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 $t < n/2$ 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

R. Safavi-Naini and R. Canetti (Eds.): CRYPTO 2012, LNCS 7417, pp. 663–680, 2012.

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.

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.

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

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

[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.

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 $\log(\max\{n, |\mathbb{F}|\})$.

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.

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 the full version [6].

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, $a^1,\ldots,a^N$, $b^1,\ldots,b^N$ and $c^1,\ldots,c^N$, we wish to verify that $a^i \cdot b^i = c^i$ for all $i = 1,\ldots,N$. 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 $i \in \{1,\ldots,N\}$, where $x_1,\ldots,x_N$ 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

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

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.

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 $P_1, \ldots, P_n$, 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 a 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 $M = 2(c_M + c_R + c_I)$.

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).

[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) 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 $\mathcal{C}orr$).

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 $x_0, x_1, \ldots, x_n \in \mathbb{F}$ with $x_0 = 0$. We also fix an additional $2n^2 - n - 1$ elements $x_{n+1}, \ldots, x_{2n^2-1}$ with the property that every pair x_i, x_j with $i \neq j \in \{0, \ldots, 2n^2 - 1\}$ 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 $w_1, \ldots, w_n \in \mathbb{F}$ 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 $j \in \{1, \ldots, n\}$. Furthermore, share w_j is held by player P_j for $j \in \{1, \ldots, n\}$. 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, 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 $\langle s\rangle_d$, $\lceil s\rfloor^i_d$ and $\langle s\rangle^i_d$. 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]

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

[7] We use set-notation for lists: for a list $\mathcal{L} = (\ell_1, \ldots, \ell_m)$, the expression $\ell \in \mathcal{L}$ is 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.

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*

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

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.

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

Fig. 2. For every multiplication triple $(a, b, c) \in \mathcal{M}$, every $w \in \{a, b, c\}$ is two-level shared as $[\![w]\!]$ (left), and $[w(d)]_d$ is a linear combination of P_d's local base sharings $[s_i(d)]_d$ (center), 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,

[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.

[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.

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 $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ 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 $\llbracket x \rrbracket = d + \llbracket a \rrbracket$. The used triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ 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 $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ 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 $\llbracket z \rrbracket = \llbracket x+y \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket$. If the gate is a multiplication gate, then the output value is computed by using Beavers technique [2] as follows. A fresh multiplication triple $\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket$ is selected and the differences $\llbracket x - a \rrbracket = \llbracket x \rrbracket - \llbracket a \rrbracket$ and $\llbracket y - b \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$ are reconstructed. Then, the output value of the gate is computed locally as $\llbracket z \rrbracket = \llbracket x \cdot y \rrbracket = (x-a)(y-b) + (x-a)\llbracket b \rrbracket + (y-b)\llbracket a \rrbracket - \llbracket c \rrbracket$. 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 $\llbracket w \rrbracket$. 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.

2.7 Two New Essential Ingredients

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

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 $([a^1], [b^1], [c^1]), \ldots, ([a^N], [b^N], [c^N])$ 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 $k \in \{1, \ldots, N\}$. The players can locally compute $[a^k]$ and $[b^k]$ with $f(x_k) = a^k$ and $g(x_k) = b^k$ for all $k \in \{N+1, \ldots, 2N-1\}$. Furthermore, by using the optimistic multiplication protocol, we let them compute $[c^{N+1}], \ldots, [c^{2N-1}]$ 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 $k \in \{1, \ldots, 2N-1\}$. It now holds that all the multiplication triples are correct — i.e., that $c^k = a^k \cdot b^k$ for all $k \in \{1, \ldots, 2N-1\}$ — 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 ok, since the μ's are (partly) re-used, and thus we can also re-use their sharings. Also, sharing ν is ok, 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, it 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.

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.

We now give the detailed protocol for multiparty computing the tag τ. We assume $\boldsymbol{\mu}$ to be shared (component-wise) as $\lceil \mu^1 \rfloor^i_V, \ldots, \lceil \mu^q \rfloor^i_V$. 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 in the latter, P_i and P_V accuse each other anyway. For simpler notation, we write $[\boldsymbol{\sigma}]_d$ instead of $[\boldsymbol{\sigma}(d)]_d$, etc.

Protocol. $\mathsf{TagComp}_{V,i,d}$

Player P_V chooses a random $\nu \in \mathbb{K}$ and shares it (non-verifiably) as $\langle \nu \rangle^i_V$ with Kudzu shares. Similarly, player P_d shares $o = 0$ (i.e., zero) over $\mathbb{K}$ as $\langle o \rangle^i_V$ with Kudzu shares. The players locally compute $\langle \tau \rangle^i = \sum_{k=1}^q [\sigma^k]_d \lceil \mu^k \rfloor^i_V + \langle \nu \rangle^i_V + \langle o \rangle^i_d$ 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 $\langle \nu \rangle^i_V$). The correctness of the computed tags can be verified by a simple cut-and-choose technique; for the details, we refer to the full version [6].

The two crucial observations regarding the efficiency of $\mathsf{TagComp}_{V,i,d}$ are that the twisted sharings $\lceil \mu^k \rfloor^i_V$ can be re-used and thus only need to be prepared *once and for all*, and that the communication complexity of $\mathsf{TagComp}_{V,i,d}$ is *independent* of q, i.e., of the number of shares that are authenticated in one go. As such, the communication complexity of the runs of $\mathsf{TagComp}_{V,i,d}$ is asymptotically negligible; hence, we can authenticate the shares "for free".

