MPC vs. SFE : Unconditional and Computational Security*

Martin Hirt, Ueli Maurer, and Vassilis Zikas

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
{hirt,maurer,vzikas}@inf.ethz.ch

Abstract. In secure computation among a set $\mathcal{P}$ of players one considers an adversary who can corrupt certain players. The three usually considered types of corruption are active, passive, and fail corruption. The adversary's corruption power is characterized by a so-called adversary structure which enumerates the adversary's corruption options, each option being a triple (A, E, F) of subsets of $\mathcal{P}$, where the adversary can actively corrupt the players in A, passively corrupt the players in E, and fail-corrupt the players in F.
This paper is concerned with characterizing for which adversary structures general secure function evaluation (SFE) and secure (reactive) multi-party computation (MPC) is possible, in various models. This has been achieved so far only for the very special model of perfect security, where, interestingly, the conditions for SFE and MPC are distinct. Such a separation was first observed by Ishai *et al.* in the context of computational security. We give the exact conditions for general SFE and MPC to be possible for information-theoretic security (with negligible error probability) and for computational security, assuming a broadcast channel, with and without setup. In all these settings we confirm the strict separation between SFE and MPC. As a simple consequence of our results we solve an open problem for computationally secure MPC in a threshold model with all three corruption types.

1 Introduction

Secure Function Evaluation and Secure Multi-Party Computation Secure function evaluation (SFE) allows a set $\mathcal{P} = \{p_1, \ldots, p_n\}$ of n players to compute an arbitrary agreed function f of their inputs $x_1, \ldots, x_n$ in a secure way. (Reactive) secure multi-party computation (MPC) is a generalization of SFE where the function to be computed is "reactive": players can give inputs and get outputs several times during the computation. If one models SFE and MPC as ideal functionalities, then the main difference is that in MPC (but not in SFE) the functionality must be able to keep state.

The potential dishonesty of players is modeled by a central adversary corrupting players, where players can be actively corrupted (the adversary takes full control over them), passively corrupted (the adversary can read their internal state), or fail-corrupted (the

* This research was partially supported by the Swiss National Science Foundation (SNF), project no. 200020-113700/1 and by the Zurich Information Security Center (ZISC). The full version of this paper is available at http://www.crypto.ethz.ch/pubs/HiMaZi08

J. Pieprzyk (Ed.): ASIACRYPT 2008, LNCS 5350, pp. 1–18, 2008.

adversary can make them crash at any suitable time). A crashed player stops sending any messages, but the adversary cannot read the internal state of the player (unless he is actively or passively corrupted at the same time).

Summary of Known Results SFE (and MPC) was introduced by Yao [Yao82]. The first general solutions were given by Goldreich, Micali, and Wigderson [GMW87]; these protocols are secure under some intractability assumptions. Later solutions [BGW88,CCD88] provide information-theoretic security. In particular, it is remarkable that if a (physical) broadcast channel is assumed, strictly more powerful adversaries can be tolerated [RB89,Bea91].

In the seminal papers solving the general SFE and MPC problems, the adversary is specified by a single corruption type (active or passive) and a threshold t on the tolerated number of corrupted players. Goldreich, Micali, and Wigderson [GMW87] proved that, based on cryptographic intractability assumptions, general secure MPC is possible if and only if $t < n/2$ players are actively corrupted, or, alternatively, if and only if $t < n$ players are passively corrupted. In the information-theoretic model, Ben-Or, Goldwasser, and Wigderson [BGW88] and independently Chaum, Crépeau, and Damgård [CCD88] proved that unconditional security is possible if and only if $t < n/3$ for active corruption and $t < n/2$ for passive corruption. Finally, in [GMW87,GL02,Gol04] it was shown that, based on cryptographic intractability assumptions, any number of active cheaters ($t < n$) can be tolerated for SFE, but only if we sacrifice fairness and guaranteed delivery of the output [Cle86]. Some of the above results were unified, and extended to include fail-corruption, in [FHM98]: perfectly secure MPC (and SFE) is achievable if and only if $3t_a + 2t_p + t_f < n$, and unconditionally secure MPC (SFE) (without a trusted setup or a broadcast channel) is achievable if and only if $2t_a + 2t_p + t_f < n$ and $3t_a + t_f < n$, where t_a, t_p, and t_f denote the upper bounds on the number of actively, passively, and fail-corrupted players, respectively. These results consider an adversary who can perform all three corruption types *simultaneously*. For the computational-security case, Ishai *et al.* [IKLP06] gave a protocol for SFE which tolerates an adversary who can either corrupt $t_a < n/2$ players actively, or, *alternatively*, $t_p < n$ players passively. They also showed that such an adversary cannot be tolerated for MPC.

Generalizing threshold models, the adversary's corruption power can be characterized by a so-called adversary structure which enumerates the adversary's corruption options, each option being a triple (A, E, F) of subsets of $\mathcal{P}$, where the adversary can actively corrupt the players in A, passively corrupt the players in E, and fail-corrupt the players in F. Of course, the adversary's choice of the option is secret and a protocol must tolerate any choice by the adversary.

General adversary structures were first considered in [HM97,HM00] for active-only and passive-only corruption. General mixed-corruption (active and passive) adversary structures were considered in [FHM99]. The full generality, including fail-corruption, was first considered in [BFH$^+$08], where only the perfect-security case could be solved, both for SFE and MPC. An interesting aspect of those results is the separation between SFE and MPC: the condition for SFE is strictly weaker than the condition for MPC. This can also be seen as a justification for the most general mixed corruption models. Such a separation was previously observed for the perfect-security case [Alt99] and, as already mentioned, for the computational-security case [IKLP06].

Contributions of this Paper We prove the exact conditions for general SFE and MPC to be possible, in the most general mixed adversary model, with synchronous communication, and where a broadcast channel is assumed. We consider the most natural and desirable security notion, where full security (including fairness and guaranteed output delivery) is required. We solve the two cases of general interest: unconditional (information-theoretic with negligible error probability) security and computational security, both with and without setup. We show a strict separation between SFE and MPC.

Our results imply that for the threshold model with all three corruption types simultaneously, and for computational security, SFE and MPC are possible if and only if $2t_a + t_p + t_f < n$. As in [FHM98] there is no separation in this model.

Outline of this paper In Section 2 we describe the model. In Sections 3,4,5 and 6 we handle the unconditional-security case; in particular, in Sections 3 and 4 we describe techniques and sub-protocols that are used for the construction of MPC and SFE protocols described in Sections 5 and 6, respectively. Finally, in Section 7 we handle the computational-security case.

2 The Model

We consider a set $\mathcal{P} = \{p_1, \ldots, p_n\}$ of players. Some of these players can be corrupted by the adversary. We consider active corruption (the adversary takes full control), passive corruption (the adversary can read the internal state), and fail-corruption (the adversary can make the player crash). We use the following characterizations for players: a player that is not corrupted is called *uncorrupted*, a player that (so far) has followed the protocol instructions is called *correct*, and a player that has deviated from the protocol (e.g., has crashed or has sent wrong messages) is called *incorrect*. The adversary's corruption capability is characterized by an adversary structure $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$ (for some m) which is a monotone set of triples of player sets. At the beginning of the protocol, the adversary chooses a triple $Z^\star = (A^\star, E^\star, F^\star) \in \mathcal{Z}$ and actively corrupts the players in $A^\star$, passively corrupts the players in $E^\star$ (eavesdropping), and fail-corrupts the players in $F^\star$;[1] this triple is called the *actual adversary class* or simply the actual adversary. Note that $Z^\star$ is not known to the honest players and appears only in the security analysis. A protocol is called $\mathcal{Z}$*-secure* if it is secure against an adversary with corruption power characterized by $\mathcal{Z}$. For notational simplicity we assume that $A \subseteq E$ and $A \subseteq F$ for any $(A, E, F) \in \mathcal{Z}$, since an actively corrupted player can behave as being passively or fail-corrupted. Furthermore, as many constructions only need to consider the maximal classes of a structure, we define the maximal structure $\overline{\mathcal{Z}}$ as the smallest subset of $\mathcal{Z}$ such that $\forall (A, E, F) \in \mathcal{Z}\ \exists (\bar{A}, \bar{E}, \bar{F}) \in \overline{\mathcal{Z}} :\ A \subseteq \bar{A}, E \subseteq \bar{E}, F \subseteq \bar{F}$.

