MPC vs. SFE: Perfect Security in a Unified Corruption Model

Zuzana Beerliová-Trubíniová, Matthias Fitzi, Martin Hirt, Ueli Maurer, and Vassilis Zikas

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

Abstract. Secure function evaluation (SFE) allows a set of players to compute an arbitrary agreed function of their private inputs, even if an adversary may corrupt some of the players. Secure multi-party computation (MPC) is a generalization allowing to perform an arbitrary on-going (also called reactive or stateful) computation during which players can receive outputs and provide new inputs at intermediate stages.
At Crypto 2006, Ishai *et al.* considered mixed threshold adversaries that either passively corrupt some fixed number of players, or, alternatively, actively corrupt some (smaller) fixed number of players, and showed that for certain thresholds, cryptographic SFE is possible, whereas cryptographic MPC is not.
However, this separation does not occur when one considers *perfect* security. Actually, past work suggests that no such separation exists, as all known general protocols for perfectly secure SFE can also be used for MPC. Also, such a separation does not show up with *general adversaries*, characterized by a collection of corruptible subsets of the players, when considering passive and active corruption.
In this paper, we study the most general corruption model where the adversary is characterized by a collection of adversary classes, each specifying the subset of players that can be actively, passively, or fail-corrupted, respectively, and show that in this model, perfectly secure MPC separates from perfectly secure SFE. Furthermore, we derive the exact conditions on the adversary structure for the existence of perfectly secure SFE resp. MPC, and provide efficient protocols for both cases.

1 Introduction

1.1 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. Security means that dishonest players can neither falsify the output of the computation, nor can obtain information about the honest players' inputs (except what they can derive from their own inputs). (Reactive) secure multi-party computation (MPC) is a slight generalization of SFE. Here, the function to be computed is reactive, meaning that players can give inputs and get outputs several times during the course of the computation, and every output can depend on all inputs given so far.

A bit more formally, SFE and MPC can be best described by considering a hypothetical trusted party which performs the specified task on behalf of the players. In SFE, the trusted party is non-reactive: it takes inputs from the players, evaluates the function, and announces the outputs (and disappears). In MPC, the trusted party is reactive: it continuously interacts with the players, taking inputs and sending outputs. It maintains an internal state which is updated with every input, and every output is computed based on this state. The goal of SFE and MPC is to *simulate* this trusted party among the set $\mathcal{P}$ of players. 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 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).

Typical examples of SFE include e-voting, i.e., the computation of the sum of the players' secret votes, or the double-agent problem, i.e., the identification of identical entries in several confidential databases. An example of MPC is the simulation of a fair stock market, where inputs (e.g. new trading orders) are given and outputs (e.g. current stock prices) are provided while the computation proceeds.

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,RB89,Bea91b] provide information-theoretic security.

1.2 Summary of Known Results

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 for passive corruption if and only if $t < n/2$.

These results were unified and extended by fail-corruption in [FHM98] by proving that perfectly secure MPC is achievable if and only if $3t_a + 2t_p + t_f < n$, where t_a, t_p, and t_f denote the upper bounds on the number of actively, passively and fail corrupted players, respectively.

Another line of generalization is concerned with so-called general adversaries: Here, the adversary is not characterized by a threshold, but rather by an enumeration of the possible subsets of players that the adversary can corrupt.[1] In [HM97] it was proved that perfect security is possible if and only if no two corruptible subsets cover the full players set (passive adversary), respectively no three corruptible subsets cover the full player set (active case). These results naturally generalize the threshold results of $2t < n$, respectively $3t < n$. These results were unified to a mixed general adversary in [FHM99], where the adversary is characterized by an enumeration of classes, each class consisting of an actively corruptible subset of players and of a passively corruptible subset of the players. Fail-corruption was not considered. The bounds on the existence of perfectly secure MPC are a natural combination of the bounds in the threshold model.

[1] This allows to model non-symmetric settings where not every player's potential dishonesty is modeled in exactly the same way. Some coalitions of colluding players might be more likely than others, and some players might have a higher level of dishonesty than others.

A similar development of generalizations (from threshold via dual-failure to general adversaries) can be observed in the area of Byzantine agreement protocols [LSP82,DS82,LF82,MP91,GP92,FM98,AFM99].

Recently, Ishai *et al.* [IKLP06] considered a mixed model in which the adversary can either corrupt t_a players actively, or, *alternatively*, t_p players passively (in contrast to previous work [FHM98], where the adversary could corrupt t_a players actively, and, simultaneously, t_p players passively). They showed that for $t_p < n$ and $t_a < n/2$ cryptographically secure SFE is possible, whereas, for $t_p = n-1$ and $t_a \geq 1$, cryptographically secure (reactive) MPC is not possible.

1.3 Contributions of this Paper

The original motivation for this paper was to determine the exact conditions for SFE and MPC in the natural and most general adversary model where all corruption types can occur. We characterize the adversary's corruption capability by an *adversary structure* $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$, where $A_k, E_k, F_k \subseteq \mathcal{P}$ and $A_k \subseteq E_k$ and $A_k \subseteq F_k$. The adversary can (secretly) choose an arbitrary *adversary class* $Z_k = (A_k, E_k, F_k) \in \mathcal{Z}$ and actively corrupt the players in A_k, passively corrupt the players in E_k, and fail-corrupt the players in F_k. In the technical sections of this paper, we present and prove exact conditions on the adversary structure to allow perfectly secure MPC and perfectly secure SFE. This unifies all previously considered models, where either not all three types of corruption were considered, or where the corruption capability was specified in terms of thresholds.

Interestingly, the conditions for SFE and MPC are different. This is surprising since all known results on perfectly secure protocols suggest no such separation. In particular, when considering active, passive and fail-corruption (but only *threshold* type), then no such separation has been observed [FHM98]. When considering general adversaries (with active and passive corruption, but *without fail-corruption*), no separation can be observed neither [FHM99]. However, in the combination of both these models, the separation shows up. This indicates that the most general adversary model considered here is both natural and appropriate since all restricted models hide the fact that SFE and MPC separate.

We describe a simple example of an adversary structure which separates, i.e., for which SFE with perfect security is possible but MPC is not. Let $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ and $\mathcal{Z} = \{Z_1, Z_2, Z_3\}$, where $Z_1 = (\emptyset, \{p_1\}, \emptyset)$, $Z_2 = (\{p_2\}, \{p_2\}, \{p_2, p_4\})$, and $Z_3 = (\{p_3\}, \{p_3\}, \{p_3, p_4\})$. In other words, the adversary can either corrupt p_1 passively, or corrupt p_2 actively and fail-corrupt p_4, or corrupt p_3 actively and fail-corrupt p_4.[2]

A protocol for SFE works as follows: First use p_4 as the trusted party with the constraint that p_4 sends the output of the function first to p_1 and then to p_2 and p_3. If p_4 crashes, then restart the protocol using p_1 as trusted party (the crashing of p_4 guarantees that the adversary did not choose $Z_1 \in \mathcal{Z}$ and hence that p_1 is uncorrupted). If p_1 has received the output from p_4 before p_4 crashed, then he forwards it to p_2 and p_3, otherwise he evaluates the function on the inputs received by p_2 and p_3 and sends them the output. The security of this protocol is trivial to verify. The impossibility of MPC for this example follows from the observation that if some intermediate value v — part of the state of an MPC protocol — is not known to p_1, then there is no protocol that always reveals it to him. Indeed, if in such a protocol the adversary crashes p_4 and forces p_2 or p_3 to send random messages whenever he is instructed to send something (he can do so by choosing Z_2 or Z_3), then with non-zero probability, p_1 will not be able to decide whether p_2 or p_3 is misbehaving and will accept a value different than v, contradicting perfect security.

