Efficient Secure Multi-Party Computation*

(Extended Abstract)

Martin Hirt[1], Ueli Maurer[1], and Bartosz Przydatek[2]**

[1] ETH Zurich, Switzerland,
{hirt,maurer}@inf.ethz.ch
[2] Carnegie Mellon University, USA,
bartosz@cs.cmu.edu

Abstract. Since the introduction of secure multi-party computation, all proposed protocols that provide security against cheating players suffer from very high communication complexities. The most efficient unconditionally secure protocols among n players, tolerating cheating by up to $t < n/3$ of them, require communicating $\mathcal{O}(n^6)$ field elements for each multiplication of two elements, even if only one player cheats.
In this paper, we propose a perfectly secure multi-party protocol which requires communicating $\mathcal{O}(n^3)$ field elements per multiplication. In this protocol, the number of invocations of the broadcast primitive is independent of the size of the circuit to be computed. The proposed techniques are generic and apply to other protocols for robust distributed computations.
Furthermore, we show that a sub-protocol proposed in [GRR98] for improving the efficiency of unconditionally secure multi-party computation is insecure.

1 Introduction

1.1 Secure Multi-Party Computation

The goal of secure multi-party computation, as introduced by Yao [Yao82], is to enable a set of n players to compute an arbitrary agreed function of their private inputs. The computation must guarantee the correctness of the outputs while preserving the secrecy of the players' inputs, even if some of the players are corrupted by an active adversary and misbehave maliciously.

As the first general solution to this problem, 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 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,

* Research supported by the Swiss National Science Foundation (SNF), SPP project no. 5003-045293. Full version at http://www.inf.ethz.ch/department/TI/um/ .
** Research done at ETH Zurich, Switzerland.

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 143–161, 2000.

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 Efficiency Considerations

All proposed multi-party protocols that provide security against misbehaving players suffer from high communication complexities. This is in sharp contrast to their private (but non-resilient) counterparts, for which reasonably efficient solutions are known [BGW88]. The communication overhead of resilient multi-party protocols over private protocols is due mainly to the sophisticated techniques for achieving resilience against faults. Specifically, these techniques make extensive use of a broadcast primitive, which must be realized with a protocol for Byzantine agreement (e.g., [PSL80,DFF$^+$82,FM88,BGP89,CW89]). Such protocols are very communication-intensive. The necessity of the broadcast channel is independent of whether or not actual faults occur: often broadcast is used to complain about an inconsistency, but when no inconsistency is detected, the players must nevertheless broadcast a confirmation message (the inherent information of the message is one bit). Many researchers take a broadcast channel for granted, neglecting the fact that this primitive does not exist in most realistic scenarios for distributed computing, and hence must be simulated. Broadcast is an efficiency bottleneck, in both information-theoretic and cryptographic settings; reducing the number of broadcast invocations is therefore crucial for reducing the overall communication complexity of distributed protocols.

There is a line of research that focused on reducing the communication complexity of multi-party protocols. First, several works [BB89,BMR90,BFKR90] concentrated on reducing the round complexity of such protocols. However, the price for the low round complexity is a substantially increased message complexity. With the current results, namely $\mathcal{O}(n^6)$ field elements per multiplication, the main efficiency bottleneck seems to be the message complexity rather than the round complexity. First steps towards lower message complexities were taken in [BFKR90]. The proposed protocol is very efficient, but it only tolerates adversaries corrupting up to $t = \mathcal{O}(\log n)$ players. Protocols with optimal resilience (i.e., $t < n/3$) were proposed in [FY92] and in [GRR98]. Their approach is to first perform a private protocol with fault-detection (for the whole protocol in [FY92], and for a part of the protocol in [GRR98]), and only in case of faults to repeat the computation 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 (and even inverting) any efficiency gain.

1.3 Contributions

This paper significantly improves the message complexity of unconditionally secure multi-party computations, without increasing the round complexity in a relevant manner. We consider a set of n players, where up to $t < n/3$ of them can be corrupted by a computationally unbounded, adaptive, active adversary. We present a protocol that allows the players to securely compute an agreed function specified as an arithmetic circuit over a finite field $\mathbb{F}$, requiring communication of $\mathcal{O}(mn^3)$ field elements, where m denotes the number of multiplication gates in the circuit. The total number of invocations of the broadcast primitive in the whole protocol is only $\mathcal{O}(n^2)$, independent of the circuit size.

This is to be compared with the most efficient unconditionally secure protocol known so far, namely the protocol of Beaver [Bea91a], which requires $\mathcal{O}(mn^6)$ field elements. Other protocols whose goal is to improve the message complexity of unconditionally secure multi-party protocols [FY92,GRR98] fail to do so in the presence of faults. The new protocol improves even on the cryptographically secure protocol [GRR98], which communicates $\mathcal{O}(mn^4)$ field elements[1] (but tolerates up to $t < n/2$ corruptions). Recently, a protocol with cryptographic security for evaluating Boolean circuits was proposed in which $\mathcal{O}(mn^3k)$ bits are communicated, where k is a security parameter [CDN00]. The round complexities of all considered protocols are essentially equal. All stated complexities include the costs of simulating the broadcast channels by a protocol for Byzantine agreement.