Communication takes place over a complete network of secure channels. Furthermore, we assume authenticated broadcast channels, which allow every $p_i \in \mathcal{P}$ to consistently send an authenticated message to all players in $\mathcal{P}$. All communication is synchronous, i.e., the delays in the network are upper-bounded by a known constant.

In the computational model (Section 7), the secrecy of the bilateral channels can be implemented by using encryption, where the public keys are distributed using the

[1] We focus on static security, although our results could be generalized to adaptive corruption.

authenticated broadcast channels. We mention that in a model with simultaneous active and passive corruption, the authenticity cannot easily be implemented using setup, as the adversary can forge signatures of passively corrupted players. Also implementing the authenticated broadcast channels by point-to-point communication seems non-trivial, as it must be guaranteed that fail-corrupted players send either the right value or no value (but not a wrong value), and that passively corrupted players always send the right value.

To simplify the description, we adopt the following convention: Whenever a player does not receive an expected message (over a bilateral or a broadcast channel), or receives a message outside of the expected range, then the special symbol $\perp \notin \mathbb{F}$ is taken for this message. Note that after a player has crashed, he only sends $\perp$.

The function to be computed is described as an arithmetic circuit over some finite field $\mathbb{F}$, consisting of addition (or linear) gates and multiplication gates. Our protocols take as input the player's inputs and additionally the maximal adversary structure. The running time of the suggested protocols is polynomial in the size of their input,[2] and the error probability is negligible.

3 Information Checking

An actively corrupted player might send a value to another player and then deny that the value was sent by him. To deal with such behavior, we need a mechanism which binds a player to the messages he sends. In [RB89,CDD+99,BHR07] the Information Checking (IC) method was developed for this purpose, and used to design unconditionally secure protocols tolerating up to $t < n/2$ active cheaters. In this section, we extend the IC method to the setting of general adversaries with active, passive, and fail-corruption.

The IC-authentication scheme involves three players, a sender p_s, a recipient p_r, and a verifier p_v, and consists of three protocols, called IC-Setup, IC-Distr, and IC-Reveal. Protocol IC-Distr allows p_s to send a value v to p_r in an authenticated way, so that p_r can, by invoking IC-Reveal, open v to p_v and prove that v was received from p_s. Both IC-Distr and IC-Reveal assume a secret key α known exclusively to p_s and p_v (but not to p_r). This key is generated and distributed in IC-Setup. Note that the same key can be used to authenticate multiple messages.

Informally, the three protocols can be described as follows: In IC-Setup, p_s generates a uniformly random key α and sends it to p_v over the bilaterally secure channel. In IC-Distr, α is used to generate an authentication tag y and a verification tag z for the sent value v. The values (v, y) and z are given to p_r and p_v, respectively. In IC-Reveal, p_r sends (v, y) to p_v, who verifies that (y, z) is a valid authentication/verification-tag pair for v with key α.

Ideally, an IC-authentication scheme should have the following properties: (1) Any value sent with IC-Distr is accepted in IC-Reveal, (2) in IC-Distr, p_v gets no information on v, and (3) only values sent with IC-Distr are accepted in IC-Reveal. However, these properties cannot be (simultaneously) perfectly satisfied. In fact, Property 3 can only be

[2] As the adversary structure might be exponentially large, our protocols' worst case running time can be exponential in the size of the player set. However, this is the best complexity one can hope to achieve for a protocol that tolerates *any* adversary structure [HM00].

achieved with negligible error probability, as the adversary might guess an authentication tag y' for a $v' \neq v$. Moreover, it can only be achieved when neither p_s nor p_v is passively corrupted, since otherwise the adversary knows α and z.

In our IC-authentication scheme the key α is chosen uniformly at random from $\mathbb{F}$ and the value v is also from $\mathbb{F}$. The authentication and verification tags, y and z, respectively, are such that for some degree-one polynomial $w(\cdot)$ over $\mathbb{F}$, $w(0) = v$, $w(1) = y$, and $w(\alpha) = z$. In other words, (y, z) is a valid IC-pair if $z = (y - v)\alpha + v$. Defining validity this way gives the IC-authentication scheme an additional *linearity* property. In particular, if (y, z) and (y', z') are valid IC-pairs for v and v', respectively, (for the same α) then $(y + y', z + z')$ is a valid IC-pair for $v + v'$. This implies that when some values have been sent with IC-Distr, then p_r and p_v can, without any interaction, compute valid authentication data for any linear combination of those values.

Due to space restrictions, the detailed description of the protocols IC-Setup, IC-Distr, and IC-Reveal, as well as the proof of the following lemma are deleted from this extended abstract.

Theorem 1. *Our IC-authentication scheme has the following properties. Correctness: When* IC-Distr *succeeds p_r learns a value v', where $v' = v$ unless p_s is actively corrupted.* IC-Distr *might abort only when p_s is incorrect. Completeness: If* IC-Distr *succeeds and p_r is correct then in* IC-Reveal *p_v accepts v'. Privacy:* IC-Distr *leaks no information on v to any player other than p_r. Unforgeability: When neither p_s nor p_v is passively corrupted, and the protocols* IC-Distr *and* IC-Reveal *have been invoked at most polynomially many times, then the probability that an adversary actively corrupting p_r makes p_v accept some v' which was not sent with* IC-Distr *is negligible.*

General IC-signatures An IC-authentication scheme allows a sender $p_i \in \mathcal{P}$ to send a value v to a recipient $p_j \in \mathcal{P}$, so that p_j can later prove authenticity of v, but *only* towards a dedicated verifier $p_k \in \mathcal{P}$. In our protocols we want to use IC-authentication as a mechanism to bind the sender p_i to the messages he sends to p_j, so that p_j can prove to *every* $p_k \in \mathcal{P}$ that these messages originate from p_i. In [CDD+99], the IC-signatures where introduced for this purpose. These can be seen as semi "digital signatures" with information theoretic security. They do not achieve all properties of digital signatures, but enough to guarantee the security of our protocols.

The protocols used for generation and verification of IC-signatures are called ICS-Sign and ICS-Open, respectively. ICS-Sign allows a player $p_i \in \mathcal{P}$ to send a value v to $p_j \in \mathcal{P}$ signed with an IC-signature. The idea is the following: for each $p_k \in \mathcal{P}$, p_i invokes IC-Distr to send v to p_j with p_k being the verifier, where p_j checks that he receives the same v in all invocations. As syntactic sugar, we denote the resulting IC-signature by $\sigma_{i,j}(v)$. The idea in ICS-Open is the following: p_j announces v and invokes IC-Reveal once for each $p_k \in \mathcal{P}$ being the verifier. Depending on the outcomes of IC-Reveal the players decide to accept or reject v. As we want every $p_i \in \mathcal{P}$ to be able to send messages with ICS-Sign, we need a secret-key setup, where every $p_i, p_k \in \mathcal{P}$ hold a secret key $\alpha_{i,k}$. Such a setup can be easily established by appropriate invocations of IC-Setup.

