Robustness for Free in Unconditional Multi-Party Computation

Martin Hirt and Ueli Maurer

ETH Zurich, Switzerland

Manuscript, February 12, 2001

Abstract. We present a very efficient multi-party computation protocol unconditionally secure against an active adversary. The security is maximal, i.e., active corruption of up to $t < n/3$ of the n players is tolerated. The communication complexity for securely evaluating a circuit with m multiplication gates over a finite field is $\mathcal{O}(mn^2)$ field elements, including the communication required for simulating broadcast. This corresponds to the complexity of the best known protocols for the passive model, where the corrupted players are guaranteed not to deviate from the protocol. Even in this model, it seems to be unavoidable that for every multiplication gate every player must send a value to every other player, and hence the complexity of our protocol may well be optimal. The constant overhead factor for robustness is small and the protocol is practical.

1 Introduction

1.1 Secure multi-party computation

Secure multi-party computation (MPC), as introduced by Yao [Yao82], allows a set of n players to compute an arbitrary agreed function of their private inputs, even if an adversary may corrupt up to t arbitrary players. Almost any distributed cryptographic protocol can be seen as a multi-party computation, and can be realized with a general MPC protocol. Multi-party computation protocols are an important building block for reducing the required trust and building secure distributed systems. While currently special-purpose protocols (e.g., for collective signing) are considered practical, this paper suggests also that general-purpose protocols may well be practical for realistic applications.

Two different notions of corrupting are usually considered. A passive (or curious) adversary may only read the information stored by the corrupted players, without controlling the player's behavior. Hence only privacy of the inputs is an issue to consider, but not the correctness of the result. In contrast, an active adversary can take full control of the corrupted players. Assuring not only the privacy of the inputs, but also the correctness of the outputs (robustness) appears to entail a substantial overhead. For instance, all known protocols make (usually heavy) use of a broadcast sub-protocol for which the optimal known complexity is $\mathcal{O}(n^2)$.

We briefly review the classical results on secure MPC. Goldreich, Micali, and Wigderson [GMW87] presented a protocol, based on cryptographic intractability assumptions, which allows n players to securely compute an arbitrary function even if an active adversary corrupts any $t < n/2$ of the players. In the secure-channels model, where bilateral secure channels between every pair of players are assumed, Ben-Or, Goldwasser, and Wigderson [BGW88] and independently Chaum, Crépeau, and Damgård [CCD88] proved that unconditional security is possible if at most $t < n/3$ of the players are corrupted. In a model where additionally physical broadcast channels are available, unconditional security is achievable if at most $t < n/2$ players are corrupted [RB89,Bea91b,CDD+99].

1.2 Previous work on efficiency

In the past, both the round complexity and the communication complexity of secure multi-party protocol were subject to many investigations: Protocols with low round complexity [BB89,BFKR90,FKN94,IK00] suffer either from an unacceptably high communication complexity (even quadratic in the number of multiplication gates), or tolerate only a very small number of cheaters.

First steps towards better communication complexity were taken by Franklin and Yung [FY92] and Gennaro, Rabin, and Rabin [GRR98], where first a private but non-resilient computation is performed (for the whole protocol in [FY92], and for a segment of the protocol in [GRR98]), and only in case of faults the computation is repeated with a slow but resilient protocol. Although this approach can improve the best-case complexity of the protocol (when no adversary is present), it cannot speed up the protocol in the presence of a malicious adversary: a single corrupted player can persistently enforce the robust but slow execution, annihilating any efficiency gain.

Recently, Hirt, Maurer, and Przydatek [HMP00] proposed a new protocol for perfectly secure multi-party computation with considerably better communication complexity than previous protocols: A set of n players can compute any function (over a finite field $\mathbb{F}$) which is specified as a circuit with m multiplication gates (and any number of linear gates) by communicating $\mathcal{O}(mn^3)$ field elements, contrasting the previously best complexity of $\mathcal{O}(mn^6)$. Subsequently, the same complexity was achieved by Cramer, Damgård, and Nielsen [CDN01] in the cryptographic model (where more cheaters can be tolerated).

1.3 Contributions

The main open question in this line of research was whether security against active cheaters can be achieved with the same communication complexity as security against passive cheaters, namely with $\mathcal{O}(mn^2)$. We answer this question in the affirmative: The only (and unavoidable) price to to pay for active security is a reduction in the number of tolerable cheaters ($t < n/3$ instead of $t < n/2$). The computation complexity of the new protocol is on the order of the communication complexity and hence not relevant. The achieved communication complexity of $\mathcal{O}(mn^2)$ appears to be optimal. Even in the passive case, it appears unavoidable that every player sends a value to every other player for each multiplication gate.

The new protocol uses Beaver's circuit-randomization technique [Bea91a] and the player-elimination framework from [HMP00].

2 Model

We consider the well-known secure-channels model as introduced in [BGW88]: The set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n players is connected by bilateral synchronous reliable secure channels. Broadcast channels are not assumed to be available. The goal of the protocol is to compute an agreed function, specified as an arithmetic circuit over a finite field $\mathbb{F}$ with $|\mathbb{F}| > n$. The number of multiplication gates in the circuit is denoted by m. To each player P_i a unique public value $\alpha_i \in \mathbb{F} \setminus \{0\}$ is assigned, where for computation efficiency we assume that $\alpha_i = \omega^i$ for some n-th root of unity ω (see Appendix B for details). The computation of the function is secure with respect to a computationally unbounded active adversary that is allowed to corrupt up to t of the players, where t is a given threshold with $t < n/3$. Once a player is corrupted, the adversary can read all his information and can make the player misbehave arbitrarily. The security of our protocol is unconditional with an arbitrarily small probability of error. Formal definitions of security can be found in [Can00] and in [MR98], and our protocols are secure for any of these definitions.

3 Protocol Overview

The protocol proceeds in two phases: In a preparation phase, which could actually be performed as a pre-computation independent of the circuit (except an upper bound on the number m of multiplication gates must be known), m random triples $(a^{(i)}, b^{(i)}, c^{(i)})$ (for $i = 1, \ldots, m$) with $c^{(i)} = a^{(i)}b^{(i)}$ are shared among the players. In the computation phase, the circuit is evaluated gate by gate, where for each multiplication gate one shared triple from the preparation phase is used [Bea91a].