The techniques that allow this speed-up are generic and apply to many protocols for general multi-party computation as well as to special-purpose protocols, in both the cryptographic model and the information-theoretic model. One key technique is *player elimination*. In contrast to previous protocols where only evident misbehavior leads to elimination and where slowing down the protocol is still possible without being detected, we proceed more rigorously: Whenever a fault occurs (and slows down the protocol execution), a set of players which contains at least a certain number of corrupted players (but possibly also some honest ones) is identified and eliminated from the further protocol execution. This ensures that faults occur only rarely, namely at most t times during the entire computation, which in turn allows to reduce the number of consistency checks performed in the protocol: Rather than after each gate, the consistency checks are performed only after a sequence of gates, a so-called *segment*. During the entire computation, up to t segments can fail and require re-computation, but with an appropriate size of the segments, the total cost of re-computation will be much smaller than the savings due to the reduced number of the checks.

Furthermore, we show that the very efficient protocol of [GRR98] for the verification of equality of shared values is insecure (cf. App. A), thus invalidating previously stated efficiency improvements.

[1] In this protocol, the field must be large for security reasons.

1.4 Outline

In Sect. 2 we introduce the general framework for efficient resilient protocols. This framework is not specific for multi-party computation. The new multi-party computation protocol is described in Sect. 3, and its efficiency is analyzed and compared with known protocols in Sect. 4. Finally, some conclusions and open problems are mentioned in Sect. 5.

2 Framework for Efficient Resilient Protocols

2.1 Introduction

Distributed protocols resilient against misbehavior of some of the players require in general much more communication than their private (but non-resilient) counterparts, even when no cheating occurs. The reasons for this contrast are two-fold: First, in a model where players might deviate from the protocol, expensive consistency checks must be performed frequently, and agreement must be reached on whether or not faults occurred. Second, if indeed at least one player misbehaves, then inconsistencies will occur, and costly fault-recovery procedures must be applied. Note that the consistency checks are necessary even when no cheating occurs, whereas fault recovery is necessary only when at least one player misbehaves.

In this section, we describe a framework for efficient resilient protocols that overcomes these disadvantages. The key idea is to eliminate at least one malicious player (and potentially some honest players) each time a fault is detected. Hence the number of fault-recovery invocations is bounded by the maximal number of corrupted players and is independent of the length of the protocol. Furthermore, the resulting seldom occurrence of faults allows to reduce the frequency of consistency checks and thereby to significantly reduce the communication-overhead caused by them.

The techniques presented in this section apply to many applications in several models, including those relying on intractability assumptions. The adversary can be static or adaptive, but not mobile: A mobile adversary [OY91,CH94] may release some of the corrupted players during the protocol execution and thereby regain the capability of corrupting new players, which contradicts the idea of elimination of corrupted players.

2.2 Incorporating Resilience into a Private Protocol

We consider a private protocol that proceeds in rounds (e.g., in each round one gate is evaluated) and wish to execute this protocol in a resilient manner. In contrast to the classical approach to resilient protocols, where after each round some consistency checks are performed and agreement on whether or not a fault occurred is reached, we divide the protocol into *segments*, each consisting of a sequence of rounds, and only at the end of each segment the consistency of the data held by the players is checked and the players agree on whether or not a fault occurred (*fault detection*). If a fault is detected, then a set of players is identified which contains at least a certain number of cheaters (*fault localization*), the players in the set are eliminated from the further protocol execution (*player elimination*), and the failed segment is repeated (*fault correction*). If privacy is an issue, then after each round some checks must be performed, but no agreement must be reached on the fact whether or not a fault occurred (*weak fault detection*).

During a protocol consisting of m rounds, the classical approach invokes m times fault detection and, if at least one player misbehaves permanently, m times fault-recovery. In our approach, where the protocol is divided into segments of m_s rounds, only the weak fault detection is invoked m times. Fault detection is performed m/m_s times, and fault localization, player elimination, and fault correction are invoked at most t times. By selecting m_s appropriately, the overhead for the (in total up to t) repetitions of a segment will not dominate the total complexity of the protocol, and the costs of fault detection and fault localization are independent of m (and polynomial in n). In many applications, this will significantly reduce the overall complexity of the protocol.

We now describe the steps in more detail:

1. **Private computation with weak fault detection.** All rounds of the segment are computed according to the private computation. The computation of this step must be *verifiable*, i.e. it must be possible to check later (see below) whether or not any faults occurred. However, *robustness* is not required, i.e. if faults occur, then the computation may fail (in such a case it must be possible to perform an appropriate fault localization, see below). In order to preserve privacy even in case of faults, consistency checks are performed after each round, and every player sends to every other player one bit indicating whether or not he observed an inconsistency. A player who observed or was informed about an inconsistency will use default (random) dummy values unrelated to the actual values in all further rounds of the segment.
2. **Fault detection.** The goal of fault detection is to reach agreement on whether or not a fault occurred during the current segment. Typically, fault detection is achieved by having every player broadcast (with a protocol for Byzantine agreement) a binary message according to whether or not he observed or was informed about an inconsistency in any round of the current segment, and a fault is detected if at least one player complains. The following steps 3. to 5. are performed if and only if a fault is detected.
3. **Fault localization.** The purpose of fault localization is to find out which players are corrupted or, because agreement about this can usually not be reached, at least to narrow down the set of players containing the cheaters. The output of fault localization is a set $\mathcal{D}$ with $|\mathcal{D}| = p$ players, guaranteed to contain at least r cheaters, denoted as (r, p)-*localization*.
4. **Player elimination.** The set $\mathcal{D}$ agreed upon during fault localization is eliminated from the further computation. In general, after eliminating the players in $\mathcal{D}$, the protocol cannot be continued immediately, but it must be

transformed to capture the new setting with $n-p$ players and at most $t-r$ cheaters.

5. **Fault correction.** Since some players are eliminated whenever a fault is detected, faults can be corrected simply by repeating the current segment of the protocol.