Proposition 1 (Privacy of the keys). *If P_V remains honest and the adversary corrupts at most $t-1$ players different to P_i, then the adversary learns no*

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

information on $\boldsymbol{\mu} = (\mu^1, \ldots, \mu^q)$ 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 $\langle\nu\rangle_V^i$ from $\langle\tilde{\nu}\rangle_V^i = \langle\nu - \sigma^1\delta^1\rangle_V^i = \langle\nu\rangle_V^i - [\sigma^1]_d\lceil\delta^1\rfloor_V^i$. But now, since

$$\begin{aligned}
&[\sigma^1]_d\lceil\tilde{\mu}^1\rfloor_V^i + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor_V^i + \langle\tilde{\nu}\rangle_V^i \\
&\qquad = [\sigma^1]_d\lceil\mu^1\rfloor + [\sigma^1]_d\lceil\delta^1\rfloor_V^i + \sum_{k>1}[\sigma^k]_d\lceil\mu^k\rfloor_V^i + \langle\nu\rangle_V^i - [\sigma^1]_d\lceil\delta^1\rfloor_V^i = \langle\tau\rangle^i
\end{aligned}$$

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 $\sum_k \sigma_i^k \mu^k + \nu = \tau$ 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 $\boldsymbol{\sigma} = (\sigma^1, \ldots, \sigma^q)$.*

The proof of Proposition 2 is similar to that of Proposition 1; for the details, we refer to [6].

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 $[s(1)]_1, \ldots, [s(n)]_n$ 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, 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 $\tau = \boldsymbol{\mu}\cdot\boldsymbol{\sigma}_i(d)+\nu$ (where $\boldsymbol{\sigma}_i(d)$ consists of many $s_i(d)$'s), this can be done by computing $\langle\tau\rangle^i = \sum_k [\sigma^k(d)]_d \lceil\mu^k\rfloor^i_V + \langle\nu\rangle^i_V + \langle o\rangle^i_d$. Since the $\boldsymbol{\mu}$'s (and their twisted shares) are re-used to some extent, and since the $\boldsymbol{\sigma}(d)$'s are already shared, the communication complexity is dominated by communicating the shares $\langle\nu\rangle^i_V$, $\langle o\rangle^i_d$ and $\langle\tau\rangle^i$; this consists of a linear number of elements in $\mathbb{K}$ per (large) block $\boldsymbol{\sigma}_i(d)$ (and per P_V and P_i), 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 the full paper [6].

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 [6].

2.9 The Full Protocol

Taking care of all the details when putting the above techniques together is rather cumbersome, and the resulting detailed protocol description and its analysis is quite complex and lengthy. Therefore, due to the space limitation, it is given in the full version [6].

3 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. Beaver, D.: Multiparty Protocols Tolerating Half Faulty Processors. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 560–572. Springer, Heidelberg (1990)
2. Beaver, D.: Efficient Multiparty Protocols Using Circuit Randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)
3. Beerliová-Trubíniová, Z., Hirt, M.: Efficient Multi-party Computation with Dispute Control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006)
4. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-Secure MPC with Linear Communication Complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 1–10 (1988)
6. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority (2011), `http://eprint.iacr.org/2011/629`
7. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences 18(2), 143–154 (1979)
8. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 11–19 (1988)
9. Chaum, D., Damgård, I.B., van de Graaf, J.: Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 87–119. Springer, Heidelberg (1988)
10. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient Multiparty Computations Secure against an Adaptive Adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
11. Cramer, R., Damgård, I., Maurer, U.: General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)
12. Cramer, R., Damgård, I., Pastro, V.: On the Amortized Complexity of Zero Knowledge Protocols for Multiplicative Relations. In: Smith, A. (ed.) ICITS 2012. LNCS, vol. 7412, pp. 62–79. Springer, Heidelberg (2012)

13. Damgård, I., Ishai, Y.: Scalable Secure Multiparty Computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006)
14. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010)
15. Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable Multiparty Computation with Nearly Optimal Work and Resilience. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 241–261. Springer, Heidelberg (2008)
16. Damgård, I., Nielsen, J.B.: Scalable and Unconditionally Secure Multiparty Computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007)
17. Goldwasser, S., Micali, S., Wigderson, A.: 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), pp. 218–229 (1987)
18. Hirt, M., Maurer, U.: Robustness for Free in Unconditional Multi-party Computation. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 101–118. Springer, Heidelberg (2001)
19. Hirt, M., Nielsen, J.B.: Upper Bounds on the Communication Complexity of Optimally Resilient Cryptographic Multiparty Computation. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 79–99. Springer, Heidelberg (2005)
20. Hirt, M., Nielsen, J.B.: Robust Multiparty Computation with Linear Communication Complexity. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 463–482. Springer, Heidelberg (2006)
21. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: 21st Annual ACM Symposium on Theory of Computing (STOC), pp. 73–85 (1989)
22. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
23. Yao, A.: Protocols for secure computations. In: 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 160–164 (1982)