In the preparation phase, some of the players in $\mathcal{P}$ might be eliminated, and the sharings are only among the set $\mathcal{P}' \subseteq \mathcal{P}$ of remaining players. However, it will be guaranteed that the number of corrupted players in $\mathcal{P}'$ is smaller than $(|\mathcal{P}'| - t)/2$, which is sufficient for evaluating the circuit.

As the underlying secret-sharing scheme we use the scheme of Shamir [Sha79], like in most threshold protocols: A value s is t-shared among the players means that every player P_i holds a share s_i, and there exists a polynomial $f(x)$ of degree at most t such that $f(0) = s$ and $f(\alpha_i) = s_i$ for $i = 1, \ldots, n$.

4 Preparation Phase

The goal of this phase is to generate m t-shared random triples $(a^{(i)}, b^{(i)}, c^{(i)})$ with $c^{(i)} = a^{(i)}b^{(i)}$ in such a way that the adversary obtains no information about $a^{(i)}$, $b^{(i)}$, and $c^{(i)}$ (except that $c^{(i)}$ is the product of $a^{(i)}$ and $b^{(i)}$). The generation of these triples makes extensive use of the player-elimination framework of [HMP00]:

Therefore, the triples are generated in blocks of $\ell = \lceil m/n \rceil$ triples. The triples of a block are generated (in parallel) in a non-robust manner; only at the end of the block, consistency is checked jointly for all triples of the block in a single verification procedure (*fault detection*). In case of an inconsistency, a set $\mathcal{D} \subseteq \mathcal{P}$ of *two* players, at least one of whom is corrupted, is identified (*fault localization*) and excluded from further computations (*player elimination*). The triples of the failed block are discarded. Player elimination ensures that at most t blocks fail, and hence in total at most $(n + t)$ blocks must be processed.

More precisely, the consistency verification takes place in *two* steps. In the first verification procedure (fault detection I), the degree of all involved sharings is verified. In other words, the players jointly verify that all sharings produced for generating the triples are of appropriate degree. The second verification step (fault detection II) is performed only if the first verification step is successful. Here, the players jointly verify that for every triple $(a^{(i)}, b^{(i)}, c^{(i)})$, every player shared the correct values such that $c^{(i)} = a^{(i)}b^{(i)}$. If a fault is detected (in either fault-detection procedure), then all triples in the actual block are discarded. Furthermore, a set $\mathcal{D} \subseteq \mathcal{P}$ of two players, one of whom is corrupted, is found (fault localization I, resp. fault localization II) and eliminated from further computations. Note that in the fault-localization procedure, the privacy of the triples is not maintained. Due to the circuit-randomization technique [Bea91a], the triples contain completely random values unrelated with all values of the actual computation.

Both verification steps use n "blinding triples", and the privacy of these triples is annihilated in the verification procedure. Therefore, in each block, $\ell + 2n$ triples are generated. The first verification step verifies the degree of all sharings of the first $\ell + n$ triples, using (and destroying) the remaining n triples for blinding. The second verification step verifies the first ℓ triples, using the remaining n triples for blinding. Note that the second verification step requires that the sharings of all $\ell + n$ involved triples are verified to be correct.

During the generation of the blocks, players can be eliminated. We denote the actual set of players with $\mathcal{P}'$, the actual number of players with $n' = |\mathcal{P}'|$, and the maximum number of cheaters in $\mathcal{P}'$ with t'. Without loss of generality, we assume that $\mathcal{P}' = \{P_1, \ldots, P_{n'}\}$. During the computation, the inequality $2t' < n' - t$ will hold as an invariant. In the beginning, $\mathcal{P}' = \mathcal{P}$, $n' = n$, and $t' = t$, and trivially $2t' < n' - t$ is satisfied. In player elimination, n' will be decreased by 2, and t' by 1. Clearly, this preserves the invariant.

0. Set $\mathcal{P}' = \mathcal{P}$, $n' = n$, and $t' = t$.
1. Repeat until n blocks (i.e., $n\ell \geq m$ triples) succeeded:
   - 1.1 Generate $\ell + 2n'$ triples (in parallel) in a non-robust manner (Sect. 4.1).
   - 1.2 Verify the consistency of all sharings involved in the first $\ell + n'$ triples (fault detection I, Sect. 4.2). If a fault is detected, identify a set $\mathcal{D} \subseteq \mathcal{P}'$ of two players such that at least one player in $\mathcal{D}$ is a cheater, and set $\mathcal{P}'$ to $\mathcal{P}' \setminus D$, n' to $n' - 2$ and t' to $t' - 1$ (fault localization I).
   - 1.3 If no fault was detected in Step 1.2, then verify that in the first ℓ triples, every player shared the correct values (fault detection II, Sect. 4.3). If a fault is detected, identify a set $\mathcal{D} \subseteq \mathcal{P}'$ of two players, at least one of whom is corrupted, and set $\mathcal{P}'$ to $\mathcal{P}' \setminus D$, n' to $n' - 2$ and t' to $t' - 1$ (fault localization II).
   - 1.4 If both verification steps were successful, then the generation of the block was successful, and the first ℓ triples can be used. If either verification procedure failed, then all triples of the actual block are discarded.

4.1 Generate one t-shared triple (a, b, c)

The purpose of this protocol is to generate one t-shared triple (a, b, c), where $c = ab$. The generation of this triple is non-robust: verification will take place only at the end of the block. In particular, in order to share a value, the dealer simply computes the shares and sends them to the players; the consistency verification of the sent shares is delayed.

The generation of the triple is straight-forward: First, the players jointly generate t'-sharings of two random values a and b. This is achieved by having every player share two random values, one for a and one for b, which are then summed up. Then, a t'-sharing of $c = ab$ is computed along the lines of [BGW88,GRR98] (passive model): Every player computes the product of his share of a and his share of b. These product shares define a $2t'$-sharing of c, and c can be computed with Lagrange interpolation. This interpolation is a linear function on the product shares. Hence, a t'-sharing of c can be computed as a linear combination of t'-sharings of the product shares. Finally, the degrees of the sharings of a, b, and c must be increased from t' to t. In order to do so, the players jointly generate three random sharings of 0, each with degree t, and add one of them to the t'-sharings of a, b, and c, respectively.