2 The Model

We consider the standard secure-channels model introduced in [BGW88,CCD88]: The players $p_1, \ldots, p_n$ are connected by a complete network of bilateral synchronous secure channels. The computation is described as an arithmetic circuit over some finite field $\mathbb{F}$, consisting of addition (or linear) gates and multiplication gates.

The security of our protocols is information-theoretic without error probability, which is called *perfect* security and is the strongest possible security notion. A protocol is defined to be secure if it realizes a trusted functionality (computing the function f), where the term "realize" is defined via the simulation paradigm

[2] Additionally, $Z_4 = (\{p_4\}, \{p_4\}, \{p_4\})$ could be tolerated, but this would unnecessarily complicate the example.

[Can00,MR91,Bea91a,DM00,PW01] which, in a nutshell, guarantees that whatever the adversary can achieve in the real world where the protocol is executed, he could also achieve in the ideal setting with the trusted functionality.[3] This security notion implies in particular that the adversary cannot obtain any information about the players' inputs beyond what is implied by the outputs (secrecy), and that he cannot influence the outputs other than by choosing the inputs of the corrupted players (correctness).

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). The adversary chooses a triple in $\mathcal{Z}$ non-adaptively,[4] i.e., before the beginning of the protocol; this triple is denoted as $Z^\star = (A^\star, E^\star, F^\star)$ and is called the *actual adversary class* or simply the actual adversary. The players in $A^\star$, $E^\star$, and $F^\star$ are actively, passively and fail-corrupted, respectively. 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 $(A, E, F) \in \mathcal{Z}$ (anyway, an actively corrupted player can behave as being passively or fail-corrupted). Furthermore, as most constructions only need to consider the maximal classes of a structure, we define the maximal structure $\overline{\mathcal{Z}} = \big\{(A, E, F) \in \mathcal{Z} : \nexists(A', E', F') \in \mathcal{Z}$ with $(A, E, F) \neq (A', E', F')$ and $A \subseteq A'$, $E \subseteq E'$, $F \subseteq F'\big\}$.

To simplify the description, we adopt the following convention: Whenever a player does not receive a message (when expecting one), 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 been crashed, he only sends $\perp$. If a player has followed the protocol instructions correctly up to a certain point, he is called *correct* at that point, independently of whether he is actually corrupted. A player who has deviated from the protocol (e.g., has crashed or has sent inconsistent messages) is called *incorrect*.

3 Tools (Sub-protocols)

In this section we present some protocols that will be used as building blocks in the main sections. Several of these protocols are non-robust, i.e., they might abort when faults occur. In case of abortion, all (correct) players agree on a non-empty set $B \subseteq \mathcal{P}$ of incorrect players; we say then that *the protocol aborts with B*.

3.1 Broadcast and Consensus

A *broadcast protocol* allows a sender p with input value v to distribute v among a set $\mathcal{P}$ of players, where it is guaranteed that all correct players in $\mathcal{P}$ output the same value v' (consistency), and that $v' = v$ when the sender is correct during the execution of the protocol (correctness). Similarly, a *consensus protocol* allows a set $\mathcal{P}$ of players, each holding an input value v_i, to reach agreement, such that every correct player in $\mathcal{P}$ outputs the same value v' (consistency), and that $v' = v$ if all (correct) players hold as input v (correctness).

In [AFM99] a tight condition on the existence of perfectly-secure broadcast and consensus is given for the model with active and fail-corruption. The presented protocols assume pairwise authenticated (but not necessarily private) channels, hence they remain secure even when the adversary is allowed to passively corrupt any number of players. Therefore these conditions immediately translate to our model:

Lemma 1. *In the secure channels model, perfectly $\mathcal{Z}$-secure broadcast and consensus among a set $\mathcal{P}$ of players is possible if and only if $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, where*

$$C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z}) \iff \forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : \; A_1 \cup A_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P}.$$

We denote the broadcast and the consensus protocol of [AFM99] by $\mathsf{Broadcast}$ and $\mathsf{Consensus}$, respectively.

[3] While our protocols can be proven secure in any of these simulation-based frameworks, with perfect indistinguishability of the real and the ideal world, we will in this paper not give full-fledged simulation-based security proofs; this is consistent with the previous literature on secure SFE and MPC.

[4] In contrast, an *adaptive* adversary can corrupt more and more players during the protocol execution, subject only to the constraint that the corrupted sets are within one of the triples in $\mathcal{Z}$. We do not consider the adaptive setting in this paper, but our results could be generalized to it.

3.2 Crash Detection

We present a protocol which allows the players in $\mathcal{P}$ to commonly detect whether a specific player $p \in \mathcal{P}$ is alive or has crashed. Such a decision cannot be sharp, as an actively corrupted player can always behave as having crashed, i.e., not send any messages during the execution of the sub-protocol. However, we require that correct players are always identified as "alive", and crashed players are always identified as "crashed".

Protocol $\mathsf{CDP}(\mathcal{P}, \mathcal{Z}, p)$

1. p sends a 1-bit to every $p_j \in \mathcal{P}$.
2. Every $p_j \in \mathcal{P}$ sets $b_j = 1$ if he received a 1-bit, and $b_j = 0$ otherwise.
3. The players in $\mathcal{P}$ invoke Consensus on inputs $b_1, \ldots, b_n$.
4. Every $p_j \in \mathcal{P}$ outputs "alive" when the output of the consensus protocol is 1, and "crashed" otherwise.

Lemma 2. *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, then the protocol* $\mathsf{CDP}(\mathcal{P}, \mathcal{Z}, p)$ *has the following properties: Consistency: The (correct) players agree on the output. Correctness: If p is correct until the end of* CDP*, then every (correct) player outputs "alive" and if p has crashed* before *the invocation of* CDP*, then every (correct) player outputs "crashed".*[5]

3.3 Strong Broadcast

Intuitively, a fail-corrupted player never sends a "wrong" message; in the worst case, he sends no message at all. This intuition does not apply to broadcast (according to the standard definition): When the sender of a broadcast protocol crashes, only consistency of the output is guaranteed. But the output value can be arbitrary.[6]

We lift the intuition that fail-corrupted players never send "wrong" messages to broadcast by introducing the notion of *strong broadcast*: A protocol with sender p, holding input v, achieves strong broadcast when it achieves broadcast and additionally ensures that the output is in $\{v, \perp\}$ when the sender is not actively-corrupted. We show how to construct a protocol for p to strongly broadcast v, given a protocol for broadcast (e.g., Broadcast) and CDP.

Protocol $\mathsf{StrongBroadcast}(\mathcal{P}, \mathcal{Z}, p, v)$

1. Invoke Broadcast to have p broadcast his input v. For each $p_j \in \mathcal{P}$, let v_j denote p_j's output in Broadcast.
2. Invoke CDP to detect whether p is alive or has crashed.
3. Every $p_j \in \mathcal{P}$ outputs v_j when p is alive, and $\perp$ when p has crashed.