3 Constructing Efficient Multi-Party Computation Protocols

In this section we present a construction of efficient multi-party computation protocols in the secure-channels model, based on the framework with player-elimination from the previous section. We first formally define the considered model, then we describe the main (top-level) protocol and finally all required sub-protocols.

3.1 Model

We consider the well-known secure-channels model as used in [BGW88,CCD88]: 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 inputs to the circuit is denoted by n_I, the total number of outputs by n_O,[2] the number of multiplication gates in the circuit by m, and the multiplicative depth by d (i.e., the maximal number of multiplication gates in any path of the circuit). To each player P_i a unique public value $\alpha_i \in \mathbb{F} \setminus \{0\}$ is assigned. There are no further assumptions about the field.[3] The computation of the function must be secure with respect to a computationally unbounded adaptive active adversary who can 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 perfect, i.e. unconditional with zero failure probability. Formal definitions of security can be found in [Can00] and in [MR98], and our protocol is secure for any of these definitions.

To simplify the presentation, we adopt the following convention throughout the description of the protocols: Unless otherwise stated, whenever a player does not receive an expected message, or receives a malformed message, then a default value for this message is taken.

[2] n_O specifies the total number of outputs — if the same value is given as output to several players, then this value is counted several times.

[3] This is in contrast to the protocol in [BGW88], where the existence of an n-th root of unity in $\mathbb{F}$ is assumed.

3.2 Main Protocol

The protocol follows the classical approach for secure multi-party computation: First, each player secret-shares his input(s) among the players. Second, the circuit is evaluated with the shared values. Third, the output value(s) are reconstructed towards the authorized players.

According to the framework from Sect. 2, the circuit will be divided into segments. If the evaluation of a segment fails, then some players are eliminated and the segment is repeated. Clearly, *all* players must be able to provide input and receive output, including players that are eliminated in the protocol evaluation (also honest players can be eliminated). This is achieved by using a resilient protocol (which does not make use of the player-elimination technique) for sharing input values. No special measures are necessary for receiving output, because the secret-reconstruction protocol can also be performed towards an eliminated player (this player only receives values and cannot cause inconsistencies).

Sharing. The sharing is based on Shamir's secret-sharing scheme [Sha79], extended to a two-dimensional sharing [GHY87,BGW88,CCD88,RB89,FHM98]. Each value is shared among the players with a polynomial of degree t, and each share is again shared among the players with a polynomial of degree t. Formally, a value s is *t-shared* among the players if there exist degree-t polynomials f and $f_1, \ldots, f_n$ with $s = f(0)$ and $f_i(0) = f(\alpha_i)$. The information held by player P_i is the share $s_i = f(\alpha_i)$, the polynomial f_i, and the share-shares $s_{ji} = f_j(\alpha_i)$ (for $j = 1, \ldots, n$). The polynomials in the sharing must be randomly chosen such that any set of t players does not obtain any information about the secret.

Segmentation. Due to the linearity of the secret-sharing scheme, linear functions of shared values can be computed non-interactively, and hence only multiplication gates are relevant for the communication complexity. In order to partition the circuit with m multiplication gates and multiplicative depth d into segments, we select an ordering of the gates which satisfies the partial order defined by the circuit (i.e., the inputs of the i-th gate must be provided by gates with index smaller than i). Every segment consists of a number of consecutive gates, subject to the following bounds:

- the number m_s of multiplication gates in each segment is at most $\lceil m/n \rceil$,
- the multiplicative depth d_s of each segment is at most $\lceil d/n \rceil$.

Furthermore, in every segment (except the last) at least one of the above bounds is satisfied with equality, hence the total number of segments is smaller than $2n$.

At the end of every segment, fault detection is performed and agreement is reached on whether or not a fault occurred within the segment. If no fault occurred, then the computation of this segment is completed, and the next segment is started. If a fault is detected, then a $(1, 2)$-localization $\mathcal{D} \subset \mathcal{P}$ will be found and eliminated (we will not consider other types of localizations), and the evaluation of the segment is repeated. During the whole circuit evaluation, at most t segments fail. The described segmentation guarantees that the repeated computation will not dominate the overall protocol complexity, neither in terms of the number of communicated bits nor in terms of the number of communication rounds.

Protocol overview. Let $\mathcal{P}$ denote the set of players, where $n = |\mathcal{P}|$, and $t < n/3$ the upper bound on the number of cheaters. During the computation, players can be eliminated, and then $\mathcal{P}'$ will denote the set of remaining players, $n' = |\mathcal{P}'|$, and t' the upper bound on the number of cheaters in this set.

0. Set $\mathcal{P}' := \mathcal{P}$, $n' := n$, $t' := t$.
1. Input stage: Every player P providing input secret-shares his input value (Sect. 3.3).
2. Computation stage (Sect. 3.4): For each segment of the circuit:
   2.1 For each gate in the segment (all gates at the same level can be evaluated in parallel):
      - If the gate is linear: Call the sub-protocol for the evaluation of linear functions.
      - If the gate is a multiplication gate: Call the multiplication sub-protocol. Players that have detected (or were notified about) a fault earlier in this segment use default shares.

   2.2 For each $P_i \in \mathcal{P}'$, broadcast one bit according to whether or not a fault was observed (or notified) in the segment. If at least one player reports a fault, then the segment fault-localization procedure is invoked to find a $(1,2)$-localization $\mathcal{D}$, and $\mathcal{P}'$ is set to $\mathcal{P}' \setminus \mathcal{D}$, t' is set to $t' - 1$, and step 2. is restarted (for the same segment).
3. Output stage: For every player P that is to receive output: Call the sub-protocol for receiving output (Sect. 3.5).