The decision to accept or reject in ICS-Open has to be taken in a way which ensures that valid signatures are accepted (completeness), and forged signatures are rejected with

overwhelming probability (unforgeability). To guarantee completeness, a signature must not be rejected when only actively corrupted players rejected in IC-Reveal. Hence, the players cannot reject the signature when there exists a class $(A_j, E_j, F_j) \in \mathcal{Z}$ such that all rejecting players are in A_j. Along the same lines, to guarantee unforgeability, the players cannot accept the signature when there exists a class $(A_i, E_i, F_i) \in \mathcal{Z}$ such that all accepting players are in E_i. To make sure that the above two cases cannot simultaneously occur, we require $\mathcal{Z}$ to satisfy the following property, denoted as $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$:

$$C_{\text{IC}}(\mathcal{P}, \mathcal{Z}) \iff \forall (A_i, E_i, F_i), (A_j, E_j, F_j) \in \mathcal{Z} : \; E_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P}$$

We refer to the full version of this paper for a detailed description of the protocols ICS-Sign and ICS-Open and for a proof of the following lemma.

Lemma 1. *Assuming that $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$ holds, our IC-signatures scheme has the following properties. Correctness: When* ICS-Sign *succeeds, then p_r learns a value v', where $v' = v$ unless p_s is actively corrupted.* ICS-Sign *might abort only when p_s is incorrect. Completeness: If* ICS-Sign *succeeds and p_r is correct then in* ICS-Open *all players accept v'. Privacy:* ICS-Sign *leaks no information on v to any player other than p_r. Unforgeability: When p_s is not passively corrupted, and the protocols* ICS-Sign *and* ICS-Open *have been invoked at most polynomially many times, then the probability that an adversary actively corrupting p_j can make the players accept some v' which was not sent with* ICS-Sign *is negligible.*

Linearity of IC-signatures The linearity property of the IC-authentication scheme is propagated to the IC-signatures. In particular, when some values have be sent by p_i to p_j with ICS-Sign (using the same secret keys), then the players can locally, i.e., without any interaction, compute p_i's signature for any linear combination of those values, by applying the appropriate linear combination on the respective signatures. This process yields a signature which, when p_j is correct, will be accepted in ICS-Open.

4 Tools - Subprotocols

In this section we describe sub-protocols that are used as building blocks for MPC and SFE protocols. Some of the sub-protocols are non-robust, i.e., they might abort. When they abort then all (correct) players agree on a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. The sub-protocols use IC-signatures to authenticate the sent values, therefore their security relies on the security of the IC-signatures. In particular, the security of the sub-protocols is guaranteed only when no signature is forged.[3] The secret-key setup, which is required for the IC-signatures, is established in a setup phase, before any of the sub-protocols is invoked. Due to space restrictions the security proofs and even the detailed descriptions of some of the sub-protocols are deleted from this extended abstract.

4.1 Share and Reconstruct

A secret-sharing scheme allows a player (called the dealer) to distribute a secret so that only qualified sets of players can reconstruct it. As secret-sharing scheme we employ

[3] We use the term "forge" only for signatures corresponding to non-passively corrupted signers.

a sum-sharing, i.e., the secret is split into summands that add up to the secret, where each summand might be given to several players. Additionally, for each summand all the players who hold it bilaterally exchange signatures on it. The sharing is characterized by a vector $\mathcal{S} = (S_1, \ldots, S_m)$ of subsets of $\mathcal{P}$, called the *sharing specification*. A value s is shared according to $\mathcal{S}$ if there exist summands $s_1, \ldots, s_m \in \mathbb{F}$ such that $\sum_{k=1}^{m} s_k = s$, and for each $k = 1, \ldots, m$ every $p_j \in S_k$ holds s_k along with IC-signatures on it from every $p_i \in S_k$. As syntactic sugar, we denote by $\sigma_{\mathcal{S}}(s)$ the set of all IC-signatures on the summands $s_1, \ldots, s_m$ held by the players. For each $p_j \in \mathcal{P}$ the vector $\langle s \rangle_j = (s_{j_1}, \ldots, s_{j_\ell})$ is considered to be p_j's *share* of s, where $s_{j_1}, \ldots, s_{j_\ell}$ are the summands held by p_j. The vector of all shares and the attached signatures, denoted as $\langle s \rangle = \big(\langle s \rangle_1, \ldots, \langle s \rangle_n, \sigma_{\mathcal{S}}(s)\big)$, is a *sharing* of s. The vector of summands in $\langle s \rangle$ is denoted as $[s] = (s_1, \ldots, s_m)$. We say that $\langle s \rangle$ is a *consistent* sharing of s according to $\mathcal{S}$ if for each $k = 1, \ldots, m$ all (correct) players in S_k have the same view on the summands s_k and hold signatures on it from all other players in S_k, and $\sum_{k=1}^{m} s_k = s$.

For an adversary structure $\mathcal{Z}$, we say that a sharing specification $\mathcal{S}$ is $\mathcal{Z}$*-private* if for any sharing $\langle s \rangle$ according to $\mathcal{S}$ and for any adversary in $\mathcal{Z}$, there exists a summand s_k which this adversary does not know. Formally, $\mathcal{S}$ is $\mathcal{Z}$-private if $\forall (A, E, F) \in \mathcal{Z}\ \exists S \in \mathcal{S} : \ S \cap E = \emptyset$.[4] For an adversary structure $\mathcal{Z}$ with maximal classes $\overline{\mathcal{Z}} = \big\{(\cdot, E_1, \cdot), \ldots, (\cdot, E_m, \cdot)\big\}$, we denote the natural $\mathcal{Z}$-private sharing specification by $S_{\mathcal{Z}} = \big(\mathcal{P} \backslash E_1, \ldots, \mathcal{P} \backslash E_m\big)$.

Protocol Share (see below) allows a dealer p to share a value s among the players in $\mathcal{P}$ according to a sharing specification $\mathcal{S}$. The protocol is non-robust and might abort with a set $B \subseteq \mathcal{P}$ of incorrect players.

Protocol Share$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p, s)$

1. Dealer p chooses summands $s_2, \ldots, s_{|\mathcal{S}|}$ randomly and sets $s_1 := s - \sum_{k=2}^{|\mathcal{S}|} s_k$.
2. For $k = 1, \ldots, |\mathcal{S}|$ the following steps are executed:
   (a) p sends s_k to each $p_j \in S_k$.
   (b) For each $p_i, p_j \in S_k$: ICS-Sign$(\mathcal{P}, \mathcal{Z}, p_i, p_j, s_k)$ is invoked to have p_i send s_k to p_j and attach an IC-signature on it. If ICS-Sign aborts, then Share aborts with $B := \{p_i\}$.
   (c) Each $p_j \in S_k$ broadcasts a complaint bit b, where $b = 1$ if p_j received a $\perp$ instead of s_k in Step 2a, or if he received some $s'_k \neq s_k$ from some p_i in Step 2b, and $b = 0$ otherwise.
   (d) If a complaint was reported p broadcasts s_k and the players in S_k create default signatures on it. If p broadcasts $\perp$ then Share aborts with set $B := \{p\}$.