Note that the protocol for computing a sharing of $c = ab$ relies on the fact that the degree of the sharings of a and b is less than one third of the number of actual players, and it would not work if a and b would be shared with degree t for $3t \geq n'$. On the other hand, it is important that finally the sharings of *all* blocks have the same degree (otherwise the multiplication protocol of Section 5 would leak information about the factors), and t' can decrease from block to block. Therefore, first the triple is generated with degree t', and then this degree is increased to t.

Protocol "Generate"

We give the exact protocol for generating one t-shared triple (a, b, c):

1. The players jointly generate t'-sharings of random values a and b:
   1.1 Every player $P_i \in \mathcal{P}'$ selects two random degree-t' polynomials $\widetilde{f}_i(x)$ and $\widetilde{g}_i(x)$, and hands the shares $\widetilde{a}_{ij} = \widetilde{f}_i(\alpha_j)$ and $\widetilde{b}_{ij} = \widetilde{g}_i(\alpha_j)$ to player P_j for $j = 1, \ldots, n'$.
   1.2 The polynomial for sharing a is $\widetilde{f}(x) = \sum_{i=1}^{n'} \widetilde{f}_i(x)$ (thus $a = \widetilde{f}(0)$), and the polynomial for sharing b is $\widetilde{g}(x) = \sum_{i=1}^{n'} \widetilde{g}_i(x)$ (thus $b = \widetilde{g}(0)$), and every player $P_j \in \mathcal{P}'$ computes his shares of a and b as
   $$\widetilde{a}_j = \sum_{i=1}^{n'} \widetilde{a}_{ij}, \quad \text{and} \quad \widetilde{b}_j = \sum_{i=1}^{n'} \widetilde{b}_{ij}.$$
2. The players jointly compute a t'-sharing of $c = ab$:
   2.1 Every player $P_i \in \mathcal{P}'$ computes his product share $\widetilde{e}_i = \widetilde{a}_i \widetilde{b}_i$, and shares it among the players with the random degree-t' polynomial $\widetilde{h}_i(x)$ (with $\widetilde{h}_i(0) = \widetilde{e}_i$), i.e., sends the share $\widetilde{e}_{ij} = \widetilde{h}_i(\alpha_j)$ to player P_j for $j = 1, \ldots, n'$.
   2.2 Every player P_j computes his share $\widetilde{c}_j$ of c as
   $$\widetilde{c}_j = \sum_{i=1}^{n'} w_i \widetilde{e}_{ij}, \quad \text{where } w_i = \prod_{\substack{j=1 \\ j \neq i}}^{n'} \frac{\alpha_j}{\alpha_j - \alpha_i}.$$
3. The players jointly increase the degree of the sharings of a, b, and c to t (this step is performed only if $t' < t$):
   3.1 Every player $P_i \in \mathcal{P}'$ selects three polynomials $\bar{f}_i(x)$, $\bar{g}_i(x)$, $\bar{h}_i(x)$ of degree $t - 1$ at random, and sends the shares $\bar{a}_{ij} = \bar{f}_i(\alpha_j)$, $\bar{b}_{ij} = \bar{g}_i(\alpha_j)$, and $\bar{c}_{ij} = \bar{h}_i(\alpha_j)$ to player P_j for $j = 1, \ldots, n'$.
   3.2 Every player $P_j \in \mathcal{P}'$ computes his t-shares a_j, b_j, and c_j of a, b, and c, respectively, as follows:
   $$a_j = \widetilde{a}_j + \alpha_j \sum_{i=1}^{n'} \bar{a}_{ij}, \quad b_j = \widetilde{b}_j + \alpha_j \sum_{i=1}^{n'} \bar{b}_{ij}, \quad c_j = \widetilde{c}_j + \alpha_j \sum_{i=1}^{n'} \bar{c}_{ij}.$$

Security analysis

At the end of the block, two verifications will take place: First, it will be verified that the degree of all sharings is as required (t', respectively $t - 1$, Section 4.2). Second, it will be verified that in Step 2.1, every player P_i indeed shares his correct product share $\widetilde{e}_i = \widetilde{a}_i \widetilde{b}_i$ (Section 4.3). In the sequel, we analyze the security of the above protocol under the assumption that these two conditions are satisfied.

After Step 1, obviously the assumption that the degree of all sharings is as required immediately implies that the resulting shares $\widetilde{a}_1, \ldots, \widetilde{a}_{n'}$ (respectively $\widetilde{b}_1, \ldots, \widetilde{b}_{n'}$) lie on a polynomial of degree t', and hence define a valid sharing. Furthermore, if at least one player in $P_i \in \mathcal{P}'$ honestly selected *random* polynomials $\widetilde{f}_i(x)$ and $\widetilde{g}_i(x)$, then a and b are random and unknown to the adversary.

In Step 2, we need the observation that c can be computed by Lagrange interpolation [GRR98]:

$$c = \sum_{i=1}^{n'} w_i \widetilde{e}_i, \quad \text{where } w_i = \prod_{\substack{j=1 \\ j \neq i}}^{n'} \frac{\alpha_j}{\alpha_j - \alpha_i}.$$

Assuming that every player P_i really shares his correct product share $\widetilde{e}_i$ with a polynomial $\widetilde{h}_i(x)$ of degree t', it follows immediately that the polynomial $\widetilde{h}(x) = \sum_{i=1}^{n'} w_i \widetilde{h}_i(x)$ is also of degree t', and furthermore

$$\widetilde{h}(0) = \sum_{i=1}^{n'} w_i \widetilde{h}_i(0) = \sum_{i=1}^{n'} w_i \widetilde{e}_i = c.$$

The privacy is guaranteed because the adversary does not obtain information about more than t' shares of any polynomial $\widetilde{h}_i(x)$ (for any $i = 1, \ldots, n'$).