3.3 Input Stage

In the input stage, every player secret-shares his input(s). Let $\mathcal{P}$ be the set of players, at most t of which are corrupted, and let P be a designated dealer holding a secret input s. The protocol for providing s as input is a variation of the verifiable secret-sharing (VSS) protocol of Ben-Or, Goldwasser and Wigderson [BGW88]:

1. Distribution. The dealer P selects at random a polynomial $p(x,y) = \sum_{i,j=0}^{t} r_{ij} x^i y^j$ of degree t in both variables, where $p(0,0) = s$, and sends the polynomials $f_i(x) = p(x, \alpha_i)$ and $\widetilde{f}_i(y) = p(\alpha_i, y)$ to player P_i (for $i = 1, \ldots, n$).[4] This implicitly defines the polynomial $f(x) = p(0,x)$.
2. Consistency checks. Each pair of players P_i, P_j (for $1 \le i, j \le n$) checks whether $f_i(\alpha_j) \stackrel{?}{=} \widetilde{f}_j(\alpha_i)$. For this, P_i sends $f_i(\alpha_j)$ to P_j, and P_j checks whether the received value is equal to $\widetilde{f}_j(\alpha_i)$.

[4] An efficiency gain of a factor 2 can be achieved by setting $r_{ij} = r_{ji}$, and hence $f_i(x) = \widetilde{f}_i(x)$. One can prove that privacy is not violated by this technique. See [CDM00] for more details.

3. Complaint stage. Every player broadcasts a message (containing one bit) indicating whether all consistency checks were successful or at least one test failed. In case of a complaint, the player afterwards broadcasts a bit-vector, where the j-th bit indicates whether or not the player has observed an inconsistency with player P_j. The dealer answers the complaints by broadcasting the corresponding correct values.
4. Accusation stage. If a player P_j observes more than t inconsistencies or discovers that the dealer's answers contradict his own values, he broadcasts an accusation. In such a case the dealer broadcasts both polynomials $f_j(x)$ and $\widetilde{f}_j(y)$. The published polynomials can cause some new inconsistencies with the values held by some other players, who react again with accusations, and so on.[5] If more than t players have accused, or if the dealer did not answer all the complaints and accusations, a default sharing (e.g., the constant sharing of 0) is taken.

In the protocol of [BGW88], the share of player P_i is $s_i = f(\alpha_i) = f_i(0)$, and the second dimension of the sharing is not used. In our scheme, the share of player P_i is the polynomial f_i (and in particular $s_i = f_i(0)$), as well as the share-shares $s_{ji} = \widetilde{f}_i(\alpha_j) = p(\alpha_i, \alpha_j)$ (for $j = 1, \ldots, n$).

In order to analyze the security of this secret-sharing protocol we distinguish two cases: (a) If the dealer is honest, all shares and share-shares of honest players will be consistent, and only values held by corrupted players can be published. No honest player will accuse the dealer, hence there will be at most t accusations. Clearly, in this case the outcome will be a proper t-sharing. (b) If the dealer is corrupted, then at the end of the protocol (if there were not more than t accusations) the cross-over points of all honest players are consistent, and their share-shares uniquely define a two-dimensional polynomial $p'(x, y)$, satisfying the conditions for a proper t-sharing. If there were more than t accusations, then at least one of the accusations origins from an honest player, and indeed the dealer is cheating. In this case it is legitimate to take some default value as the dealer's secret.

3.4 Computation Stage

The computation of the circuit proceeds segment by segment. We denote the current set of players with $\mathcal{P}'$, where $n' = |\mathcal{P}'|$, and the current upper bound on the number of cheaters in $\mathcal{P}'$ with t'. Without loss of generality, we assume that $\mathcal{P}' = \{P_1, \ldots, P_{n'}\}$. A segment is computed as follows: First, the gates of the segment are computed. Linear functions can be computed robustly (as no communication is needed). In contrast, the computation of multiplication gates is private and verifiable, but not robust. At the end of each multiplication sub-protocol, the (honest) players inform each other in a weak fault detection

[5] One can show that two rounds of accusations are sufficient to reach agreement. After two rounds of accusations, either the total number of accusations exceeds t, or all accusations in the second round originate from corrupted players.

procedure whether or not they observed an inconsistency. If a player observed such an inconsistency, or was informed about one in weak fault detection, then he continues the computation of the segment with default values independent of the actual shares. At the end of each segment, fault detection is performed and, if necessary, fault localization, player elimination and fault correction.

Linear functions. Let $\mathcal{L}$ be a linear function, and assume that the values $a, b, \ldots$ are t-shared with polynomials $f, f_1, \ldots, f_{n'}$, $g, g_1, \ldots, g_{n'}$, $\ldots$, respectively. Due to the linearity of $\mathcal{L}$, the polynomials $h = \mathcal{L}(f, g, \ldots)$ and $h_i = \mathcal{L}(f_i, g_i, \ldots)$ define a t-sharing of $c = \mathcal{L}(a, b, \ldots)$. Hence, player P_i can compute his share of c as $h_i = \mathcal{L}(f_i, g_i, \ldots)$ and $c_{ji} = \mathcal{L}(a_{ji}, b_{ji}, \ldots)$ (for $j = 1, \ldots, n'$). The privacy of this protocol is trivial (there is no communication), and the correctness is due to the linearity of the sharing.

Multiplication. The crucial sub-protocol for multiplication is a *re-sharing* protocol. A re-sharing protocol is a protocol that takes a degree-γ sharing of a value s and generates an independent degree-δ sharing of s. This re-sharing is possible in a verifiable (but non-robust) manner if $t' < n' - \gamma$. Privacy can be guaranteed if $t' \leq \gamma$ and $t' \leq \delta$.

