block. Note that in total (over all segments), the number of blocks is bounded by the multiplicative depth d_M of the circuit.

Fact 12 *Except with probability $mn/2^\kappa$, at the end of* Rec *the uncorrupt players hold the correct openings of all $[\![w]\!] \in \mathcal{W}_{seg}$ or fault localization is performed. In the latter case, a new dispute is found. Also, the adversary learns no information beyond w and what he already knows and/or can simulate himself.*

4 Conclusion

We showed that MPC with unconditional security against $t < n/2$ corrupt players is possible with amortized asymptotic near-linear communication complexity $O(n \log n)$ bits per multiplication gate for binary circuits. For circuits over a bigger field $\mathbb{F}$, the $\log n$ term is replaced by $\max\{\log n, \log |\mathbb{F}|\}$. This matches the communication complexity of the best scheme in the much simpler honest-but-curious setting. Room for improvement exists in the terms of the communication complexity that are circuit-size independent, for instance in the $O(n^7\kappa)$ term. Improving this term permits the amortization to step in for smaller circuits.

Acknowledgment

E.B. was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258. S.F. is grateful to UCLA for being a regular host; it is thanks to these visits that this project got started and crucial progress was made. R.O. was supported by NSF grants 0830803, 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, B. John Garrick Foundation, OKAWA Foundation, IBM, Lockheed-Martin Corporation and the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

1. D. Beaver. Multiparty protocols tolerating half faulty processors. In *Advances in Cryptology—CRYPTO '89*, volume 435 of *LNCS*, pages 560–572. Springer, 1989.
2. D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO '91*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.
3. Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 305–328. Springer, 2006.
4. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography Conference (TCC)*, volume 4948 of *LNCS*, pages 213–230. Springer, 2008.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.
6. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology—CRYPTO '09*, volume 7417 of *LNCS*. Springer, 2012.
7. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.
8. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.
9. D. Chaum, I. B. Damgård, and J. Graaf. Multiparty computations ensuring the privacy of each party's input and the correctness of the result. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 87–119. Springer, 1987.
10. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *LNCS*, pages 311–326. Springer, 1999.
11. R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology—EUROCRYPT '00*, volume 1807 of *LNCS*, pages 316–334. Springer, 2000.
12. R. Cramer, I. Damgard, and V. Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. `http://eprint.iacr.org/2011/301`, 2011.
13. I. Damgård and Y. Ishai. Scalable secure multiparty computation. In *Advances in Cryptology—CRYPTO '06*, volume 4117 of *LNCS*, pages 501–520. Springer, 2006.
14. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology—EUROCRYPT '10*, volume 6110 of *LNCS*, pages 445–465. Springer, 2010.
15. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology—CRYPTO '08*, volume 5157 of *LNCS*, pages 241–261. Springer, 2008.
16. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology—CRYPTO '07*, volume 4622 of *LNCS*, pages 572–590. Springer, 2007.

17. S. Goldwasser, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
18. M. Hirt and U. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology—CRYPTO '01*, volume 2139 of *LNCS*, pages 101–118. Springer, 2001.
19. M. Hirt and J. B. Nielsen. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. In *Advances in Cryptology—ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 79–99. Springer, 2005.
20. M. Hirt and J. B. Nielsen. Robust multiparty computation with linear communication complexity. In *Advances in Cryptology—CRYPTO '06*, volume 4117 of *LNCS*, pages 463–482. Springer, 2006.
21. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–85, 1989.
22. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
23. A. Yao. Protocols for secure computations. In *23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.