Step 3 is only performed if $t' < t$. Assuming that the polynomials $\bar{f}_i(x)$, $\bar{g}_i(x)$, and $\bar{h}_i(x)$ of every player $P_i \in \mathcal{P}'$ have degree at most $t - 1$, it immediately follows that all the polynomials defined as

$$\bar{f}(x) = \sum_{i=1}^{n'} \bar{f}_i(x), \quad \bar{g}(x) = \sum_{i=1}^{n'} \bar{g}_i(x), \quad \bar{h}(x) = \sum_{i=1}^{n'} \bar{h}_i(x)$$

also all have degree at most $t - 1$. Hence, the polynomials $x\bar{f}(x)$, $x\bar{g}(x)$, and $x\bar{h}(x)$ have degree at most t, and they all share the secret 0. Thus, the sums $\widetilde{f}(x) + x\bar{f}(x)$, $\widetilde{g}(x) + x\bar{g}(x)$, and $\widetilde{h}(x) + x\bar{h}(x)$ are of degree t and share a, b, and c, respectively. The privacy of the protocol is obvious for $t' \leq t - 1$.

Complexity analysis

We briefly analyze the communication complexity of the above protocol: Every sharing requires n field elements to be sent, and in total there are $6n$ sharings, which results in a total of $6n^2$ field elements to be communicated per triple.

4.2 Verification of the degrees of all sharings in a block

The goal of this fault-detection protocol is to verify the degree of the sharings of $\ell + n'$ triples in a single step, using (and destroying) another n' triples.

The basic idea of this protocol is to verify the degree of a *random linear combination* of the polynomials. More precisely, every player distributes a random challenge vector of length $\ell + n'$ with elements in $\mathbb{F}$, and the corresponding linear combinations of each involved polynomial is reconstructed towards the challenging player, who then checks that the resulting polynomial is of appropriate degree. In order to preserve the privacy of the involved polynomials, for each verifier one additional blinding polynomial of appropriate degree is added. If a verifier detects a fault (i.e., one of the linearly combined polynomials has too high degree), then the triples of the actual block are discarded, and in a fault-localization protocol, a set $\mathcal{D} \subseteq \mathcal{P}'$ of two players, at least one of whom is corrupted, is found and eliminated.

Protocol "Fault-detection I"

The following steps for verifying the degree of all sharings in one block are performed in parallel, once for every verifier $P_v \in \mathcal{P}'$:

1. The verifier P_v selects a random vector $[r_1, \ldots, r_{\ell+n'}]$ with elements in $\mathbb{F}$ and sends it to each player $P_j \in \mathcal{P}'$.
2. Every player P_j computes and sends to P_v the following corresponding linear combinations (plus the share of the blinding polynomial) for every $i = 1, \ldots, n'$:

$$\widetilde{a}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \widetilde{a}_{ij}^{(k)} \quad + \quad \widetilde{a}_{ij}^{(\ell+n'+v)} \qquad\qquad \bar{a}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \bar{a}_{ij}^{(k)} \quad + \quad \bar{a}_{ij}^{(\ell+n'+v)}$$

$$\widetilde{b}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \widetilde{b}_{ij}^{(k)} \quad + \quad \widetilde{b}_{ij}^{(\ell+n'+v)} \qquad\qquad \bar{b}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \bar{b}_{ij}^{(k)} \quad + \quad \bar{b}_{ij}^{(\ell+n'+v)}$$

$$\widetilde{c}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \widetilde{c}_{ij}^{(k)} \quad + \quad \widetilde{c}_{ij}^{(\ell+n'+v)} \qquad\qquad \bar{c}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell+n'} r_k \bar{c}_{ij}^{(k)} \quad + \quad \bar{c}_{ij}^{(\ell+n'+v)}$$

3. P_v verifies whether for each $i = 1, \ldots, n'$, the shares $\widetilde{a}_{i1}^{(\Sigma)}, \ldots, \widetilde{a}_{in'}^{(\Sigma)}$ lie on a polynomial of degree at most t'. The same verification is performed for the shares $\widetilde{b}_{i1}^{(\Sigma)}, \ldots, \widetilde{b}_{in'}^{(\Sigma)}$ and for the shares $\widetilde{c}_{i1}^{(\Sigma)}, \ldots, \widetilde{c}_{in'}^{(\Sigma)}$, for $i = 1, \ldots, n'$. Furthermore, P_v verifies whether for each $i = 1, \ldots, n'$, the shares $\bar{a}_{i1}^{(\Sigma)}, \ldots, \bar{a}_{in'}^{(\Sigma)}$ lie on a polynomial of degree at most $t-1$. The same verification is performed for the shares $\bar{b}_{i1}^{(\Sigma)}, \ldots, \bar{b}_{in'}^{(\Sigma)}$ and for the shares $\bar{c}_{i1}^{(\Sigma)}, \ldots, \bar{c}_{in'}^{(\Sigma)}$ for $i = 1, \ldots, n'$.
4. Finally, P_v broadcasts (using an appropriate sub-protocol) one bit according to whether all the $6n'$ verified polynomials have degree at most t', respectively $t-1$ (confirmation), or at least one polynomial has too high degree (complaint).

Protocol "Fault-localization I"

This protocol is performed if and only if at least one verifier has broadcasts a complaint in Step 4 of the above fault-detection protocol. We denote with P_v the verifier who has reported a fault. If there are several such verifiers, the one with the smallest index v is selected.

5. The verifier P_v selects one of the polynomials of too high degree and broadcasts the location of the fault, consisting of the index i and the "name" of the sharing ($\widetilde{a}$, $\widetilde{b}$, $\widetilde{c}$, $\bar{a}$, $\bar{b}$, or $\bar{c}$). Without loss of generality, we assume that the fault was observed in the sharing $\widetilde{a}_{i1}^{(\Sigma)}, \ldots, \widetilde{a}_{in'}^{(\Sigma)}$.
6. The owner P_i of this sharing (i.e., the player who acted as dealer for this sharing) sends to the verifier P_v the correct linearly combined polynomial $\widetilde{f}_i^{(\Sigma)}(x) = \sum_{k=1}^{\ell+n'} \widetilde{f}_i^{(k)}(x) + \widetilde{f}_i^{(\ell+n'+v)}(x)$.
7. P_v finds the (smallest) index j such that $\widetilde{a}_{ij}^{(\Sigma)}$ (received from P_j in Step 2) does not lie on the polynomial $\widetilde{f}_i^{(\Sigma)}(x)$ (received from the owner P_i in Step 6), and broadcasts j among the players in $\mathcal{P}'$.
8. Both P_i and P_j send the list $\widetilde{a}_{ij}^{(1)}, \ldots, \widetilde{a}_{ij}^{(\ell+n')}, \widetilde{a}_{ij}^{(\ell+n'+v)}$ to P_v.
9. P_v verifies that the linear combination $[r_1, \ldots, r_{\ell+n'}]$ applied to the values received from P_i is equal to $\widetilde{f}_i^{(\Sigma)}(\alpha_j)$. Otherwise, P_v broadcasts the index i, and the set of players to be eliminated is $\mathcal{D} = \{P_i, P_v\}$. Analogously, P_v verifies the values received from P_j to be

consistent with $\widetilde{a}_{ij}^{(\Sigma)}$ received in Step 2, and in case of failure broadcasts the index j, and $\mathcal{D} = \{P_j, P_v\}$.