Lemma 3. *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, then the protocol* $\mathsf{StrongBroadcast}(\mathcal{P}, \mathcal{Z}, p, v)$ *has the following properties: Consistency: All (correct) players output the same value v'. Correctness: If the sender p is correct, then $v' = v$; if p crashed* before *the invocation of the protocol, then $v' = \perp$; if p crashes during the protocol, then $v' \in \{v, \perp\}$.*

3.4 Secret Sharing

A secret-sharing scheme allows a player (called the dealer) to distribute a secret, in such a way that only qualified sets of players can reconstruct it. As secret-sharing scheme, we employ a sum sharing (i.e., the secret is split into summands that add up to the secret), folded with a replication sharing (i.e., every summand is given to a subset of the players): Such a sharing is characterized by a *sharing specification* $\mathcal{S}$, which is a vector of subsets of the player set $\mathcal{P}$. A value s is *shared* with respect to a sharing specification $\mathcal{S} = (S_1, \ldots, S_m)$, when there exist summands $s_1, \ldots, s_m$ with $s = \sum s_k$, and s_k is given to every $p_i \in S_k$. For a player $p_i \in \mathcal{P}$, we consider the vector $(s_{i_1}, \ldots, s_{i_\ell})$ of summands held by p_i to be p_i's *share* of s, denoted as $\langle s \rangle_i$. The vector of all shares, denoted as $\langle s \rangle = \big(\langle s \rangle_1, \langle s \rangle_2, \ldots, \langle s \rangle_n\big)$, is a *sharing* of s. We say that $\langle s \rangle$ is a (consistent) sharing of s according to $(\mathcal{P}, \mathcal{S})$, if for each $S_i \in \mathcal{S}$ all (correct) players in S_i have the same view on s_i and $s = \sum_{i=1}^{m} s_i$.

[5] Note that in any case the adversary learns the output of CDP.

[6] In [AFM99], the output of broadcast can even be chosen by the adversary, when the sender crashes.

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 $\forall(A,E,F) \in \mathcal{Z}\ \exists S \in \mathcal{S} : S \cap E = \emptyset$. For an adversary structure $\mathcal{Z}$ with maximal classes $\overline{\mathcal{Z}} = \{(\cdot, E_1, \cdot), \ldots, (\cdot, E_m, \cdot)\}$, we denote the natural $\mathcal{Z}$-private sharing specification by $S_{\mathcal{Z}} = (\mathcal{P}\backslash E_1, \ldots, \mathcal{P}\backslash E_m)$.

The following protocol 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 a modification of the sharing protocol from [Mau02] to tolerate fail-corruption. It may abort when p is incorrect.

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

1. Dealer p chooses the summands $s_2, \ldots, s_{|\mathcal{S}|}$ randomly and sets $s_1 = s - \sum_{k=2}^{|\mathcal{S}|} s_k$.
2. Execute the following steps for $k = 1, \ldots, |\mathcal{S}|$:
   (a) p sends s_k to every $p_i \in S_k$, who denotes the received value as $s_k^{(i)}$ ($\perp$ when no value is received).
   (b) Every $p_i \in S_k$ sends $s_k^{(i)}$ to every $p_j \in S_k$, who denotes the received value as $s_k^{(i,j)}$.
   (c) For each $p_j \in S_k$ StrongBroadcast is invoked to have p_j broadcast a complaint bit $b_{k,j}$, where $b_{k,j} = 1$ when $s_k^{(j)} =\perp$ or $s_k^{(i,j)} \notin \{s_k^{(j)}, \perp\}$ for some i, and $b_{k,j} = 0$ otherwise.
   (d) If a complaint was reported (i.e., $b_{k,j} = 1$ for some j), then StrongBroadcast is invoked to have p broadcast s_k, and every $p_j \in S_k$ sets $s_k^{(j)}$ to the broadcasted value.
3. If p broadcasts $\perp$ in Step 2d, then Share aborts with $B = \{p\}$.

Lemma 4. *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds and $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, then the protocol* Share *$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p, s)$ has the following properties. Correctness:* Share *either outputs a consistent sharing of some s', where $s' = s$ unless the dealer is actively corrupted, or it aborts with $B = \{p\}$; it does not abort if p is correct. Secrecy: No information on s leaks to the adversary.*

Reconstructing a shared value towards a player is straight-forward: All players send the summands they know (i.e., their share) to the output player, who tries to find the correct value for each summand and computes the secret as the sum of the summands. However, finding the correct value of a summand is not always possible when corrupted players send wrong values or no value to the output player, so we need an extra condition on the adversary structure to ensure that the output player can always decide on the value of every summand. We can slightly relax this condition when a sharing is reconstructed publicly (rather than towards a dedicated output player): In this case, the players can decide depending on the published values whether a summand is uniquely defined or not, and if not, agree on a set $B \subseteq \mathcal{P}$ of incorrect players.

In the sequel, we present the protocols Announce and Reconstruct to announce a summand, respectively reconstruct a sharing, towards a dedicated player, and the protocols PublicAnnounce and PublicReconstruct to announce a summand, respectively to reconstruct a sharing, towards all players. The latter protocols are non-robust; they might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. The abortion of the protocol PublicAnnounce will allow to derive information on the actual adversary class, which will be helpful in the output protocol of SFE.

Protocol Announce$(\mathcal{P}, \mathcal{Z}, S_k, s_k, p)$

1. Every $p_i \in S_k$ sends s_k to p, who denotes the received value as $s_k^{(i)}$ ($\perp$ when no value is received).
2. Let $V \subseteq \mathbb{F}$ denote the set of values v that are "explainable" with some adversary in $\mathcal{Z}$, i.e., for which there is an adversary class $(A, E, F) \in \mathcal{Z}$, such that $\{p_i \in S_k : s_k^{(i)} =\perp\} \subseteq F$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$.
3. p sets s_k to be the smallest element in V.

Lemma 5. *If $\forall(A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z}$: $S_k \not\subseteq A_1 \cup A_2 \cup (F_1 \cap F_2)$, then the protocol* Announce *robustly announces s_k to p.*

Proof. We have to prove that (i) the set V contains the correct summand s_k and (ii) the set V contains no other values. (i) Observe that the summands $s_k^{(i)}$ received by p satisfy that $\{p_i \in S_k : s_k^{(i)} =\perp\} \subseteq F^\star$ and

$\{p_i \in S_k : s_k^{(i)} \notin \{s_k, \perp\}\} \subseteq A^\star$, where $(A^\star, E^\star, F^\star)$ denotes the actual adversary class. As $(A^\star, E^\star, F^\star) \in \mathcal{Z}$, it follows that $s_k \in V$. (ii) Consider any value $v \in V$. There exists an adversary class $(A, E, F) \in \mathcal{Z}$ such that $\{p_i \in S_k : s_k^{(i)} = \perp\} \subseteq F$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$. By assumption we know that $S_k \not\subseteq A \cup A^\star \cup (F \cap F^\star)$, hence there exists a player $p_i \in S_k$ with $s_k^{(i)} \neq \perp$, $p_i \notin A$ and $p_i \notin A^\star$. This implies that $v = s_k^{(i)} = s_k$. □

Protocol Reconstruct$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, p)$

1. For every $S_k \in \mathcal{S}$, Announce is invoked to have the correct summand s_k announced towards p.
2. p computes $s = \sum_{k=1}^{|\mathcal{S}|} s_k$ and outputs s.