The protocol for computing the t-shared product c of two t-shared values a and b proceeds in three steps: First, both inputs a and b are re-shared with degree t'. Second, every player locally multiplies his respective shares and share-shares of a and b, resulting in a degree-$2t'$ sharing of c. And third, this degree-$2t'$ sharing of c is re-shared to a degree-t sharing.

We have to show that the necessary (and sufficient) conditions for all re-sharings are satisfied: After a sequence of k $(1, 2)$-localizations and eliminations, we have $n' = n - 2k$ and $t' = t - k$. The requirements for the re-sharing are $t' < n' - t$ and $t' < n' - 2t'$, and both are satisfied for $3t < n$.

Re-sharing protocol. The goal of re-sharing is to transform a γ-sharing of a value s into a proper and independent δ-sharing of s, where $t' < n' - \gamma$, $t' \leq \gamma$ and $t' \leq \delta$. The re-sharing sub-protocol can fail in the presence of malicious players. However, if it fails, all (honest) players will learn so, and at the end of the segment, agreement on whether or not such a fault occurred will be reached and the segment will be repeated if necessary.

Roughly speaking, our re-sharing protocol works along the lines of degree reduction of [BGW88,GRR98], but it is significantly more efficient, due to various techniques in the spirit of the player-elimination framework (cf. Sect. 2).

Assume that s is γ-shared with the polynomials f and $f_1, \ldots, f_{n'}$, and player P_i holds the polynomial $f_i(x)$ (hence his share $s_i = f_i(0)$), and his share-shares $s_{ji} = f_j(\alpha_i)$ (for $j = 1, \ldots, n'$). The value s can be expressed as a linear combination (Lagrange interpolation) of the values $s_1, \ldots, s_{n'}$ [BGW88,GRR98]. Therefore, once the values $s_1, \ldots, s_{n'}$ are δ-shared, the required δ-sharing of s can be computed by a distributed evaluation of the appropriate linear function (as described in Sect. 3.4). Thus, the re-sharing can be performed as follows: Every

player δ-shares his share s_i, proves that the shared value is indeed s_i, and computes his degree-δ share of s as a linear combination of the received shares of $s_1, \ldots, s_{n'}$.
We describe the steps in more detail:

1. Non-robust VSS. Every player P_i shares his share s_i with the degree-δ polynomials $h^{(i)}$, $h_1^{(i)}, \ldots, h_{n'}^{(i)}$ in a non-robust but verifiable manner. The protocol works like the first two steps of the VSS in the input stage (Sect. 3.3):
   a) P_i selects at random a polynomial $p^{(i)}(x,y)$ of degree δ in both variables, where $p^{(i)}(0,0) = s_i$, and sends the polynomials $h_j^{(i)}(x) = p^{(i)}(x,\alpha_j)$ and $\widetilde{h}_j^{(i)}(y) = p^{(i)}(\alpha_j,y)$ to player P_j (for $j = 1,\ldots,n'$). This implicitly defines the polynomial $h^{(i)}(x) = p^{(i)}(0,x)$.
   b) Each pair of players P_j, P_k (for $1 \leq j,k \leq n'$) verifies the equality of their common shares. For this, P_j sends $h_j^{(i)}(\alpha_k)$ to P_k, who then checks whether the received value is equal to $\widetilde{h}_k^{(i)}(\alpha_j)$.
2. Proving correctness. Every player P_i proves that $h^{(i)}(0) = f_i(0)$ by showing that the free coefficient of the polynomial $h^{(i)}(x) - f_i(x)$ is equal to zero. This is done in two steps:
   a) Let $\mu = \max(\gamma,\delta)$. P_i computes the polynomial $g^{(i)}(x) := (h^{(i)}(x) - f_i(x))/x$ (whose degree is at most $\mu-1$), and distributes the shares on $g^{(i)}$ among the players. For this purpose the non-robust VSS protocol from Step 1 is used, where the corresponding two-dimensional polynomial, say $q^{(i)}(x,y)$, is chosen randomly, but such that $q^{(i)}(0,x) = g^{(i)}(x)$.
   b) Every player P_k checks whether $\alpha_k g^{(i)}(\alpha_k) = h^{(i)}(\alpha_k) - f_i(\alpha_k)$.
3. Weak fault detection. Every player sends to every other player one bit indicating whether or not any of his consistency checks in Steps 1, 2a and 2b, have failed.
4. Lagrange interpolation. Every player P_i who has neither detected nor was informed about any inconsistencies computes his degree-δ share of s as a linear combination of his shares of $s_1,\ldots,s_{n'}$.

It is easy to see (using basic algebra), that if no player has reported inconsistencies during the weak fault detection, then the result of re-sharing is a proper δ-sharing of s. Otherwise, if at least one (honest) player has sent or received a bit indicating inconsistencies, it will be possible to identify a $(1,2)$-localization.

Fault detection. At the end of the segment, every player P_i *broadcasts* one bit indicating whether or not an inconsistency was observed by or reported to P_i in one of the re-sharing protocols in the segment. If all players broadcast a confirmation (i.e., no inconsistency was observed), then the computation of the segment is completed and the next segment can be started. If at least one player broadcasts a complaint, then fault localization is invoked.

Fault localization. The goal of fault-localization is to identify a $(1,2)$-localization $\mathcal{D}$, i.e. a set $\mathcal{D} \subset \mathcal{P}$ containing two players, at least one of them being corrupted. These players will then be eliminated from the protocol, and hence fault localization is invoked at most t times.