10. P_v finds the (smallest) index k such that the values $\widetilde{a}_{ij}^{(k)}$ received from P_i and P_j differ, and broadcasts k and both values $\widetilde{a}_{ij}^{(k)}$ from P_i and $\widetilde{a}_{ij}^{(k)}$ from P_j.
11. Both P_i and P_j broadcast their value of $\widetilde{a}_{ij}^{(k)}$.
12. If the values broadcast by P_i and P_j differ, then the localized set is $\mathcal{D} = \{P_i, P_j\}$. If the value broadcast by P_i differs from the value that P_v broadcast (and claimed to be the value received from P_i), then $\mathcal{D} = \{P_i, P_v\}$. Else, $\mathcal{D} = \{P_j, P_v\}$.

Security analysis

It follows from simple algebra that if all players are honest, then the above fault-detection protocol will always pass. On the other hand, if at least one of the involved sharings (in any of the $\ell + n'$ triples) has too high degree, then every honest verifier will detect this fault with probability at least $1/|\mathbb{F}|$. For at least $n' - t' \geq n - 2t$ honest players, this gives an overall error probability of at most $|\mathbb{F}|^{-(n-2t)}$.

The correctness of the fault-localization protocol can be verified by inspection. There is no privacy issue in this protocol; the generated triples are discarded.

Complexity analysis

The fault-detection protocol requires $n(n(\ell+n)+6n^2) = n^2\ell + 7n^3$ field elements to be sent and n bits to be broadcast. For fault localization, up to $n + 2(\ell + n + 1) = 2\ell + 3n + 2$ field elements must be sent and $2\log n + \log 6 + \log(\ell + n + 1) + 4\log|\mathbb{F}|$ bits must be broadcast.

4.3 Verification that all players share the correct product shares

It remains to verify that in each triple $k = 1, \ldots, \ell$, every player P_i shared the correct product share $\widetilde{e}_i^{(k)} = \widetilde{a}_i^{(k)}\widetilde{b}_i^{(k)}$ (Step 2.1 of protocol Generate). Since it is already verified that the sharings of all factor shares are of degree t', it is sufficient to verify that the shares $\widetilde{e}_1^{(k)}, \ldots, \widetilde{e}_{n'}^{(k)}$ lie on a polynomial of degree at most $2t'$. Note that the at least $n' - t' > 2t'$ shares of the honest players uniquely define this polynomial. The key idea of this verification protocol is the same as in the previous verification protocol: Every verifier P_v distributes a random challenge vector, and the corresponding linear combination of the polynomials (plus one blinding polynomial) is opened towards P_v. If a fault is detected, then a set $\mathcal{D}$ of two players (one of whom is corrupted) can be found with the fault-localization protocol.

Protocol "Fault-detection II"

The following steps are performed for each verifier $P_v \in \mathcal{P}'$ in parallel.

1. The verifier P_v selects a random vector $[r_1, \ldots, r_\ell]$ with elements in $\mathbb{F}$ and sends it to each player $P_j \in \mathcal{P}'$.
2. Every player P_j computes and sends to P_v the following linear combinations (with blinding) for every $i = 1, \ldots, n'$:

$$\widetilde{e}_{ij}^{(\Sigma)} = \sum_{k=1}^{\ell} r_k \widetilde{e}_{ij}^{(k)} \quad + \quad \widetilde{e}_{ij}^{(\ell+v)}.$$

3. P_v verifies whether for each $i = 1, \ldots, n'$ the shares $\widetilde{e}_{i1}^{(\Sigma)}, \ldots, \widetilde{e}_{in'}^{(\Sigma)}$ lie on a polynomial of degree at most t', and if so, whether the secrets $\widetilde{e}_1^{\Sigma}, \ldots, \widetilde{e}_{n'}^{\Sigma}$ of the above sharings (computed by interpolating the corresponding share-shares) lie on a polynomial of degree at most $2t'$. P_v broadcasts one bits according to whether all polynomials have appropriate degree (confirmation), or at least one polynomial has too high degree (complaint).

Protocol "Fault-localization II"
We denote with P_v the verifier who has reported a fault in Step 3 of the above fault-detection protocol. If there are several such verifiers, the one with the smallest index v is selected.

4. If in Step 3, the degree of one of the second-level sharings $\widetilde{e}_{i1}^{(\Sigma)}, \ldots, \widetilde{e}_{in'}^{(\Sigma)}$ was too high, then P_v applies error-correction to find the smallest index j such that $\widetilde{e}_{ij}^{(\Sigma)}$ must be corrected (cf. Appendix B). Since all sharings have been verified to have correct degree, P_v can conclude that P_j has sent the wrong value $\widetilde{e}_{ij}^{(\Sigma)}$. P_v broadcasts the index j, and the set of players to be eliminated is $\mathcal{D} = \{P_j, P_v\}$ (and the following steps need not be performed).
5. Every player P_i sends to P_v all his factor shares $\widetilde{a}_i^{(1)}, \ldots, \widetilde{a}_i^{(\ell)}, \widetilde{a}_i^{(\ell+v)}$ and $\widetilde{b}_i^{(1)}, \ldots, \widetilde{b}_i^{(\ell)}, \widetilde{b}_i^{(\ell+v)}$.
6. P_v verifies for every $k = 1, \ldots, \ell, \ell+v$ whether the shares $\widetilde{a}_1^{(k)}, \ldots, \widetilde{a}_{n'}^{(k)}$ lie on a polynomial of degree t'. If not, then P_v applies error-correction and finds and broadcasts the (smallest) index j such that $\widetilde{a}_j^{(k)}$ must be corrected. The set of players to be eliminated is $\mathcal{D} = \{P_j, P_v\}$. The same verification is performed for the shares $\widetilde{b}_1^{(k)}, \ldots, \widetilde{b}_{n'}^{(k)}$ for $k = 1, \ldots, \ell, \ell+v$.
7. P_v verifies for every $i = 1, \ldots, n'$ whether the value $\widetilde{e}_i^{\Sigma}$ computed in Step 4 is correct, i.e., whether
$$\widetilde{e}_i^{\Sigma} \stackrel{?}{=} \sum_{k=1}^{\ell} r_k \widetilde{a}_i^{(k)} \widetilde{b}_i^{(k)} \quad + \quad \widetilde{a}_i^{(\ell+v)} \widetilde{b}_i^{(\ell+v)}.$$
This test will fail for at least one i, and P_v broadcasts this index i. The players in $\mathcal{D} = \{P_i, P_v\}$ are eliminated.