Lemma 2. *If $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, then protocol* Share$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p, s)$ *has the following properties. Correctness: It either outputs a consistent sharing of s' according to $\mathcal{S}$, where $s' = s$ unless the dealer p is actively corrupted, or it aborts with*

[4] Recall that for all $(A, E, F) \in \mathcal{Z} : \ A \subseteq E$.

a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about s leaks to the adversary.

Reconstructing a shared value s is straightforward: The summands are announced one by one, and s is computed as the sum of the announced summands. To announce a summand s_k, each $p_i \in S_k$ broadcasts s_k and opens all the signatures on s_k which he holds (i.e., the signatures on s_k from all players in S_k). If all the signatures announced by p_i are accepted, then the value he announced is taken for s_k. If no $p_i \in S_k$ correctly announces all the signatures the announcing aborts with $B := S_k$. Protocols PubAnnounce and PubReconstruct invoked to publicly announce a summand and to publicly reconstruct a shared value are given in details in the full version of this paper. In the following two lemmas (also proved in the full version) we state their security.

Lemma 3. *Assume that $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$ holds, the condition $\forall (A, E, F) \in \mathcal{Z} : S_k \not\subseteq E$ holds, and no signature is forged. Then protocol* PubAnnounce *either publicly announces the correct summand s_k, or it aborts with a non-empty set B of incorrect players. It might abort only if $S_k \subseteq F^\star$.*

Lemma 4. *Assume that $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$ holds, the condition $\forall S \in \mathcal{S}$, $\forall (A, E, F) \in \mathcal{Z} : S \not\subseteq E$ holds, $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$, and no signature is forged. Then protocol* PubReconstruct *either publicly reconstructs s, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players.*

Protocol PubReconstruct allows for public reconstruction of a shared value. However, in some of our protocols we need to reconstruct a shared value s privately, i.e., only towards some dedicated output player p. Such a private reconstruction protocol can be built using standard techniques (p shares a one-time pad used for perfectly blinding the output). We refer to the protocol for private reconstruction as Reconstruct, and point to the full version of this paper for a detailed description as well as for a proof of the following lemma.

Lemma 5. *Assume that $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, the condition $\forall S \in \mathcal{S}$, $\forall (\cdot, E, \cdot) \in \mathcal{Z} : S \not\subseteq E$ holds, $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$, and no signature is forged. Then protocol* Reconstruct$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p, \langle s \rangle)$ *has the following properties. Correctness: Either it reconstructs s towards p, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about $\langle s \rangle$ leaks to the adversary.*

Addition Due to the linearity of our secret sharing scheme, the players can locally compute a sharing of the sum of two shared values s and t as follows: each player adds his shares of s and t, and the corresponding signatures are also (locally) added. We refer to this sub-protocol as Add.

4.2 Multiplication

The goal of this section is to design a protocol for securely computing a sharing of the product of two shared values. Our approach combines techniques from [GRR98,Mau02,Mau06,BFH+08].

At a high level, the multiplication protocol for two shared values s and t works as follows: As s and t are already shared, we can use the summands $s_1, \ldots, s_m$ and $t_1, \ldots, t_m$ to compute the product as $st = \sum_{k,\ell=1}^{m} s_k t_\ell$. For each term $x_{k,\ell} = s_k t_\ell$, we have a player $p^{(k,\ell)} \in (S_k \cap S_\ell)$ share $x_{k,\ell}$ and prove that he shared the correct value. The sharing of st is computed as the sum of the sharings of the terms $x_{k,\ell}$.

For $p^{(k,\ell)} \in (S_k \cap S_\ell)$ to share $s_k t_\ell$ and prove that he did so properly the idea is the following: First, $p^{(k,\ell)}$ shares $s_k t_\ell$ by invoking Share. Denote by $x'_{k,\ell}$ the shared value.[5] Next, $p^{(k,\ell)}$ shares the summands s_k and t_ℓ by a protocol, called SumShare, which guarantees that he shares the correct summands. Finally, $p^{(k,\ell)}$ uses the sharings of s_k, t_ℓ, and $x'_{k,\ell}$ in a protocol, called MultProof, which allows him to prove that $x'_{k,\ell} = s_k t_\ell$. In the following we discuss the sub-protocols SumShare and MultProof, and then give a detailed description of the multiplication protocol.

Protocol SumShare (see full version) allows a player $p \in S_k$ to share a summand s_k of a sharing $\langle s \rangle$ according to $\mathcal{S}$, where $S_k \in \mathcal{S}$. The sharing specification of the output sharing can be some $\mathcal{S}' \neq \mathcal{S}$. In contrast to Share, protocol SumShare guarantees that p_i shares the correct value s_k. The idea is to have p share s_k, by Share, and then reconstruct the sharing (privately) towards each $p_j \in S_k$ who publicly approves or disapproves it. We refer to the full version of this paper for a proof of the following lemma.

Lemma 6. *Assume that $C_{\mathrm{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}'$ is a $\mathcal{Z}$-private sharing specification, the conditions $\forall(\cdot, E, \cdot) \in \mathcal{Z} : S_k \not\subseteq E$ and $\forall S' \in \mathcal{S}' \, \forall(\cdot, E, \cdot) \in \mathcal{Z} : S' \not\subseteq E$ hold, and no signature is forged. Then* SumShare$(\mathcal{P}, \mathcal{Z}, \mathcal{S}', S_k, p, s_k)$ *has the following properties. Correctness: Either it outputs a consistent sharing of s_k (p also outputs the vector $[s_k]$ of summands) according to $\mathcal{S}'$, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about s_k leaks to the adversary.*

Protocol MultProof (see full version) allows a player p, called the prover, who has shared three values a, b, and c (and knows the corresponding vectors $[a], [b]$, and $[c]$ of summands) to prove that $c = ab$. The protocol can be seen as a distributed challenge-response protocol with prover p and verifier being all the players in $\mathcal{P}$. On a high level, it can be described as follows: First p shares some appropriately chosen values. Then the players jointly generate a uniformly random challenge r and expose it, and p answers the challenge. If p's answer is consistent with the sharings of a, b, and c and the sharings which he created in the first step, then the proof is accepted otherwise it is rejected. MultProof is non-robust and might abort with a set $B \subseteq \mathcal{P}$ of incorrect players. The proof of the following lemma is deleted from this extended abstract.

Lemma 7. *Assume that $C_{\mathrm{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, the condition $\forall S \in \mathcal{S}, \ \forall(\cdot, E, \cdot) \in \mathcal{Z} : S \not\subseteq E$ holds, $\langle a \rangle, \langle b \rangle$, and $\langle c \rangle$ are consistent sharings according to $\mathcal{S}$, and no signature is forged. Then the protocol* MultProof *has the following properties. Correctness: If $c = ab$, then either the proof is accepted or* MultProof *aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Otherwise (i.e, if $c \neq ab$), with overwhelming probability, either the proof is rejected or* MultProof *aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about $\langle a \rangle, \langle b \rangle$, and $\langle c \rangle$ leaks to the adversary.*

[5] Note that Share does not guarantee that $x'_{k,\ell} = s_k t_\ell$.

For completeness, we describe the multiplication protocol Mult (see next page), which allows to compute a sharing of the product of two shared values. Mult is non-robust and might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. When it succeeds, then with overwhelming probability it outputs a consistent sharing of the product.