The two players to be eliminated are selected from the players involved in the first fault that occurred in the current segment. In order to determine the first fault, every player who complained during fault detection broadcasts the index (relative to the segment) of the re-sharing protocol, in which for the first time an inconsistency occurred, together with a number denoting the step of the re-sharing protocol in which the fault was detected (Step 1, 2a or 2b), or reported (Step 3). Among all the broadcast indices the smallest one is selected. Let P_k denote the player who complained about the selected re-sharing protocol.[6] The method of determining the $(1,2)$-localization $\mathcal{D}$ depends on the step of the re-sharing protocol in which the first fault appeared. Four cases must be distinguished:

(i) The first fault is in Step 1, i.e. for some i and j, the value $h_j^{(i)}(\alpha_k)$ sent by P_j differs from $\widetilde{h}_k^{(i)}(\alpha_j)$:
P_k broadcasts i, j, and $\widetilde{h}_k^{(i)}(\alpha_j)$. On this request, P_j broadcasts $\widetilde{h}_j^{(i)}(\alpha_k)$, and P_i broadcasts $p^{(i)}(\alpha_k, \alpha_j)$. Given these three values, the set $\mathcal{D}$ is determined as follows:
  – If $\widetilde{h}_k^{(i)}(\alpha_j) = h_j^{(i)}(\alpha_k)$, then $\mathcal{D} := \{P_j, P_k\}$, else
  – if $p^{(i)}(\alpha_k, \alpha_j) \neq \widetilde{h}_k^{(i)}(\alpha_j)$, then $\mathcal{D} := \{P_i, P_k\}$, else
  – $p^{(i)}(\alpha_k, \alpha_j) \neq h_j^{(i)}(\alpha_k)$, and $\mathcal{D} := \{P_i, P_j\}$.

(ii) The first fault is in Step 2a: analogously to the case (i).

(iii) The first fault is in Step 2b, i.e., for some i the check $\alpha_k g^{(i)}(\alpha_k) \stackrel{?}{=} h^{(i)}(\alpha_k) - f_i(\alpha_k)$ failed:
According to P_k, player P_i is cheating, so P_k broadcasts the index i, and $\mathcal{D}$ is set to $\{P_i, P_k\}$.

(iv) The first fault is in Step 3, i.e., P_k claims that in Step 3 some player reported a fault to him:
Since no player admits the discovery of an inconsistency (as follows from the rule for choosing P_k), obviously either P_k is lying or the player who reported the fault to him was malicious. P_k broadcasts the index i of the player P_i who in Step 3 reported the fault to him, and $\mathcal{D}$ is set to $\{P_i, P_k\}$.

It is obvious that all players find the same set $\mathcal{D}$, and that in each case at least one player in $\mathcal{D}$ is corrupted, hence $\mathcal{D}$ is a $(1,2)$-localization.

Player elimination. All players set $\mathcal{P}'$ to $\mathcal{P}' \setminus \mathcal{D}$, and reduce t' to $t' - 1$.

Fault correction. Fault correction is achieved by repeating the failed segment. Since after each failure at least one malicious player is eliminated, at most t segments will be repeated in a complete protocol run.

[6] If there are several such players, we consider those who have broadcast the smallest step-number, and from that group the player with the smallest index k is chosen.

3.5 Output Stage

Let P be the designated player supposed to receive a value s that is t-shared among the players in $\mathcal{P}'$ with the polynomials f and $f_1, \ldots, f_{n'}$. First, every player $P_i \in \mathcal{P}'$ sends the polynomial $f_i(x)$ and the share-shares $s_{1i}, \ldots s_{n'i}$ to P. Then, P interpolates the secret s from the shares $s_i = f_i(0)$ for all i where $f_i(x)$ is consistent with all but (at most) t' share-shares s_{ij}. Note that this protocol needs neither error correction nor broadcast.

The privacy of this protocol is obvious. The correctness can be proven as follows: At most t' players send a bad polynomial $f_i' \neq f_i$, and they will be inconsistent with at least $n' - t - t' > t'$ share-shares. Hence, P will ignore bad polynomials and interpolate the correct secret s.

4 Complexity Analysis

In this section we analyze the communication complexity of the proposed multi-party computation protocol and compare it with the most efficient protocols known before. We focus on the case when an adversary is present and neglect the efficiency gain that some protocols (e.g., [FY92]) achieve when no fault at all occurs.

The communication complexity of a protocol is characterized by two quantities: the *message complexity* (MC, the total number of bits transmitted by all players during the protocol), and the *round complexity* (RC, the number of communication rounds of the protocol).

When analyzing the communication complexity of a multi-party protocol, one must also include the communication costs for simulating the broadcast channels. For most protocols in the literature (but not for ours), these costs are dominating the overall complexity of the protocol. We consider two different types of broadcast sub-protocols: Protocols with optimal message complexity ($\mathcal{O}(n^2)$, but $\mathcal{O}(n)$ rounds), e.g., [BGP89,CW89,DR85,HH91], and protocols with optimal round complexity ($\mathcal{O}(1)$, but $\mathcal{O}(n^4)$ messages), e.g., [FM88]. So far, no broadcast protocol with $\mathcal{O}(1)$ rounds and $\mathcal{O}(n^2)$ messages is known. In the cryptographic setting, such a protocol is known for a model where a trusted dealer is available in the set-up phase [CKS00], but this requirement contradicts the main purpose of of secure multi-party computation, namely getting rid of the need for a trusted party. There exist also various techniques which improve the efficiency of (stand-alone) protocols for Byzantine agreement, e.g. "early stopping" [DRS82]. However, they lead to "staggered termination", and it is unclear how and whether at all they are applicable for multi-party computation protocols.