Security analysis
It follows from simple algebra that if all players are honest, then the above fault-detection protocol will always pass. On the other hand, if the degree of at least one of the involved sharings is higher than $2t'$, then every honest verifier will detect this fault with probability at least $1/|\mathbb{F}|$. For at least $n' - t' \geq n - 2t$ honest players, this makes an overall error probability of at most $|\mathbb{F}|^{-(n-2t)}$.

The correctness of the fault-localization protocol can be verified by inspection. There is no privacy issue in this protocol; the generated triples are discarded.

Complexity analysis
The fault-detection protocol requires $n(n\ell + n^2) = n^2\ell + n^3$ elements to be sent, and n bits to be broadcast. The fault-localization protocol requires $2n(\ell+1)$ field elements to be sent and $\log n$ bits to be broadcast.

5 Computation Phase

The evaluation of the circuit is along the lines of the protocol of [Bea91a]. Slight modifications are needed because the degree t of the sharings and the upper bound t' on the number of cheaters need not be equal. Furthermore, special focus is given to the fact that in our protocol, also eliminated players must be able to give input to and receive output from the computation.

From the preparation phase, we have m random triples $(a^{(i)}, b^{(i)}, c^{(i)})$ with $c^{(i)} = a^{(i)}b^{(i)}$, where the sharings are of degree t among the set $\mathcal{P}'$ of players. The number of corrupted players in $\mathcal{P}'$ is at most t' with $2t' < n' - t$, where $n' = |\mathcal{P}'|$. This is sufficient for efficient computation of the circuit.

5.1 Input sharing

First, every player who has input secret-shares it (with degree t) among the set $\mathcal{P}'$ of players. We use the verifiable secret-sharing protocol of [BGW88] (with perfect security), with a slight modification to support $t \neq t'$. The dealer is denoted by P, and the secret to be shared by s. We do not assume that $P \in \mathcal{P}'$ (neither $P \in \mathcal{P}$).

1. The dealer P selects at random a polynomial $f(x, y)$ of degree t in both variables, with $p(0, 0) = s$, and sends the polynomials $f_i(x) = f(\alpha_i, x)$ and $g_i(x) = p(x, \alpha_i)$ to player P_i for $i = 1, \ldots, n'$.
2. Every player $P_i \in \mathcal{P}'$ sends to P_j for $j = i + 1, \ldots, n'$ the values $f_i(\alpha_j)$ and $g_i(\alpha_j)$.
3. Every player P_j broadcasts one bit according to whether all received values are consistent with the polynomials $f_j(x)$ and $g_j(x)$ (confirmation) or not (complaint).
4. If no player has broadcast a complaint, then the secret-sharing is finished, and the share of player P_j is $f_j(0)$. Otherwise, every player P_j who has complaint broadcasts a bit vector of length n', where a 1-bit in position i means that one of the values received from P_i was not consistent with $f_j(x)$ or $g_j(x)$. The dealer P must answer all complaints by broadcasting the correct values $f(\alpha_i, \alpha_j)$ and $f(\alpha_j, \alpha_i)$.
5. Every player P_i checks whether the values broadcast by the dealer in Step 4 are consistent with his polynomials $f_i(x)$ and $g_i(x)$, and broadcasts either a confirmation or an accusation. The dealer P answers every accusation by broadcasting both polynomials $f_i(x)$ and $g_i(x)$ of the accusing player P_i, and P_i replaces his polynomials by the broadcast ones.
6. Every player P_i checks whether the polynomials broadcast by the dealer in Step 5 are consistent with his polynomials $f_i(x)$ and $g_i(x)$, and broadcasts either a confirmation or an accusation.
7. If in Steps 5 and 6, there are in total at most t' accusations, then every player P_i takes $f_i(0)$ as his share of s. Otherwise, clearly the dealer is faulty, and the players take a default sharing (e.g., the constant sharing of 0).

It is clear that an honest player never accuses an honest dealer. On the other hand, if there are at most t' accusations, then the polynomials of at least $n' - 2t' > t$ honest players are consistent, and these polynomials uniquely define the polynomial $f(x, y)$ with degree t. Hence, the polynomials of all honest players are consistent, and their shares $f_1(0), \ldots, f_{n'}(0)$ lie on a polynomial of degree t.

This protocol communicates $3n^2$ field elements, and it broadcasts n bits (in the best case), respectively $n^2 + 3n + 2t^2 \log |\mathbb{F}|$ bits (in the worst case).

5.2 Evaluation of the circuit

The circuit is evaluated gate by gate. Linear gates can be evaluated without any communication due to the linearity of the used sharing. Multiplication gates are evaluated according to [Bea91a]:

Assume that the factors x and y are t-shared among the players. Furthermore, a t-shared triple (a, b, c) with $c = ab$ is used. The product xy can be written as follows:

$$xy = ((x-a)+a)((y-b)+b) = ((x-a)(y-b)) + (x-a)b + (y-b)a + c.$$

The players in $\mathcal{P}'$ reconstruct the differences $d_x = x - a$ and $d_y = y - b$. This reconstruction is possible because $2t' < n' - t$ (cf. Appendix B). Note that reconstructing these values does not give any information about x or y, because a and b are random. Then, the following equation holds:

$$xy = d_x d_y + d_x b + d_y a + c.$$

This equation is linear in a, b, and c, and we can compute linear combinations on shared values without communication. This means that the players can compute the above linear combination on their respective shares of x and y and they receive a t-sharing of the product xy. More details can be found in [Bea91a].