Protocol $\mathsf{Mult}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$

1. For every $(S_k, S_\ell) \in \mathcal{S} \times \mathcal{S}$, the following steps are executed, where $p^{(k,\ell)}$ denotes the player in $S_k \cap S_\ell$ with the smallest index:
   (a) $p^{(k,\ell)}$ computes $x_{k,\ell} := s_k t_\ell$ and shares it, by Share. Denote by $\langle x_{k,\ell} \rangle$ the resulting sharing.[a]
   (b) $\mathsf{SumShare}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, S_k, p^{(k,\ell)}, s_k)$ and $\mathsf{SumShare}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, S_\ell, p^{(k,\ell)}, t_\ell)$ are invoked. Denote by $\langle s_k \rangle$ and $\langle t_\ell \rangle$ the resulting sharings.
   (c) $\mathsf{MultProof}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p^{(k,\ell)}, \langle s_k \rangle, \langle t_\ell \rangle, \langle x_{k,\ell} \rangle)$ is invoked. If the proof is rejected then Mult aborts with set $B = \{p^{(k,\ell)}\}$.
2. A sharing of the product st is computed as the sum of the sharings $\langle x_{k,\ell} \rangle$ by repeatedly invoking Add.
3. If any of the invoked sub-protocols aborts with B, then also Mult aborts with B.

[a] In addition to his share of $\langle x_{k,\ell} \rangle$, $p^{(k,\ell)}$ also outputs the vector of summands $[x_{k,\ell}]$.

Lemma 8. *Assume that $C_{\mathrm{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, the conditions $\forall S \in \mathcal{S}, \forall(\cdot, E, \cdot) \in \mathcal{Z} : S \not\subseteq E$ and $\forall S_k, S_\ell \in \mathcal{S} : S_k \cap S_\ell \neq \emptyset$ hold, $\langle s \rangle$ and $\langle t \rangle$ are consistent sharings according to $\mathcal{S}$, and no signature is forged. Then protocol $\mathsf{Mult}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$ has the following properties except with negligible probability. Correctness: It either outputs a consistent sharing of st according to $\mathcal{S}$ or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about $\langle s \rangle$ and $\langle t \rangle$ leaks to the adversary.*

4.3 Resharing

In the context of MPC, we will need to reshare shared values according to a different sharing specification. To do that, each summand is shared by SumShare (see Section 4.2) according to the new sharing specification, and the players distributively add the sharings of the summands, resulting in a new sharing of the original value. A detailed description of the protocol Reshare as well as a proof of the following lemma can be found in the full version of this paper.

Lemma 9. *Assume that $C_{\mathrm{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}'$ is a $\mathcal{Z}$-private sharing specification, the conditions $\forall S \in \mathcal{S} \forall(\cdot, E, \cdot) \in \mathcal{Z} : S \not\subseteq E$, and $\forall S' \in \mathcal{S}' \forall(\cdot, E, \cdot) \in \mathcal{Z} : S' \not\subseteq E$ hold, and no signature is forged. Then $\mathsf{Reshare}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{S}', \langle s \rangle)$ has the following properties. Correctness: Either it outputs a consistent sharing of s according to $\mathcal{S}'$, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy: No information about $\langle s \rangle$ leaks to the adversary.*

5 (Reactive) Multi-Party Computation

In this section we prove the necessary and sufficient condition on the adversary structure $\mathcal{Z}$ for the existence of unconditionally (i.e., i.t. with negligible error probability) $\mathcal{Z}$-secure multi-party computation protocols, namely, we prove the following theorem:

Theorem 2. *A set $\mathcal{P}$ of players can unconditionally $\mathcal{Z}$-securely compute any (reactive) computation, if and only if $C^{(2)}(\mathcal{P},\mathcal{Z})$ and $C^{(1)}(\mathcal{P},\mathcal{Z})$ hold, where*

$$C^{(2)}(\mathcal{P},\mathcal{Z}) \iff \forall (A_i,E_i,F_i),(A_j,E_j,F_j):\ E_i \cup E_j \cup (F_i \cap F_j) \neq \mathcal{P}$$
$$C^{(1)}(\mathcal{P},\mathcal{Z}) \iff \forall (A_i,E_i,F_i),(A_j,E_j,F_j):\ E_i \cup F_j \neq \mathcal{P}$$

The sufficiency of the above condition is proved by constructing an MPC protocol for any given circuit C consisting of input, addition, multiplication, and output gates.[6] The reactiveness of the computation is modeled by assigning to each gate a point in time when it should be evaluated.

The circuit is evaluated in a gate-by-gate fashion, where for input, addition, multiplication, and output gates, the corresponding sub-protocol Share, Add, Mult, and Reconstruct, respectively, is invoked.

The computation starts off with the initial player set $\mathcal{P}$ and adversary structure $\mathcal{Z}$, and with the sharing specification being $\mathcal{S} := \mathcal{S}_{\mathcal{Z}}$. Each time a sub-protocol aborts with set B of incorrect players, the players in B are deleted from the player set and from every set in the sharing specification, and the corresponding gate is repeated. Any future invocation of a sub-protocol is done in the updated player set $\mathcal{P}'$ and sharing specification $\mathcal{S}'$, and with the updated adversary structure $\mathcal{Z}'$, which contains only the classes in $\mathcal{Z}$ compatible with the players in $\mathcal{P} \setminus \mathcal{P}'$ being incorrect. Note that, as the players in $\mathcal{P} \setminus \mathcal{P}'$ are incorrect, any sharing according to $(\mathcal{P},\mathcal{S})$ can be transformed, without any interaction, to a sharing according to $(\mathcal{P}',\mathcal{S}')$ by having the players delete all signatures of signers from $\mathcal{P} \setminus \mathcal{P}'$.

The delicate task is the multiplication of two shared values s and t. The idea is the following: First, we invoke Reshare to have both s and t shared according to the sharing specification $\mathcal{S}_{\mathcal{Z}'}$, i.e., the specification associated with the structure $\mathcal{Z}'$. Then we invoke Mult to compute a sharing of the product st according to $\mathcal{S}_{\mathcal{Z}'}$, and at the end we invoke Reshare once again to have the product shared back to the initial setting (i.e, according to $(\mathcal{P}',\mathcal{S}')$).

The security of the computation is guaranteed as long as no signature is forged. We argue that the forging probability is negligible. Observe that the total number of signatures in each sub-protocol invocation is polynomial in the input size; also, the total number of sub-protocol invocations is polynomial in the size of the circuit (since each time a sub-protocol aborts a new set B of incorrect players is identified, the total number of abortions is bounded by n). Hence, the total number of signatures in the computation is polynomial and, by the unforgeability property, the probability that a signature is forged is negligible.

We use the following operators on adversary structures, which were introduced in [BFH+08]: For a set $B \subseteq \mathcal{P}$, we denote by $\mathcal{Z}|^{B \subseteq F}$ the sub-structure of $\mathcal{Z}$ that contains

[6] This does not exclude probabilistic circuits, as a random gate can be simulated by having each player input a random value and take the sum of those values as the input.

only adversaries who can fail-corrupt all the players in B, i.e., $\mathcal{Z}|^{B\subseteq F} = \{(A,E,F) \in \mathcal{Z} : B \subseteq F\}$. Furthermore, for a set $\mathcal{P}' \subseteq \mathcal{P}$, we denote by $\mathcal{Z}|_{\mathcal{P}'}$ the adversary structure with all classes in $\mathcal{Z}$ restricted to the player set $\mathcal{P}'$, i.e., $\mathcal{Z}|_{\mathcal{P}'} = \{(A\cap\mathcal{P}', E\cap\mathcal{P}', F\cap\mathcal{P}') : (A,E,F) \in \mathcal{Z}\}$. We also use the same operator on sharing specifications with similar semantics, i.e., for $\mathcal{S} = (S_1,\ldots,S_m)$ we denote $\mathcal{S}|_{\mathcal{P}'} = (S_1 \cap \mathcal{P}', \ldots, S_m \cap \mathcal{P}')$. As syntactic sugar, we write $\mathcal{Z}|_{\mathcal{P}'}^{B\subseteq F}$ for $(\mathcal{Z}|^{B\subseteq F})\,|_{\mathcal{P}'}$.