4.1 Complexity of the new Protocol

The communication complexity of the proposed MPC protocol (cf. Sect. 3) is stated in the following theorem. This result is achieved by employing a Byzantine agreement protocol with optimal message complexity [BGP89,CW89].

Theorem 1. The protocol of Sect. 3 allows a set of n players, with at most $t < n/3$ of them being corrupted, to securely compute a function over a finite field $\mathbb{F}$, using $\mathcal{O}(d+n^2)$ communication rounds and with total communication complexity $\mathcal{O}(n_I n^4 + mn^3 + n_O n^2)$ field elements, where n_I and n_O denote the number of inputs and outputs, respectively, m denotes the number of multiplications and d the multiplicative depth of the circuit computing the function.

The detailed analysis of this protocol is omitted from this extended abstract. We give only a very brief overview. The VSS protocol for providing one input requires in the worst case $\mathcal{O}(n^2)$ field elements to be broadcast, which results in $\mathcal{O}(n^4)$ field elements per input when using the most efficient broadcast protocols [BGP89,CW89]. Each multiplication requires each player to secret-share (with the non-robust VSS protocol) one element, which adds up to $\mathcal{O}(n^3)$ field elements per multiplication, and hence $\mathcal{O}(m_s n^3)$ elements per segment with m_s multiplication gates. Fault-detection requires $\mathcal{O}(n)$ bits to be broadcast per segment, and fault-localization requires $\mathcal{O}(n \log m_s + \log |\mathbb{F}|)$ bits to be broadcast at the end of up to t segments. For the proposed segmentation with $m_s = \lceil m/n \rceil$ and $d_s = \lceil d/n \rceil$, at most $2n$ segments are computed, which results in a total message complexity of $\mathcal{O}(mn^3)$ field elements. Only $\mathcal{O}(n^2)$ field elements must be broadcast in total (independently of the circuit size!), which does not dominate the overall costs when $m \geq n$. The message complexity of secret reconstruction is $\mathcal{O}(n^2)$ elements per output (broadcast is not needed).

4.2 Comparison with other Protocols

The complexity of the new protocol is compared with the most efficient multiparty computation protocols for the unconditional model known before. In the sequel, we summarize the most important results. A more detailed complexity analysis can be found in [Prz99]. For simplicity we focus on the complexity of the evaluation of the circuit, and ignore the complexities of providing inputs and receiving inputs. The following table lists the message complexity (MC) and the round complexity (RC) of the most efficient protocols for the unconditional model, once when a broadcast protocol with optimal bit complexity is applied, and once when a broadcast protocol with optimal round complexity is applied. The second last row in the table refers to the protocol of [BGW88], where the "Rabin's trick" [GRR98] for simpler multiplication is used. Note that the other technique for increasing the efficiency of [BGW88] suggested in the same paper, namely the efficient proof that a shared secret is indeed the product of two shared factors, is shown to be insecure (see App. A), and hence its impact on the complexity is not analyzed.

For completeness, in Table 2 we also state the complexities of the best protocol for the cryptographic model [GRR98], in which up to $t < n/2$ of the players can be corrupted, but the security of the protocol relies on unproven assumptions. Subsequently to our work, a new protocol with cryptographic security was proposed in [CDN00], and its complexity is also listed in the table (where k denotes the security parameter). In contrast to other protocols, here the function must be specified as a Boolean circuit, and the complexity is indicated in bits.

MPC protocol	Broadcast protocol	MC	RC
[BGW88]	[FM88] [BGP89,CW89]	$\mathcal{O}(mn^8)$ $\mathcal{O}(mn^6)$	$\mathcal{O}(d)$ $\mathcal{O}(dn)$
[CCD88]	[FM88] [BGP89,CW89]	$\mathcal{O}(mn^9)$ $\mathcal{O}(mn^7)$	$\mathcal{O}(dn)$ $\mathcal{O}(dn^2)$
[Bea91a]	[FM88] [BGP89,CW89]	$\mathcal{O}(mn^8)$ $\mathcal{O}(mn^6)$	$\mathcal{O}(d)$ $\mathcal{O}(d+n)$
[FY92]	[FM88] [BGP89,CW89]	$\mathcal{O}(mn^8)$ $\mathcal{O}(mn^6)$	$\mathcal{O}(d)$ $\mathcal{O}(dn)$
[BGW88,GRR98]	[FM88] [BGP89,CW89]	$\mathcal{O}(mn^8)$ $\mathcal{O}(mn^6)$	$\mathcal{O}(d)$ $\mathcal{O}(dn)$
this paper	[BGP89,CW89]	$\mathcal{O}(mn^3)$	$\mathcal{O}(d+n^2)$

Table 1. Worst-case communication complexities of unconditional MPC protocols.

MPC protocol	Broadcast protocol	MC	RC
[GRR98]	[BGP89,CW89] [FM88]	$\mathcal{O}(mn^4)$ $\mathcal{O}(mn^6)$	$\mathcal{O}(dn)$ $\mathcal{O}(d)$
[CDN00]	[BGP89,CW89] [FM88]	$\mathcal{O}(mn^3k)$ $\mathcal{O}(mn^5k)$	$\mathcal{O}(dn)$ $\mathcal{O}(d)$

Table 2. Worst-case communication complexities of cryptographic MPC protocols.

5 Conclusions and Open Problems

General secure multi-party computation protocols for evaluating an algebraic circuit will have important applications in distributed information systems. One major reason why such protocols are not yet widely used in practical applications is their hopeless inefficiency. In particular, they all make extensive use of a reliable broadcast channel, which in any reasonable application scenario is not available, and hence must be simulated by an expensive protocol among the players.