This multiplication protocol requires two secret-reconstructions per multiplication gate. Secret-reconstruction requires every player in $\mathcal{P}'$ to send his share to every other player (who then applies error-correction to the received shares and interpolates the secret). The communication costs per multiplication gate are hence $2n^2$. Broadcast is not needed.

5.3 Output reconstruction

Any player P can receive output (not only players in $\mathcal{P}'$ or in $\mathcal{P}$). In order to reconstruct a shared value x towards player P, every player in $\mathcal{P}'$ sends his share of x to P, who then applies error-correction and interpolation to compute the output x. In the error-correction procedure, up to $(n' - t - 1)/2 \geq t'$ errors can be corrected (see Appendix B).

Reconstructing one value requires n field elements of communication, and no broadcast.

5.4 Probabilistic functions

The presented protocol is for deterministic functions only. In order to capture probabilistic functions, one can generate one (or several) blocks with single values $a^{(i)}$ only (with simplified verification), and use these values as shared randomness.

Alternatively, but somewhat wastefully, one just picks the value $a^{(i)}$ from a shared triple $(a^{(i)}, b^{(i)}, c^{(i)})$, and discards the rest of the triple. Then, m denotes the number of multiplication gates plus the number of "randomness gates".

5.5 On-going computations

In an on-going computation, inputs and outputs can be given and received at any time during the computation, not only at the beginning and at the end. Furthermore, it might even not be specified beforehand which function will be computed. And example of an on-going computation is the simulation of a fair stock market.

In contrast to the protocol of [HMP00], the proposed protocol can easily be extended to capture the scenario of on-going computations. First, the players generate ℓ triples (a, b, c) with $c = ab$, and perform the computation until all triples are exhausted. Then, a new block of ℓ triples is generated, and so on.

6 Complexity Analysis

A detailed complexity analysis is given in Appendix A. Here we summarize the most important results: Let n denote the number of players, $\mathbb{F}$ the field over which the function (circuit) is defined, m the number of multiplication gates in the circuit, C_d the depth of the circuit, n_I the number of inputs and n_O the number of outputs of the function. Evaluating this circuit securely with respect to an active adversary corrupting any $t < n/3$ of the players is possible with communicating $14mn^2 + \mathcal{O}(n_I n^4 + n_O n + n^4)$ field elements. The number of communication rounds is $C_d + \mathcal{O}(n^2)$. All complexities include the costs for simulating broadcast.

This complexity should be compared with the complexity of the most efficient protocols. In the secure-channels model, the most efficient protocol for unconditionally secure multi-party protocols [HMP00] requires $\mathcal{O}(mn^3)$ field elements in $\mathcal{O}(C_d + n^2)$ rounds (where both hidden constants are slightly higher than ours).

For completeness, we also compare the complexity of our protocol with the complexity of the most efficient protocol for the cryptographic model [CDN01]. This protocol requires a communication complexity of $\mathcal{O}(mn^3)$ field elements in $\mathcal{O}(C_d n)$ rounds. The high round complexity results from the fact that the protocol invokes a broadcast sub-protocol for each multiplication gate. The most efficient broadcast protocols require $\mathcal{O}(n)$ rounds. Constant-round broadcast protocols are known [FM88], but they have higher communication complexities and would results in a communication complexity of $\mathcal{O}(mn^5)$ field elements.

Finally, we compare the protocol with the most efficient known protocol for passive security, namely [BGW88] with the simplification of [GRR98]. This protocol communicates $mn^2 + \mathcal{O}(n_I n + n_O n)$ field elements. Hence, providing robustness can be achieved with a communication overhead of about factor 14.

7 Conclusions and Open Problems

We have presented a protocol for secure multi-party computation unconditionally secure against an active adversary which is (up to a small constant factor) as efficient as protocols with passive security. The protocol provides some (arbitrarily small) probability of error. Note that due to the player-elimination technique, this error-probability does not grow with the length of the protocol (like in all previous MPC protocols with error probability), but only in the upper bound t of the number of corrupted players.

It remains open whether quadratic complexity can also be achieved in the model of an active computationally-bounded adversary (cryptographic model), where up to $t < n/2$ of the players may be corrupted. The most efficient result for this model requires communication of $\mathcal{O}(n^3)$ field elements (and $O(n)$ rounds!) per multiplication gate [CDN01].

Also, it would be interesting to combine the techniques of this paper with techniques of papers with protocols that require a constant number of rounds only (but have a high communication complexity), to achieve a multi-party protocol which has both low communication complexity and very low round complexity.

Furthermore, the presented protocol is for the synchronous model. Some real-world networks appear to be more appropriately modeled by the asynchronous model, and the protocol must be adapted for this setting. It seems that this can be done along the lines of [BCG93,Can95,SR00].

Finally, it would be interesting to have a proof that quadratic complexity is optimal for passive security. This would immediately imply that the protocol of this paper is optimally efficient (up to a constant factor).

References

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 201–210, Aug. 1989.

[BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proc. 25th ACM Symposium on the Theory of Computing (STOC)*, pp. 52–61, 1993.

[Bea91a] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 420–432, 1991.

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

[BFKR90] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BGP89] P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 410–415, 1989.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[Bla84] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1984.

[Can95] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

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

[CDD+99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pp. 311–326, 1999.

[CDN01] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology — EUROCRYPT '01*, Lecture Notes in Computer Science, 2001. To appear.

[FKN94] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pp. 554–563, 1994.

[FM88] P. Feldman and S. Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 148–161, 1988.

[FY92] M. K. Franklin and M. Yung. Communication complexity of secure computation. In *Proc. 24th ACM Symposium on the Theory of Computing (STOC)*, pp. 699–710, 1992.

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

[GRR98] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 101–111, 1998.

[HMP00] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In T. Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pp. 143–161. Springer-Verlag, Dec. 2000.