It follows from the above definitions that when the players in $\mathcal{P} \setminus \mathcal{P}'$ have been detected to be incorrect, then the actual adversary $Z^\star$ is in $\mathcal{Z}|^{\mathcal{P}\setminus\mathcal{P}'\subseteq F}$. Furthermore, as the updated player set is $\mathcal{P}'$, the corresponding sharing specification and adversary structure are $\mathcal{S}' = \mathcal{S}|_{\mathcal{P}'}$ and $\mathcal{Z}' = \mathcal{Z}|_{\mathcal{P}'}^{\mathcal{P}\setminus\mathcal{P}'\subseteq F}$, respectively. One can easily verify that the conditions $C^{(\mathbf{2})}$ and $C^{(\mathbf{1})}$ hold in $(\mathcal{P}', \mathcal{Z}')$ when they hold in $(\mathcal{P}, \mathcal{Z})$. This results in protocol MPC (see below).

Protocol $\mathsf{MPC}(\mathcal{P}, \mathcal{Z}, C)$

0. Initialize $\mathcal{P}' := \mathcal{P}$, $\mathcal{Z}' := \mathcal{Z}$, and $\mathcal{S}' := \mathcal{S}_{\mathcal{Z}}$.
1. For every gate to be evaluated, do the following:
   - *Input gate for p:* If $p \in \mathcal{P}'$ invoke Share to have p share his input according to $(\mathcal{P}', \mathcal{S}')$. Otherwise, a default sharing of some pre-agreed default value is taken as the sharing of p's input.
   - *Addition gate:* Invoke Add to compute a sharing of the sum according to $\mathcal{S}'$.
   - *Multiplication gate:* Denote the sharings of the factors as $\langle s\rangle$ and $\langle t\rangle$, respectively, and the sharing specification corresponding to $\mathcal{Z}'$ as $\mathcal{S}_{\mathcal{Z}'}$. Invoke $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}', \mathcal{S}_{\mathcal{Z}'}, \langle s\rangle)$ and $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}', \mathcal{S}_{\mathcal{Z}'}, \langle t\rangle)$ to obtain the sharings $\langle s\rangle'$ and $\langle t\rangle'$ according to $(\mathcal{P}', \mathcal{S}_{\mathcal{Z}'})$, respectively. Invoke $\mathsf{Mult}(\mathcal{P}', \mathcal{Z}', \mathcal{S}_{\mathcal{Z}'}, \langle s\rangle', \langle t\rangle')$ to obtain a sharing $\langle st\rangle'$ of the product, according to $(\mathcal{P}', \mathcal{S}_{\mathcal{Z}'})$. Invoke $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}_{\mathcal{Z}'}, \mathcal{S}', \langle st\rangle')$ to reshare this product according to $(\mathcal{P}', \mathcal{S}')$.
   - *Output gate for p:* If $p \in \mathcal{P}'$ invoke Reconstruct to have the output reconstructed towards p.
2. If any of the sub-protocols aborts with set B, then update $\mathcal{P}' := \mathcal{P}' \setminus B$, set $\mathcal{S}' := \mathcal{S}'|_{\mathcal{P}'}$ and $\mathcal{Z}' := \mathcal{Z}|_{\mathcal{P}'}^{\mathcal{P}\setminus\mathcal{P}'\subseteq F}$ and repeat the corresponding gate.

Lemma 10. *The protocol* MPC *is unconditionally $\mathcal{Z}$-secure if $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ and $C^{(\mathbf{1})}(\mathcal{P}, \mathcal{Z})$ hold.*

To complete this section, we give two lemmas that imply that unconditionally secure (reactive) MPC is not possible for some circuits when $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ or $C^{(\mathbf{1})}(\mathcal{P}, \mathcal{Z})$ is violated. The proofs of the lemmas are deleted from this extended abstract.

Lemma 11. *If $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ is violated then there exist (even non-reactive) circuits which cannot be evaluated unconditionally $\mathcal{Z}$-securely.*

Lemma 12. *If $C^{(\mathbf{1})}(\mathcal{P}, \mathcal{Z})$ is violated, then the players cannot hold a secret joint state with unconditional security.*

6 Secure Function Evaluation

In this section we prove the necessary and sufficient condition on the adversary structure $\mathcal{Z}$ for the existence of unconditionally $\mathcal{Z}$-secure function evaluation protocols. Note that the condition for SFE is weaker than the condition for MPC.

Theorem 3. *A set $\mathcal{P}$ of players can unconditionally $\mathcal{Z}$-securely compute any function if and only if $C^{(2)}(\mathcal{P},\mathcal{Z})$ and $C^{(1)}_{\mathsf{ORD}}(\mathcal{P},\mathcal{Z})$ hold, where*

$$C^{(2)}(\mathcal{P},\mathcal{Z}) \iff \forall (A_i,E_i,F_i),(A_j,E_j,F_j) \in \mathcal{Z}:\ E_i \cup E_j \cup (F_i \cap F_j) \neq \mathcal{P}$$

$$C^{(1)}_{\mathsf{ORD}}(\mathcal{P},\mathcal{Z}) \iff \begin{cases} \exists \textit{ ordering } \big((A_1,E_1,F_1),\ldots,(A_m,E_m,F_m)\big) \textit{ of } \overline{\mathcal{Z}} \textit{ s.t.}^{7} \\ \forall i,j \in \{1,\ldots,m\},\ i \leq j:\ E_j \cup F_i \neq \mathcal{P} \end{cases}$$

The sufficiency of the condition is proved by constructing an SFE protocol. Our approach is similar to the approach from [BFH+08]: First all players share their inputs, then the circuit is evaluated gate-by-gate, and then the output is publicly reconstructed. However, our conditions do not guarantee robust reconstructibility. In fact, the adversary can break down the computation and cause all the sharings to be lost. As the circuit is non-reactive, we handle such an abortion by repeating the whole protocol, including the input gates. In each repetition, the adversary might choose new inputs for the actively corrupted players. By ensuring that the adversary gets no information on any secrets unless the full protocol succeeds (including the evaluation of output gates), we make sure that she chooses these inputs independently of the other players' inputs.

Termination is guaranteed, by the fact that whenever the protocol aborts, a new set B of incorrect players is identified, and the next iteration proceeds without them. Hence, the number of iterations is bounded by n. This implies also that the total number of signatures in the computation is polynomial, hence the forging probability is negligible.

Special care needs to be taken in the design of the output protocol. For simplicity, we describe the protocol for a single public output. Using standard techniques one can extend it to allow several outputs and, furthermore, private outputs.