Lemma 6. *If* $\forall k = 1, \ldots, |\mathcal{S}|$, $\forall (A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z}$*:* $S_k \not\subseteq A_1 \cup A_2 \cup (F_1 \cap F_2)$*, then the protocol* Reconstruct *robustly reconstructs* s *towards* p*.*

The proof follows immediately from Lemma 5.

Protocol PublicAnnounce$(\mathcal{P}, \mathcal{Z}, S_k, s_k)$

1. Every $p_i \in S_k$ publishes his value for s_k (denoted as $s_k^{(i)}$) using StrongBroadcast.
2. Every $p_j \in \mathcal{P}$: determine the set $V \subseteq \mathbb{F}$ of values that are "explainable" with some adversary in $\mathcal{Z}$ (see protocol Announce).
3. Every $p_j \in \mathcal{P}$: output $s_k \in V$ if $|V| = 1$, otherwise abort with $B = \{p_i \subseteq S_k : s_k^{(i)} = \perp\}$.

Lemma 7. *If* $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ *holds and* $\forall (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z}$*:* $S_k \not\subseteq A_1 \cup A_2$*, then the protocol* PublicAnnounce *either publicly announces* s_k*, or aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. When it aborts, then there exists an adversary class* $(A, E, F) \in \mathcal{Z}$ *such that* $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$*.*

Proof. As V contains at least the correct summand s_k (see proof of Lemma 5), it is clear that PublicAnnounce either outputs s_k or aborts. It remains to be shown that when it aborts with B, then $|B| > 0$ and there exists an adversary class $(A, E, F) \in \mathcal{Z}$ such that $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$. Note that $s_k \in V$, hence PublicAnnounce aborts only when there exists a value $v \neq s_k$ with $v \in V$. This implies that there is an adversary class $(A, E, F) \in \mathcal{Z}$ with $\{p_i \in S_k : s_k^{(i)} = \perp\} \subseteq F$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$. Because $v \neq s_k$, we need $\{p_i \in S_k : s_k^{(i)} \neq \perp\} \subseteq A \cup A^\star$, which implies that $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$. Furthermore, B must be non-empty, because otherwise $S_k \subseteq (A^\star \cup A)$ would hold, contradicting the assumption in the Lemma. □

Protocol PublicReconstruct$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle)$

1. For every $S_k \in \mathcal{S}$, PublicAnnounce is invoked to have the correct summand s_k announced. If an invocation of PublicAnnounce aborts with B, then also PublicReconstruct aborts with B.
2. Every $p_j \in \mathcal{P}$ computes $s = \sum_{k=1}^{|\mathcal{S}|} s_k$ and outputs s.

Lemma 8. *If* $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ *holds and* $\forall k = 1, \ldots, |\mathcal{S}|$, $\forall (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z}$*:* $S_k \not\subseteq A_1 \cup A_2$*, then the protocol* PublicReconstruct *either publicly reconstructs* s*, or aborts with a non-empty set* B *of incorrect players.*

The proof follows immediately from Lemma 7.

3.5 Multiplication

We present a protocol for securely computing a sharing of the product of two shared values. The protocol is a variation of the multiplication protocol of [Mau02], capturing fail-corruptions. The multiplication protocol may abort when faults occur, with outputting a set $B \subseteq \mathcal{P}$ of incorrect players.

The idea of the protocol is the following: As s and t are shared according to $\mathcal{S}$, we can use the summands $s_1, \ldots, s_{|\mathcal{S}|}$ and $t_1, \ldots, t_{|\mathcal{S}|}$ to compute the product st as $st = \sum_{k,\ell=1}^{|\mathcal{S}|} s_k t_\ell$. To do so, each term $x_{k,\ell} = s_k t_\ell$ of this sum is shared by every player knowing both s_k and t_ℓ. Then the players perform consistency checks on the shared summands, and compute the sum of the shared terms $x_{k,\ell}$, which results in a sharing of st.

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:
   (a) Every $p_i \in (S_k \cap S_\ell)$ computes the products $x_{k,\ell} = s_k t_\ell$ and invokes $\mathsf{Share}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p_i, x_{k,\ell})$; denote the resulting sharing as $\langle x_{k,\ell}^{(i)} \rangle$.
   (b) Let p_i denote the player with the smallest index in $(S_k \cap S_\ell)$. For every $p_j \in (S_k \cap S_\ell)$, the difference $\langle x_{k,\ell}^{(j)} \rangle - \langle x_{k,\ell}^{(i)} \rangle$ is computed and, by invoking PublicReconstruct, reconstructed.
   (c) If all differences are 0, then the sharing $\langle x_{k,\ell}^{(i)} \rangle$ of p_i is adopted as sharing of $x_{k,\ell}$, i.e., $\langle x_{k,\ell} \rangle = \langle x_{k,\ell}^{(i)} \rangle$. Otherwise (i.e., some difference is non-zero), PublicAnnounce is invoked to have both s_k and t_ℓ announced, and a default sharing $\langle x_{k,\ell} \rangle$ of $x_{k,\ell} = s_k t_\ell$ is created (e.g., the first summand is set to $x_{k,\ell}$ and the other summands are set to 0).
2. Each player in $\mathcal{P}$ (locally) computes his share of the product st as the sum of his shares of all terms $x_{k,\ell}$.
3. If any of the invoked sub-protocols aborts with B, then also Mult aborts with B.

Lemma 9. *Assuming that $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, $\langle s \rangle$ and $\langle t \rangle$ are consistent sharings according to $\mathcal{S}$, $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, $\forall S_k, S_\ell \in \mathcal{S}, \forall (A, \cdot, \cdot) \in \mathcal{Z} : S_k \cap S_\ell \not\subseteq A$, and $\forall S_k \in \mathcal{S}, \forall (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z} : S_k \not\subseteq A_1 \cup A_2$, the protocol* $\mathsf{Mult}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$ *has the following properties. Correctness: It either outputs a sharing of st according to $(\mathcal{P}, \mathcal{S})$ or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Secrecy: No information on the inputs (i.e., on $\langle s \rangle$ and $\langle t \rangle$) leaks to the adversary.*

Proof. Correctness: The conditions in the lemma are sufficient for all the invoked sub-protocols (Share,PublicReconstruct,PublicAnnounce). The condition $\forall S_k, S_\ell \in \mathcal{S}, \forall (A, \cdot, \cdot) \in \mathcal{Z} : S_k \cap S_\ell \not\subseteq A$ ensures that every $x_{k,\ell}$ is known to at least one player p_i who is not actively corrupted; hence if no invocation of Share aborts and all differences are zero, then the shared values are correct. Privacy: Due to the security of Share, the invocations of Share do not leak information to the adversary. Furthermore, PublicAnnounce is only invoked on summands s_k, t_ℓ when two players in $S_k \cap S_\ell$ contradict each other; at least one of these players is actively corrupted, hence the adversary already knows s_k, t_ℓ before PublicAnnounce is invoked. □

3.6 Resharing

In the context of MPC, we will need to reshare shared values according to a different sharing specification. The key idea is to have every summand s_i in the original sharing being reshared according to the new sharing specification, and then distributively add the sharings of the summand, resulting in a new sharing of the original value. Due to space restrictions, the protocol $\mathsf{Reshare}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{S}', \langle s \rangle)$ is given in full detail in Appendix A. The following lemma, proved in Appendix A, states the achieved security.