[IK00] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st IEEE Symposium on the Foundations of Computer Science (FOCS)*, Oct. 2000.

[MR98] S. Micali and P. Rogaway. Secure computation: The information theoretic case. Manuscript, 1998. Former version: Secure computation, In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Note in Computer Science*, pp. 392–404, Springer-Verlag, 1991.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 73–85, 1989.

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

[SR00] K. Srinathan and C. P. Rangan. Efficient asynchronous secure multiparty distributed computation. In *Indocrypt 2000*, Lecture Notes in Computer Science, Dec. 2000.
[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 160–164. IEEE, 1982.

A Detailed Complexity Analysis

We summarize the complexities of all involved sub-protocols. For each sub-protocol, we indicate both the message complexity (MC, in communicated field elements) and the broadcast complexity (BC, in bits) of the protocol involved once, and specify how often the protocol is called at least (when no adversary is present) and at most (when the corrupted players misbehave in the most effective way). The complexity of the verifiable secret-sharing protocol of [BGW88], which is used for giving input, depends on whether or not some of the players misbehave. We list both complexities.

In the table, n denotes the number of players, t the upper bound on the number of actively corrupted players, m the total number of multiplication gates, ℓ the number of multiplication gates per block, n_I the number of inputs to the function, and n_O the number of outputs of the function.

The indicated complexities are upper bounds: In particular, when a player has to send a message to all players, we count this as n messages (instead of $n-1$).

What	**MC** (field elements)	**BC** (bits)	**#Calls** (min... max)	
Generate triples	$6n^2$	—	$n(\ell+2n) \ldots (n+t)(\ell+2n)$	(1)
Fault detection I	$\ell n^2 + 7n^3$	n	$n \ldots n+t$	(2)
Fault localization I	$2\ell + 3n + 2$	$2\log n + 4\log\lvert\mathbb{F}\rvert + \log(\ell+n+1) + \log 6$	$0 \ldots t$	(3)
Fault detection II	$\ell n^2 + n^3$	n	$n \ldots n+t$	(4)
Fault localization II	$2\ell n + 2n$	$\log n$	$0 \ldots t$	(5)
Give input (best)	$3n^2$	n	n_I	(6)
Give input (worst)	$3n^2$	$n^2 + 3n + 2t^2 \log\lvert\mathbb{F}\rvert$	n_I	(7)
Multiply	$2n^2$	—	m	(8)
Get output	n	—	n_O	(9)

We add up the above complexities for $\ell \leq m/n+1$, $n \geq 4$, and $t \leq n/3$. In order to simplify the expressions, some of the terms are slightly rounded up.

In the best case (when no cheating occurs), $10mn^2 + 22n^4 + 3n_In^2 + n_On$ field elements are communicated and $2n^2 + n_In$ bits are broadcast. Applying the broadcast protocol of [BGP89] (which communicates $9n^2$ bits for broadcasting one bit), this results in a total complexity of less than $10mn^2\log\lvert\mathbb{F}\rvert + 22n^4(\log\lvert\mathbb{F}\rvert + 1) + n_In^2(3\log\lvert\mathbb{F}\rvert + 9n) + n_On\log\lvert\mathbb{F}\rvert$ bits.

In the worst case, the protocol communicates $13mn^2 + 30n^4 + 3n_In^2 + n_On$ field elements and broadcasts $3n^2 + 2n\log\lvert\mathbb{F}\rvert + \frac{n}{3}\log m + n_In^2\log\lvert\mathbb{F}\rvert$ bits. Simulating broadcast with [BGP89], this gives less than $14mn^2\log\lvert\mathbb{F}\rvert + 35n^4(\log\lvert\mathbb{F}\rvert + 1) + 9n_In^4\log\lvert\mathbb{F}\rvert + n_On\log\lvert\mathbb{F}\rvert$ bits. This is about $14mn^2 + \mathcal{O}(n_In^4 + n_On + n^4)$ field elements.

B Error-correction with Erasures

In the protocol of this paper, we often need to interpolate a shared value from a subset of the shares. In this appendix, we briefly summarize the techniques used for performing these interpolations efficiently.

We assume that to each player $P_i \in \mathcal{P}$ a unique value $\alpha_i \in \mathbb{F} \setminus \{0\}$ is assigned. We say that a secret s is t-shared among the players in $\mathcal{P}$ if there exists a degree-t polynomial $f(x)$ with $f(0) = s$, and every player $P_i \in \mathcal{P}$ holds a share $s_i = f(\alpha_i)$. It is well-known that correcting up to t_f faulty shares in a codeword $s_1, \ldots, s_n$ is possible as long as $2t_f < n - t$. Furthermore, when $\alpha_i = \omega^i$ (for $i = 1, \ldots, n$) for an n-th root of unity ω, the shares $s_1, \ldots, s_n$ correspond to a codeword in a Reed-Solomon code of length n with minimum distance $n - t$, and error-correction can be performed efficiently (polynomial in n).

This error-correction procedure can be slightly generalized to capture erasures in the codeword: Error-correction in a Reed-Solomon code is possible and efficient if up to t_f shares are faulty and up to t_e shares are omitted (erasures), as long as $2t_f + t_e < n - t$ (see e.g. [Bla84, Sect. 9.2] for details).

This generalized procedure can be used for two purposes: First, instead of requiring existence of an n-th root of unity in $\mathbb{F}$, it is sufficient to require existence of an $\bar{n}$-th root of unity for some $\bar{n} \geq n$. The $\bar{n} - n$ missing shares in each sharing are treated as erasures of the code. Note that an n-th root of unity exists in $\mathbb{F}$ if and only if n divides $|\mathbb{F}| - 1$, and for many settings of n and $\mathbb{F}$, such a root simply does not exist.

Second, it captures the setting with player elimination. After a sequence of k player eliminations, we have $n' = n - 2k$ remaining players, where up to $t' = t - k$ of them are corrupted. The n' shares of the remaining players define a codeword with $t_e = \bar{n} - n' = \bar{n} - n + 2k$ erasures and $t_f = t' = t - k$ faults. Such a sharing can be interpolated efficiently for $2(t-k) + (\bar{n} - n + 2k) < \bar{n} - t$, which is satisfied for $t < n/3$.