The idea of the output protocol is the following: First observe that the privacy of our sharing scheme is protected by a particular summand which is not given to the adversary. In fact, such a summand s_k is guaranteed to exist for each $(A_k,E_k,F_k) \in \mathcal{Z}$ by the $\mathcal{Z}$-privacy of the sharing specification $\mathcal{S}_{\mathcal{Z}}$. As long as this summand is not published, an adversary of class (A_k,E_k,F_k) gets no information about the output (from the adversary's point of view, s_k is a perfect blinding of the output, and all other summands s_i are either known to the adversary or are distributed uniformly). Second, observe that whenever the publishing of some summand s_k fails (i.e., PubAnnounce aborts), the players get information about the actual adversary $(A^\star,E^\star,F^\star)$, namely that $S_k \subseteq F^\star$. The trick is to announce the summands in such an order, that if the announcing of a summand s_k aborts, then from the information that $S_k \subseteq F^\star$ the players can deduce that the summand associated with the actual adversary class has not been yet announced. In particular, if an adversary class $Z_i = (A_i,E_i,F_i)$ could potentially abort the announcing of the summand s_k (i.e., if $S_k \subseteq F_i$), then the summand s_k should be announced strictly before s_i, i.e., the summand associated with Z_i, is announced.

[7] Remember that $\overline{\mathcal{Z}}$ denotes the maximum classes in $\mathcal{Z}$. One can verify that such an ordering exists for $\overline{\mathcal{Z}}$ exactly if it exists for $\mathcal{Z}$.

Let $((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m))$ denote an ordering of the maximal structure $\overline{\mathcal{Z}}$ satisfying: $\forall 1 \leq i \leq j \leq m : E_j \cup F_i \neq \mathcal{P}$, and let $\mathcal{S}$ denote the induced sharing specification $\mathcal{S} = (S_1, \ldots, S_m)$ with $S_k = \mathcal{P} \setminus E_k$. Then the protocol OutputGeneration (see next page) either publicly reconstructs a sharing $\langle s \rangle$ according to $\mathcal{S}$ or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy is guaranteed under the assumption that the summands of $\langle s \rangle$ not known to the adversary are uniformly distributed. As long as no signature is forged, this holds for all sharings in our protocols.

Protocol OutputGeneration$(\mathcal{P}, \mathcal{Z}, \mathcal{S} = (S_1, \ldots, S_m), \langle s \rangle)$

1. For $k = 1, \ldots, m$, the following steps are executed *sequentially*:
   (a) PubAnnounce$(\mathcal{P}, \mathcal{Z}, S_k, s_k, \sigma_{S_k}(s_k))$ is invoked to have the summand s_k published.
   (b) If PubAnnounce aborts with B, then OutputGeneration *immediately* aborts with B.
2. Every $p_j \in \mathcal{P}$ (locally) computes $s := \sum_{k=1}^{m} s_k$ and outputs s.

Lemma 13. *Assume that $C_{\text{IC}}(\mathcal{P}, \mathcal{Z})$ holds, $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification constructed as explained, the condition $\forall S_k \in \mathcal{S}, (\cdot, E, \cdot) \in \mathcal{Z} : S_k \not\subseteq E$ holds, $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$ with the property that those summands that are unknown to the adversary are randomly chosen, and no signature is forged. Then the protocol* OutputGeneration *either publicly reconstructs s, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. If* OutputGeneration *aborts, then the protocol does not leak any information on s to the adversary.*

For completeness, we also include a detailed description of the SFE protocol (see below) and state its security in the following lemma.

Protocol SFE$(\mathcal{P}, \mathcal{Z}, C)$

0. Let $\mathcal{S} = (\mathcal{P} \setminus E_1, \ldots, \mathcal{P} \setminus E_m)$ for the assumed ordering $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ of $\overline{\mathcal{Z}}$.
1. *Input stage:* For every input gate in C, Share is invoked to have the input player p_i share his input x_i according to $\mathcal{S}$.[a]
2. *Computation stage:* The gates in C are evaluated as follows:
   - *Addition gate:* Invoke Add to compute a sharing of the sum according to $\mathcal{S}$.
   - *Multiplication gate:* Invoke Mult to compute a sharing of the product according to $\mathcal{S}$.
3. *Output stage:* Invoke OutputGeneration$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle)$ for the sharing $\langle s \rangle$ of the public output.
4. If any of the sub-protocols aborts with B, then set $\mathcal{P} := \mathcal{P} \setminus B$, and set $\mathcal{Z}$ to the adversary structure which is compatible with B being incorrect, i.e., $\mathcal{Z} := \mathcal{Z}|_{\mathcal{P}}^{B \subseteq F}$, and go to Step 1.

[a] If in a later iteration a player $p_i \notin \mathcal{P}$ should give input, then the players in $\mathcal{P}$ pick the default sharing of a default value.

Lemma 14. *The protocol* SFE *is unconditionally $\mathcal{Z}$-secure if $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ and $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ hold.*

To complete the proof of Theorem 3 we need to show that unconditionally $\mathcal{Z}$-secure SFE is not possible for some circuits when $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ or $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ is violated. The necessity of $C^{(\mathbf{2})}(\mathcal{P}, \mathcal{Z})$ follows immediately from Lemma 11. The following lemma states the necessity of $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$. The idea of the proof is that when $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ is violated then in any protocol evaluating the identity function, the adversary can break down the computation at a point where she has gained noticeable (i.e., not negligible) information about the output, although the correct players have only negligible information. For a more detailed proof the reader is referred to the full version of this paper.

Lemma 15. *If $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ is violated, then there are functions that cannot be unconditionally $\mathcal{Z}$-securely evaluated.*

7 Computational Security

In this section we show that conditions $C^{(\mathbf{1})}(\mathcal{P}, \mathcal{Z})$ and $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ from Theorems 2 and 3 are sufficient and necessary for the existence of computationally $\mathcal{Z}$-secure MPC and SFE, respectively.

Theorem 4. *Assuming that enhanced trapdoor permutations exist, a set $\mathcal{P}$ of players can computationally $\mathcal{Z}$-securely compute any (reactive) computation (MPC) if and only if $C^{(\mathbf{1})}(\mathcal{P}, \mathcal{Z})$ holds, and any non-reactive function (SFE) if and only if $C^{(\mathbf{1})}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ holds.*

The proof of necessity is very similar to the proofs of Lemmas 12 and 15 and, therefore, it is omitted. The sufficiency is proved by describing protocols that realize the corresponding primitive. Our approach is different than the one used in the previous sections. In particular, first, we design a protocol for SFE and then use it to design a protocol for MPC.

Note that the above bounds directly imply corresponding bounds for a threshold adversary who actively corrupts t_a players, passively corrupts t_p players, and fail-corrupts t_f players, simultaneously. Using the notation from [FHM98], we say that a protocol is (t_a, t_p, t_f)-*secure* if it tolerates such a threshold adversary.

Corollary 1. *Assuming that enhanced trapdoor permutations exist, a set $\mathcal{P}$ of players can computationally (t_a, t_p, t_f)-securely compute any computation (reactive or not) if and only if $2t_a + t_p + t_f < |\mathcal{P}|$.*

7.1 The SFE Protocol

Our approach to SFE uses ideas from [IKLP06]. The evaluation of the given circuit C proceeds in two stages, called the *computation stage* and the *output stage*. In the computation stage a uniformly random *sharing* of the output of C on inputs provided by the players is computed.[8] For this purpose we use the (non-robust) SFE protocol from [Gol04]

[8] Without loss of generality (as in Section 6) we assume that the circuit C to be computed has one public output.

for dishonest majority which achieves partial fairness and unanimous abort [GL02]. In the output stage the sharing of the output is publicly reconstructed, along the lines of the reconstruction protocol from Section 6. Both stages are non-robust and they might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players, but without violating privacy of the inputs. When this happens the whole evaluation is repeated among the players in $\mathcal{P} \setminus B$, where the inputs of the players in B are fixed to a default pre-agreed value, and the adversary structure $\mathcal{Z}$ is reduced to the structure $\mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B}$, i.e., the structure which is compatible with the players in B being incorrect.