Lemma 10. *Assuming that $\mathcal{S}'$ is a $\mathcal{Z}$-private sharing specification, $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$, $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, and $\forall S_k \in \mathcal{S}, S'_k \in \mathcal{S}', (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z} : \; (S_k \not\subseteq A_1 \cup A_2) \wedge (S'_k \not\subseteq A_1 \cup A_2)$, the protocol* $\mathsf{Reshare}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{S}', \langle s \rangle)$ *has the following properties. Correctness: It either outputs a sharing of s according to $(\mathcal{P}, \mathcal{S}')$ or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Secrecy: No information on the inputs (i.e., on $\langle s \rangle$) leaks to the adversary.*

4 (Reactive) Multi-Party Computation

In this section we prove the sufficient and necessary condition on the adversary structure $\mathcal{Z}$ for the existence of perfectly $\mathcal{Z}$-secure multi-party computation protocols. The sufficiency of the condition is proven by constructing an MPC protocol. The necessity is proven by an impossibility argument.

Theorem 1. *A set $\mathcal{P}$ of players can perfectly $\mathcal{Z}$-securely compute any (reactive) computation when $C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ and $C_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z})$ hold, where*

$$C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z}) \iff \forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : \; E_1 \cup E_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P}$$
$$C_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z}) \iff \forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : \; E_1 \cup A_2 \cup A_3 \cup (F_2 \cap F_3) \neq \mathcal{P}$$

The condition C_{MULT} is needed for (non-robust) multiplication. The condition C_{REC} is needed for robust reconstruction.

4.1 The MPC Protocol

The circuit C to be computed consists of input, addition, multiplication and output gates.[7] 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, multiplication and output gates, the corresponding sub-protocol Share, Mult, and $\mathsf{Reconstruct}$, respectively, is invoked. Due to the linearity of the sharing, addition (or linear) gates can be evaluated locally by the players.

The non-robustness of the used sub-protocols is addressed differently depending on the type of the gate: When in an input gate the input player does not share his input, the players just pick a default sharing of some pre-agreed default value. The reconstruction protocol of the output gate is robust under the necessary condition for MPC. The multiplication of shared values can abort (with a set $B \subseteq \mathcal{P}$ of incorrect players). If this happens, the multiplication is retried in a smaller setting, namely with the player set $\mathcal{P}' = \mathcal{P} \setminus B$ and the adversary structure $\mathcal{Z}'$ which contains only those adversary classes which are compatible with the fact that the players in B are incorrect. More precisely, first both factors are re-shared to the new setting with $\mathcal{P}'$ and $\mathcal{Z}'$, then the multiplication sub-protocol is invoked within this setting, and upon success, the resulting sharing of the product is re-shared to the original setting with $\mathcal{P}$ and $\mathcal{Z}$. This process is repeated until the multiplication succeeds, and with each repetition, the active player set $\mathcal{P}'$ becomes smaller.

For the sake of clarity, we introduce two operators on adversary structures: For a set $B \subseteq \mathcal{P}$, we denote by $\mathcal{Z}|^{B \subseteq F}$ the sub-structure of $\mathcal{Z}$ that contains 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., $\mathcal{Z}|_{\mathcal{P}'} = \{(A \cap \mathcal{P}', E \cap \mathcal{P}', F \cap \mathcal{P}') : (A, E, F) \in \mathcal{Z}\}$. As syntactic sugar, we write $\mathcal{Z}|^{B \subseteq F}_{\mathcal{P}'}$ for $\left(\mathcal{Z}|^{B \subseteq F}\right)|_{\mathcal{P}'}$.

It immediately follows from the above definitions that when the players in B have been detected to be incorrect, then the actual adversary $Z^\star$ is in $\mathcal{Z}|^{B \subseteq F}$. Furthermore, we exclude the players in B from the multiplication protocol, and the new setting is $\mathcal{P}' = \mathcal{P} \setminus B$ and $\mathcal{Z}' = \mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B}$. One can easily verify that the conditions C_{BC}, C_{MULT}, and C_{REC} hold in $(\mathcal{P} \setminus B, \mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B})$ when they hold in $(\mathcal{P}, \mathcal{Z})$, for an arbitrary $B \subseteq \mathcal{P}$. This results in the following MPC protocol:

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

1. Initialize the set of detected as incorrect players to $\mathcal{P}_\perp = \emptyset$. Set the default sharing specification $\mathcal{S} = \mathcal{S}_\mathcal{Z}$.
2. For every gate to be evaluated, do the following:
   - *Input gate for p:* Invoke Share to have p share his input according to $(\mathcal{P}, \mathcal{S})$. If Share aborts, then a default sharing of some pre-agreed default value is taken.
   - *Addition gate:* Every $p_i \in \mathcal{P}$ locally computes the sum of his respective shares.
   - *Multiplication gate:* Denote the sharings of the factors as $\langle s \rangle$ and $\langle t \rangle$, respectively, and denote the set of active players as $\mathcal{P}' = \mathcal{P} \setminus \mathcal{P}_\perp$ and the adversary structure compatible with $\mathcal{P}_\perp$ being incorrect as $\mathcal{Z}' = \mathcal{Z}|^{\mathcal{P}_\perp \subseteq F}_{\mathcal{P} \setminus \mathcal{P}_\perp}$, and the corresponding ($\mathcal{Z}'$-private) sharing specification as $\mathcal{S}' = \mathcal{S}_{\mathcal{Z}'}$. Invoke $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}, \mathcal{S}', \langle s \rangle)$ and $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}, \mathcal{S}', \langle t \rangle)$ to obtain the sharings $\langle s \rangle'$ and $\langle t \rangle'$ for $(\mathcal{P}', \mathcal{S}')$, respectively. Invoke $\mathsf{Mult}(\mathcal{P}', \mathcal{Z}', \langle s \rangle', \langle t \rangle')$ to obtain a sharing $\langle st \rangle'$ of the product, according to $(\mathcal{P}', \mathcal{S}')$. Invoke $\mathsf{Reshare}(\mathcal{P}', \mathcal{Z}', \mathcal{S}', \mathcal{S}, \langle st \rangle')$ to reshare this product according to $(\mathcal{P}, \mathcal{S})$.[a] If any of the sub-protocols aborts with set B then set $P_\perp = P_\perp \cup B$ and repeat the gate.
   - *Output gate for p:* Invoke $\mathsf{Reconstruct}$ to have the output reconstructed towards p.

[a] $\mathsf{Reshare}$ outputs a sharing according to $(\mathcal{P}', \mathcal{S})$, which is trivially also a sharing according to $(\mathcal{P}, \mathcal{S})$ since all players in $\mathcal{P} \setminus \mathcal{P}'$ are incorrect.

Lemma 11. *The above MPC protocol is perfectly $\mathcal{Z}$-secure if $C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ and $C_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z})$ hold.*

Proof (sketch). One can easily verify that the conditions in the lemma imply all conditions required in the sub-protocols, hence the security of the MPC protocol follows from the security of the sub-protocols. □

[7] 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 output.

4.2 Impossibility of MPC

In this section we prove that perfectly secure (reactive) MPC is not possible for some circuits when $C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ or $C_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ is violated. We first prove that when $C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ is violated, then even non-reactive computations cannot be securely evaluated (Lemma 12). Secondly, we prove that when $C_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ is violated, then the players in $\mathcal{P}$ cannot hold a secret joint state, which excludes the evaluation of (non-trivial) reactive circuit (Lemma 13).