In this paper we proposed a new framework for communication-efficient distributed protocols, applied it to secure multi-party computations, resulting in a very efficient protocol. We stress that the message complexity (and possibly the round complexity), but not the computation complexity, are the bottlenecks in most distributed applications.

There are several open problems to be solved to make general multi-party protocols applicable in distributed systems. The main issue is definitely the model: It is an open problem to generalize the framework to the asynchronous model, and to convert the used techniques accordingly. Furthermore, it might be interesting to generalize the results to non-threshold adversary structures [HM00]. Finally, it is questionable whether comparable efficiency improvements can be achieved in a model with mobile adversaries, where player elimination seems not to be applicable.

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, 1989.

[Bea91a] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology — CRYPTO '91*, vol. 576 of *LNCS*, 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 (extended abstract). In *Advances in Cryptology — CRYPTO '90*, pp. 62–76, 1990.

[BGP89] P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proc. 21st IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 410–415, 1989. Expanded version: Bit optimal distributed consensus. In *Computer Science Research*, 1992.

[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.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. 22nd ACM Symposium on the Theory of Computing (STOC)*, pp. 503–513, 1990.

[Can00] R. Canetti. Security and composition of multi-party 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*, vol. 1592 of *LNCS*, pp. 311–326, 1999.

[CDM00] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret sharing scheme. In *Advances in Cryptology — EUROCRYPT '00*, vol. 1807 of *LNCS*, pp. 316–334, 2000.

[CDN00] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. Manuscript, 2000.

[CH94] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Advances in Cryptology — CRYPTO '94*, vol. 839 of *LNCS*, pp. 425–438, 1994.

[CKS00] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 123–132, 2000.

[CW89] B. A. Coan and J. L. Welch. Modular construction of nearly optimal Byzantine agreement protocols. In *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 295–305, 1989. Expanded version: Modular construction of a Byzantine agreement protocol with optimal message bit complexity. In *Information and Computation*, 97(1):61-85, 1992.

[DFF+82] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[DR85] D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, 1985.

[DRS82] D. Dolev, R. Reischuk, and H. R. Strong. 'Eventual' is earlier than 'Immediate'. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 196–203, 1982. Final version: Early Stopping in Byzantine Agreement. In *Journal of the ACM*, 37(4):720-741, October 1990.

[FHM98] M. Fitzi, M. Hirt, and U. Maurer. Trading correctness for privacy in unconditional multi-party computation. In *Advances in Cryptology — CRYPTO '98*, vol. 1462 of *LNCS*, pp. 121–136, 1998.

[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. Expanded version in *SIAM Journal on Computing* 26(4):873–933, August 1997.

[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.

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

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th 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)*, 1998.

[HH91] V. Hadzilacos and J. Y. Halpern. Message-optimal protocols for byzantine agreement. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 309–324, 1991. Final version in *Mathematical Systems Theory*, 26:41-102, October 1993.

[HM00] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000. Extended abstract in *Proc. 16th of ACM PODC '97*.

[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 *LNCS*, pp. 392–404, Springer-Verlag, 1991.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 51–59, 1991.

[Prz99] B. Przydatek. Efficiency in multi-party computation. Master's thesis, ETH Zurich, 1999.

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

[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.

[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 Security Flaw in [GRR98]

In Appendix B ("Computing Multiplication with Faults") of [GRR98],[7] a very efficient sub-protocol was proposed for proving that for three shared values a, b, and c, the equation $c = ab$ holds. This sub-protocol was intended to replace the (rather inefficient) verification sub-protocol ("tool (II)") of [BGW88]. We show in the sequel that this new sub-protocol of [GRR98] is insecure. First we briefly summarize the protocol and then demonstrate the security flaw.

Assume that player P has shared the values a, b, and c with polynomials $f(x)$, $g(x)$, and $h(x)$ respectively, all of degree at most t. Let a_i, b_i, and c_i denote the corresponding shares of player P_i, $i = 1, \ldots, n$. The protocol of [GRR98] works as follows:

1. The dealer P shares (using "normal" secret sharing, not VSS) a random value with a polynomial $r(x)$ of degree $2t - 1$. The share r_i of player P_i is $r_i = r(\alpha_i)$. Furthermore, P computes and broadcasts the polynomial $R(x) = x \cdot r(x) + f(x) \cdot g(x) - h(x)$. $R(x)$ is a random polynomial of degree $2t$, and if $c = ab$ holds then $R(0) = 0$.
2. Every player P_i verifies that $R(0) = 0$ and $R(\alpha_i) = \alpha_i \cdot r_i + a_i \cdot b_i - c_i$. P_i broadcasts either "OK", if both checks were successful, or otherwise a request to make his values public.
3. If in the previous step some requests occurred (at most t), P broadcasts all the requested data. If there were more than t requests, P is clearly cheating.

This protocol does not guarantee correctness, in contrast to what is claimed in the paper and was believed before. The dealer P can pass this verification even if $c = ab$ does not hold:

1. Instead of selecting a random polynomial $r(x)$ of degree $2t - 1$, the dealer first selects a (random) polynomial $R(x)$ of degree $2t$ with $R(0) = 0$, then computes and distributes the "shares" $r_1, \ldots, r_n$ as $r_i = \alpha_i^{-1}(R(\alpha_i) - a_i \cdot b_i + c_i)$. The dealer can do so because the degree of the polynomial $r(x)$ cannot be verified. Finally, P broadcasts the polynomial $R(x)$.

Clearly, the checks in Step 2 of all players will succeed, and no (honest) player will complain.

[7] After the security problem was discovered, this appendix was deleted from the version available online.