The secret-sharing scheme used here is similar to the one we use in the unconditional-security case. More precisely, the secret is split into uniformly random summands $s_1, \ldots, s_m \in \mathbb{F}$ that add up to the secret, where each player might hold several of those summands, according to some sharing specification $\mathcal{S} = (S_1, \ldots, S_m)$. The difference is that the players do not hold signatures on their summands, but they are committed to them (towards all players) by a perfectly hiding commitment scheme.[9] In particular, for each summand s_k, all players hold a commitment to s_k such that each $p_i \in S_k$ holds the corresponding decommitment information to open it.

The computation stage In the computation stage, instead of C we evaluate the circuit C' which computes a uniformly random *sharing* $\langle y \rangle$ of the output y of C according to $\mathcal{S}_{\mathcal{Z}}$, i.e., the sharing specification associated with $\mathcal{Z}$. The circuit C' can be easily constructed from C [IKLP06]. To evaluate C' the players invoke the protocol for SFE from [Gol04] for the model where authenticated broadcast channels (but no bilateral point-to-point channels) are given, which tolerates any number of $t < n$ actively corrupted players. As proved in [Gol04], with this protocol we achieve the following properties: There is a $p \in \mathcal{P}$ (specified by the protocol), such that when p is uncorrupted the circuit C' is securely evaluated, otherwise the adversary can decide either to make *all* players abort the protocol or to allow C' to be securely evaluated. Note that the adversary can decide whether or not the protocol aborts even after having received the outputs of the passively corrupted players. Furthermore, by inspecting the protocol in [Gol04], one can verify that it actually satisfies some additional properties, which are relevant when all three corruption types are considered, namely (1) if p is *correct* then the protocol does not abort,[10] (2) a correct player always gives his (correct) input to the evaluation of C', and (3) a non-actively corrupted player does not give a wrong input (but might give no input if he crashes). By the above properties it is clear that the protocol can abort only if p is incorrect (i.e., $B = \{p\}$). Moreover, when it aborts privacy of the inputs is not violated as the outputs of passively corrupted players are their shares of $\langle y \rangle$ plus perfectly hiding commitments to all the summands of $\langle y \rangle$.

The output stage The output stage is similar to the output stage of protocol SFE described in Section 6. The summands of $\langle y \rangle$ are announced sequentially in the order implied by $C^{(1)}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$. This guarantees (as in protocol $\mathsf{OutputGeneration}$) that when the announcing of a summand aborts, then the output stage can abort without violating privacy (the summand of $\langle y \rangle$ associated with the actual adversary has not been announced

[9] Such commitment schemes are known to exist if (enhanced) trapdoor permutations exist [GMW86].

[10] Note that a correct player is not necessary uncorrupted.

yet). To announce a summand, protocol CompPubAnnounce is invoked which is a trivially modified version of PubAnnounce to use openings of commitments instead of signatures. We refer to the abovely described SFE protocol as CompSFE.

Lemma 16. *Assuming that enhanced trapdoor permutations exist, the protocol* CompSFE *is computationally $\mathcal{Z}$-secure if $C^{(1)}_{\mathsf{ORD}}(\mathcal{P}, \mathcal{Z})$ holds.*

7.2 The MPC Protocol

A protocol for MPC can be built based on a (robust) general SFE protocol and a robustly reconstructible secret-sharing scheme, in a straightforward way: the SFE protocol is used to securely evaluate the circuit gate-by-gate, where each intermediary result is shared among the players. In fact, the secret-sharing scheme described is Section 7.1, for sharing specification $\mathcal{S}_{\mathcal{Z}}$, is robustly reconstructible if $C^{(1)}(\mathcal{P}, \mathcal{Z})$ holds. Indeed, condition $C^{(1)}(\mathcal{P}, \mathcal{Z})$ ensures that for any shared value each summand is known to at least one player who is not actively or fail-corrupted and will not change or delete it. Hence, the shared value is uniquely determined by the views of the players. Therefore, we can use protocol CompSFE to evaluate any (reactive) circuit as follows: For each input gate, invoke CompSFE to evaluate the circuit C_{input} which computes a sharing (according to $\mathcal{S}_{\mathcal{Z}}$) of the input value. For the addition and multiplication gate, invoke CompSFE to evaluate the circuits C_{add} and C_{mult} which on input the sharings of two values s and t output a sharing of the sum $s + t$ and of the product st, respectively. For output gates, invoke CompSFE to evaluate the circuit C_{output} which on input the sharing of some value s outputs s towards the corresponding player. We refer to the resulting MPC protocol as CompMPC.

Lemma 17. *Protocol* CompMPC *is computationally $\mathcal{Z}$-secure if $C^{(1)}(\mathcal{P}, \mathcal{Z})$ holds.*

8 Conclusions

We considered MPC and SFE in the presence of a general adversary who can actively, passively, and fail corrupt players, simultaneously. For both primitives we gave exact characterizations of the tolerable adversary structures for achieving unconditional (aka statistical) and computational security, when a broadcast channel is given. As in the case of threshold adversaries, the achieved bounds are strictly better than those required for perfect security, where no error probability is allowed. Our results confirm that in all three security models (perfect, unconditional, and computational) there are adversary structures that can be tolerated for SFE but not for MPC.

References

[Alt99] B. Altmann. Constructions for efficient multi-party protocols secure against general adversaries. Diploma Thesis, ETH Zurich, 1999.

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):370–381, 1991.

[BFH+08] Z. Beerliová-Trubíniová, M. Fitzi, M. Hirt, U. Maurer, and V. Zikas. MPC vs. SFE: Perfect security in a unified corruption model. In *TCC 2008, LNCS 4948*, pp. 231–250, 2008.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88*, pp. 1–10, 1988.

[BHR07] Z. Beerliová-Trubíniová, M. Hirt, and M. Riser. Efficient Byzantine agreement with faulty minority. In *ASIACRYPT 2007, LNCS 4833*, pp. 393–409, 2007.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC '88*, pp. 11–19, 1988.

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

[Cle86] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC '86*, pp. 364–369, 1986.

[FHM98] M. Fitzi, M. Hirt, and U. Maurer. Trading correctness for privacy in unconditional multi-party computation. In *CRYPTO '98, LNCS 1462*, pp. 121–136, 1998. Corrected version is available online.

[FHM99] M. Fitzi, M. Hirt, and U. Maurer. General adversaries in unconditional multi-party computation. In *ASIACRYPT '99, LNCS 1716*, pp. 232–246, 1999.

[GL02] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC 2002, LNCS 2508*, pp. 17–32, 2002.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS '86*, pp. 174–187, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *STOC '87*, pp. 218–229, 1987.

[Gol04] O. Goldreich. Foundations of Cryptography: Volume 2, Basic Applications. *Cambridge University Press*, New York, NY, USA, 2004.

[GRR98] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98*, pp. 101–111, 1998.

[HM97] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *PODC '97*, pp. 25–34, 1997.

[HM00] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[IKLP06] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *CRYPTO 2006, LNCS 4117*, pp. 483–500, 2006.

[Mau02] U. Maurer. Secure multi-party computation made simple. In *SCN 2002, LNCS 2576*, pp. 14–28, 2002.

[Mau06] U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370-381, 2006.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC '89*, pp. 73–85, 1989.

[Yao82] A. C. Yao. Protocols for secure computations. In *FOCS '82*, pp. 160–164, 1982.