Lemma 12. *If $C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ is violated, then there exist (even non-reactive) circuits which cannot be evaluated perfectly $\mathcal{Z}$-securely.*

Proof. Consider $\mathcal{P}$ and $\mathcal{Z}$ with $C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ violated, and assume for the sake of contradiction, that for every circuit C, a perfectly $\mathcal{Z}$-secure protocol exists. There exist $(A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3)\in\mathcal{Z}$ with $E_1\cup E_2\cup A_3\cup(F_1\cap F_2\cap F_3)=\mathcal{P}$. Let $F=F_1\cap F_2\cap F_3$, $\mathcal{P}'=\mathcal{P}\setminus F$, and for $i=1,2,3$, let $A_i'=A_i\setminus F$ and $E_i'=E_i\setminus F$. The alleged protocol must also be perfectly secure for the player set $\mathcal{P}'$ and the adversary structure (with only active and passive corruption) $\mathcal{Z}'=\{(A_1',E_1'),(A_2',E_2'),(A_3',E_3')\}$, because one particular strategy of the adversary is to fail-corrupt the players in F and make them crash at the very beginning of the protocol. However, for $(\mathcal{P}',\mathcal{Z}')$ perfectly secure (non-reactive) MPC protocols do not exist for all circuits, as proven in [FHM99, Thm. 1]. □

Lemma 13. *If $C_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ is violated, then the players cannot hold a secret joint state with perfect security.*

Proof. Consider $\mathcal{P}$ and $\mathcal{Z}$ with $C_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ violated, hence there exist (A_1,E_1,F_1), (A_2,E_2,F_2), $(A_3,E_3,F_3)\in\mathcal{Z}$ with $E_1\cup A_2\cup A_3\cup(F_2\cap F_3)\neq\mathcal{P}$. Wlog assume that $E_1=\{p_1\}$, $A_2=\{p_2\}$, $A_3=\{p_3\}$, and $F_2=F_3=\{p_4\}$. We denote the view of p_i as v_i. For the sake of contradiction, assume that these views define a secret joint state v. Privacy requires that v_1 does not determine v, hence there exists a different state $v'\neq v$ which could be represented by the views (v_1,v_2',v_3',v_4'). Now consider the following two cases: (i) The secret state is v, and the adversary corrupts (A_2,E_2,F_2) and makes p_4 crash and p_2 take a random view, which (with perhaps negligible probability) could be v_2'. (ii) The secret state is v', and the adversary corrupts (A_3,E_3,F_3) and makes p_4 crash and p_3 take a random view, which (with perhaps negligible probability) could be v_3. In both cases, the views of the players are $(v_1,v_2',v_3,\perp)$, but the joint state is once v and once $v'\neq v$, contradicting perfect security. □

5 Secure Function Evaluation

In this section we prove the sufficient and necessary condition on the adversary structure $\mathcal{Z}$ for the existence of perfectly $\mathcal{Z}$-secure function evaluation protocols. The sufficiency of the condition is proven by constructing an SFE protocol, and necessity is proven by an impossibility argument. Note that the condition for SFE is weaker than the condition for MPC.

Theorem 2. *A set $\mathcal{P}$ of players can perfectly $\mathcal{Z}$-securely compute any function if and only if $C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and C_{NREC} hold, where*

$$C_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})\iff\forall(A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3)\in\mathcal{Z}:\ E_1\cup E_2\cup A_3\cup(F_1\cap F_2\cap F_3)\neq\mathcal{P},$$
$$C_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})\iff\textit{there exists an ordering }\big((A_1,E_1,F_1),\ldots,(A_m,E_m,F_m)\big)\textit{ of }\overline{\mathcal{Z}}\textit{ s.t.}^{8}$$
$$\forall i,j,k\in\{1,\ldots,m\},i\leq k:\ E_k\cup A_i\cup A_j\cup(F_i\cap F_j)\neq\mathcal{P}.$$

The condition C_{MULT} is needed for (non-robust) multiplication. The condition C_{NREC} is needed for non-robust reconstruction. Essentially, the latter condition allows for a reconstruction protocol in which the actual adversary gets information on the output only once it cannot disturb the protocol anymore.

[8] 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 $\mathcal{Z}$.

5.1 The SFE Protocol

Our SFE protocol follows the standard approach of SFE protocols, namely to first secret-share all inputs, then to evaluate the circuit gate by gate, then to reconstruct the output. However, the protocol employs sharings which are not robustly reconstructible. This means that the adversary can break down the computation in such a way that all sharings are lost. As the circuit is non-reactive, we can handle such an abortion by repeating the whole protocol, including the input stage. The correct players will give the same inputs in every iteration, but the adversary might give different inputs. However, in a failed iteration, the adversary does not get any information about any secrets (more precisely, the adversary could perfectly simulate all messages received within a failed iteration already beforehand), so the inputs chosen by the adversary in the successful iteration are independent of the other players' inputs.

Termination is guaranteed by the fact that whenever an iteration aborts, then a non-empty set $B \subseteq \mathcal{P}$ of incorrect players is identified, and the next iteration will proceed without these players. Hence the number of iterations is bounded by n.

The delicate task is the output protocol. For simplicity, we describe the protocol only for a single public output s; however, it naturally extends to a vector $\vec{s}$ of several public outputs, which then can be extended to capture private outputs with standard techniques (the output player inputs a one-time pad used for perfectly blinding the private element of the output vector).

The intuition of the output protocol is as follows: First observe that in our sharing, the privacy against each adversary is protected by a particular summand. More precisely, for every adversary class $(A_k, E_k, F_k) \in \mathcal{Z}$ there exists a summand s_k which is given only to the players in $S_k \in \mathcal{S}$ with $S_k \cap E_k = \emptyset$ (we even have $S_k = \mathcal{P} \setminus E_k$). As long as this summand is not published, an adversary of class (A_k, E_k, F_k) does not obtain information about the output (from the point of view of the adversary, 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. the protocol $\mathsf{PublicAnnounce}$ aborts), then a set $B \subseteq \mathcal{P}$ of incorrect players is identified. The information that the players in B are incorrect leaks information about the actual adversary $(A^\star, E^\star, F^\star)$, namely that $B \subseteq F^\star$. The key idea of the output protocol is to publish the summands in such an order that whenever $\mathsf{PublicAnnounce}$ aborts with B, then the information that the players in B are incorrect excludes the possibility that the actual adversary is from a class whose summand has already being published. In other words: Whenever an adversary of class (A_i, E_i, F_i) could potentially abort the announcing of the summand s_k associated with the adversary class (A_k, E_k, F_k), then the summand s_k must be announced strictly before the summand s_i is announced.

Let $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ denote an ordering of the maximum structure $\overline{\mathcal{Z}}$ satisfying

$$\forall 1 \leq i, j, k \leq m, i \leq k : E_k \cup A_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P},$$

and let S denote the induced sharing specification $\mathcal{S} = (S_1, \ldots, S_m)$ with $S_k = \mathcal{P} \setminus E_k$. Then the following protocol perfectly $\mathcal{Z}$-securely publicly reconstructs a sharing $\langle s \rangle$ according to $\mathcal{S}$, or aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Privacy of the protocol is guaranteed under the assumption that those summands of $\langle s \rangle$ that are unknown to the adversary are uniformly distributed. This is the case 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) $\mathsf{PublicAnnounce}(\mathcal{P}, \mathcal{Z}, S_k, s_k)$ is invoked to have the correct summand s_k published.
   (b) If $\mathsf{PublicAnnounce}$ aborts with B, then $\mathsf{OutputGeneration}$ *immediately* aborts with B.
2. Every $p_j \in \mathcal{P}$ (locally) computes $s = \sum_{k=1}^{m} s_k$ and outputs s.

Lemma 14. *Assuming that $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification constructed as explained, $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, and $\forall S_k \in \mathcal{S}, (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z} : S_k \not\subseteq A_1 \cup A_2$, and $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$ with the property that those summands that are unknown to the adversary are randomly chosen, then the protocol* $\mathsf{OutputGeneration}$ *either publicly reconstructs s, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. If* $\mathsf{OutputGeneration}$ *aborts, then the protocol does not leak any information on s to the actual adversary.*

Proof. First observe that the pre-conditions of PublicAnnounce are satisfied. Second, observe that by construction of $\mathcal{S}$, we have $\forall i, j, k \in \{1, \ldots, m\}, i \leq k : (\mathcal{P} \setminus S_k) \cup A_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P}$. Now assume that the invocation of PublicAnnounce$(\mathcal{P}, \mathcal{Z}, S_k, s_k)$ aborts with $B \subseteq \mathcal{P}$. It follows from Lemma 7 that the actual adversary $(A^\star, E^\star, F^\star)$ satisfies the property that there exists $(A_j, E_j, F_j) \in \mathcal{Z}$ such that $S_k \subseteq A^\star \cup A_j \cup (F^\star \cap F_j)$. By the construction of $\mathcal{S}$, no adversary class $(A_i, E_i, F_i) \in \mathcal{Z}$ with $i \leq k$ satisfies this condition, hence the summand associated with actual adversary has not yet been announced. □

With this protocol, the SFE protocol can be constructed easily:

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

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:* Every $p_i \in \mathcal{P}$ locally computes the sum of his respective shares.
   - *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 subprotocols aborts with B, then set $\mathcal{P} \leftarrow \mathcal{P} \setminus B$, and set $\mathcal{Z}$ to the adversary structure which is compatible with B being incorrect, i.e., $\mathcal{Z} \leftarrow \mathcal{Z}|_{\mathcal{P}'}^{B \subseteq F}$, 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 15. *The above SFE protocol is perfectly $\mathcal{Z}$-secure if $C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ and $C_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ hold.*

Proof (sketch). One can easily verify that the conditions in the lemma imply all conditions required in the sub-protocols, hence the security of the SFE protocol follows from the security of the sub-protocols.
Special care needs to be taken for the fact that the adversary can abort the protocol and provoke repetitions. Termination of this process is obvious, as in every repetition the player set shrinks. Also correctness is straightforward. Privacy is argued as follows: The adversary can perfectly simulate his view in every iteration which aborts (even without knowing the public output), hence his capability to abort an iteration does not give him any additional power. □

5.2 Impossibility of SFE

In this section we prove that perfectly $\mathcal{Z}$-secure SFE is not possible for some circuits when $C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ or $C_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ is violated. The necessity for $C_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ follows immediately from Lemma 12. It remains to show that $C_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ is necessary:

Lemma 16. *If $C_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ is violated, then there exist functions which cannot be evaluated perfectly $\mathcal{Z}$-securely.*

Proof. Consider $\mathcal{P}$ and $\mathcal{Z}$ with $C_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ violated, i.e., for every ordering $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ of $\overline{\mathcal{Z}}$ there exists $i, j, k \in \{1, \ldots, m\}$ such that $i \leq k$ and $E_k \cup A_i \cup A_j \cup (F_i \cap F_j) = \mathcal{P}$. Consider the identity function, where every player $p_i \in \mathcal{P}$ inputs some value x_i, and the public output is the vector $(x_1, \ldots, x_n)$. For the sake of contradiction, assume that there exists a perfectly $\mathcal{Z}$-secure SFE protocol for this function. This protocol implicitly defines for every set $L \subseteq \mathcal{P}$ the protocol round in which the players in L obtain full joint information about the output. We denote the index of this round as $\phi(L)$, i.e., the joint view of the players in L in round $\phi(L)$ gives full information on $(x_1, \ldots, x_n)$, but their joint view in round $\phi(L) - 1$ does not give full information. The function ϕ implies an ordering $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ on the adversary classes in $\overline{\mathcal{Z}}$ such that for every $1 \leq i \leq k \leq m : \phi(E_i) \leq \phi(E_k)$. Denote by i, j, k those indices that satisfy $i \leq k$ and $E_k \cup A_i \cup A_j \cup (F_i \cap F_j) = \mathcal{P}$ (which are assumed to exist for contradiction). The adversary corrupts

(A_i, E_i, F_i) and behaves as follows: Up to round $\phi(E_i) - 1$, the adversary lets the corrupted players behave correctly. In round $\phi(E_i)$, the adversary crashes the players in $F_i \cap F_j$, and has the players in $A_i \setminus (F_i \cap F_j)$ send random values (also in all subsequent rounds). Still, the adversary obtains full information on the output in round $\phi(E_i)$ (he knows all correct messages that were sent, respectively should have been sent to the players in E_i). However, the players in E_k do not have full information *before* round $\phi(E_k) \geq \phi(E_i)$. Hence these players cannot with certainty distinguish the current situation from the situation when the output vector would be different, the players in class (A_j, E_j, F_j) would be corrupted, those in $F_j \cap F_i$ would be crashed, and those in $A_j \setminus (F_j \cap F_i)$ would send random messages. Hence the adversary has obtained full information about the output vector, but some uncorrupted players do not, contradicting perfect security. □

6 Separation and Conclusions

We have considered an adversary whose corruption capability is described by a collection $\mathcal{Z}$ of adversary classes (A, E, F), where the adversary may actively corrupt the players in A, passively corrupt the players in E, and fail-corrupt the players in F. This model unifies all corruption models considered in the literature, as they are all special cases, either in terms that not all corruption types were considered, or in terms that only threshold corruption was considered.

For this general adversary model, we have derived exact conditions for the existence of perfectly secure multi-party computation (MPC) and secure function evaluation (SFE). It turned out that the condition for SFE is strictly weaker than the condition for MPC. In fact, there are simple adversary structures for which perfectly secure SFE is possible, but perfectly secure MPC and verifiable secret sharing are not possible. This separation does not show up in the restricted models considered so far. The following theorem states this separation. It follows immediately from the separating example in the introduction with $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ and $\overline{\mathcal{Z}} = \big\{(\emptyset, \{p_1\}, \emptyset), (\{p_2\}, \{p_2\}, \{p_2, p_4\}), (\{p_3\}, \{p_3\}, \{p_3, p_4\})\big\}$.

Theorem 3. *Perfectly secure MPC and SFE separate, i.e., there exist $\mathcal{P}$ and $\mathcal{Z}$ such that perfectly $\mathcal{Z}$-secure SFE among the players in $\mathcal{P}$ is possible, whereas perfectly $\mathcal{Z}$-secure MPC is not.*

References

[AFM99] Bernd Altmann, Matthias Fitzi, and Ueli Maurer. Byzantine agreement secure against general adversaries in the dual failure model. In *Distributed Computing — DISC '99*, volume 1693 of *LNCS*, pages 123–137, 1999.

[Bea91a] Donald Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *LNCS*, pages 377–391, 1991.

[Bea91b] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):370–381, 1991.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on the Theory of Computing — STOC '88*, pages 1–10, 1988.

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

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *ACM Symposium on the Theory of Computing — STOC '88*, pages 11–19, 1988.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *LNCS*, pages 74–92, 2000.

[DS82] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *ACM Symposium on the Theory of Computing — STOC '82*, pages 401–407, 1982.

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multi-party computation. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *LNCS*, pages 121–136, 1998. Corrected version is available online.

[FHM99] Matthias Fitzi, Martin Hirt, and Ueli Maurer. General adversaries in unconditional multi-party computation. In *Advances in Cryptology — ASIACRYPT '99*, volume 1716 of *LNCS*, pages 232–246, 1999.

[FM98] Matthias Fitzi and Ueli Maurer. Efficient Byzantine agreement secure against general adversaries. In *Distributed Computing — DISC '98*, volume 1499 of *LNCS*, pages 134–148, 1998.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *ACM Symposium on the Theory of Computing — STOC '87*, pages 218–229, 1987.

[GP92] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In *Distributed Algorithms, 6th International Workshop — WDAG '92*, volume 647 of *LNCS*, pages 153–165, 1992.

[HM97] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *ACM Symposium on Principles of Distributed Computing — PODC '97*, pages 25–34, 1997. Full version appeared in *Journal of Cryptology*,13(1): 31-60, 2000.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology — CRYPTO 2006*, volume 4117 of *LNCS*, pages 483–500, 2006.

[LF82] Leslie Lamport and Michael J. Fischer. Byzantine generals and transaction commit protocols. Technical Report Opus 62, SRI International (Menlo Park CA), TR, 1982.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[Mau02] Ueli Maurer. Secure multi-party computation made simple. In *Third Conference on Security in Communication Networks — SCN 2002*, volume 2576 of *LNCS*, pages 14–28, 2002. Full version appeared in *Discrete Applied Mathematics*, 154(2):370-381, 2006.

[MP91] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, 1991.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *LNCS*, pages 392–404, 1991.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *ACM Symposium on the Theory of Computing — STOC '89*, pages 73–85, 1989.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *IEEE Symposium on the Foundations of Computer Science — FOCS '82*, pages 160–164, 1982.

Appendix

A Protocol Reshare

The following protocol allows the players in $\mathcal{P}$ to $\mathcal{Z}$-securely reshare a sharing of $\langle v \rangle$ according to sharing specification $\mathcal{S}$ to the new sharing specification $\mathcal{S}'$.

Protocol Reshare$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{S}', \langle s \rangle)$

1. For every $S_k \in \mathcal{S}$, the following steps are executed:
   (a) Every $p_i \in S_k$ invokes $\mathsf{Share}(\mathcal{P}, \mathcal{Z}, \mathcal{S}', p_i, s_k)$; denote the resulting sharing as $\langle s_k^{(i)} \rangle$.
   (b) Let p_i denote the player with the smallest index in S_k. For every $p_j \in S_k$, the difference $\langle s_k^{(j)} \rangle - \langle s_k^{(i)} \rangle$ is computed and, by invoking PublicReconstruct, publicly reconstructed.
   (c) If all differences are 0, then the sharing $\langle s_k^{(i)} \rangle$ of p_i is adopted as sharing of s_k, i.e., $\langle s_k \rangle = \langle s_k^{(i)} \rangle$. Otherwise (i.e., some difference is non-zero), PublicAnnounce is invoked to have s_k announced, and a default sharing $\langle s_k \rangle$ of s_k according to $\mathcal{S}'$ is created.
2. Every $p_i \in \mathcal{P}$ (locally) computes the sum of his shares of all summands s_k.
3. If any of the invoked sub-protocols aborts with B, then also Reshare aborts with B.

Lemma 10. *Assuming that $\mathcal{S}'$ is a $\mathcal{Z}$-private sharing specification, $\langle s \rangle$ is a consistent sharing according to $\mathcal{S}$, $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, and $\forall S_k \in \mathcal{S}, S'_k \in \mathcal{S}', (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z}: \ (S_k \not\subseteq A_1 \cup A_2) \wedge (S'_k \not\subseteq A_1 \cup A_2)$, the protocol* Reshare$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{S}', \langle s \rangle)$ *has the following properties. Correctness: It either outputs a sharing of s according to $(\mathcal{P}, \mathcal{S}')$ or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. Secrecy: No information on the inputs (i.e., on $\langle s \rangle$) leaks to the adversary.*

Proof. Correctness: The conditions in the lemma are sufficient for all the invoked sub-protocols (Share,PublicReconstruct,PublicAnnounce). The condition $\forall S_k \in \mathcal{S}, \forall (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z}: \ S_k \not\subseteq A_1 \cup A_2$ implies that $\forall S_k \in \mathcal{S}, \forall (A, \cdot, \cdot) \in \mathcal{Z}: S_k \not\subseteq A$, which ensures that every s_k is known to at least one player p_i who is not actively corrupted; hence if no invocation of Share aborts and all differences are zero, then the shared values are correct. Privacy: Due to the security of Share, the invocations of Share do not leak information to the adversary. Furthermore, PublicAnnounce is only invoked on the summand s_k when two players in S_k contradict each other; at least one of these players is actively corrupted, hence the adversary already knows s_k before PublicAnnounce is invoked. □

B Proofs of Lemmata

Lemma 2 (Crash Detection). *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, then the protocol* CDP$(\mathcal{P}, \mathcal{Z}, p)$ *has the following properties: Consistency: The (correct) players agree on the output. Correctness: If p is correct until the end of* CDP, *then every (correct) player outputs "alive" and if p has crashed* before *the invocation of* CDP, *then every (correct) player outputs "crashed".*

Proof. Correctness: When p is correct, then every (correct) $p_j \in \mathcal{P}$ sets $b_j = 1$, and by definition of consensus, all correct players decide on 1 and output "alive". When p has crashed before CDP is invoked, then every correct $p_j \in \mathcal{P}$ sets $b_j = 0$, and hence all correct players output "crashed". Consistency: As the output is decided by using consensus, the output of all correct players is identical. □

Lemma 3 (Strong Broadcast). *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds, then the protocol* StrongBroadcast$(\mathcal{P}, \mathcal{Z}, p, v)$ *has the following properties: Consistency: All (correct) players output the same value v'. Correctness: If the sender p is correct, then $v' = v$; if p crashed* before *the invocation of the protocol, then $v' = \perp$; if p crashes during the protocol, then $v' \in \{v, \perp\}$.*

Proof. Consistency follows immediately from the consistency property of Broadcast and the consistency property of CDP. For correctness we consider 3 cases: (a)If the sender p is correct through the whole protocol, then the consistency property of Broadcast implies that for all correct p_j 's, $v_j = v$ and the correctness property of CDP implies that all correct players will output "alive" in CDP, hence they will all output v in StrongBroadcast. (b) If p has already crashed *before* the invocation of StrongBroadcast, then this is detected in Step 2 (by CDP) and the protocol outputs $\perp$. (c) If p crashes during the protocol but is correct up to that point, then either this is detected in Step 2 and the protocol outputs $\perp$, or p is still alive at the beginning of Step 2 and has correctly broadcast his input v. Since, when p is not actively-corrupted one of the above 3 cases must hold, the output of StrongBroadcast for such a p is always in $\{v, \perp\}$. □

Lemma 4 (Share). *If $C_{\mathrm{BC}}(\mathcal{P}, \mathcal{Z})$ holds and $\mathcal{S}$ is a $\mathcal{Z}$-private sharing specification, then the protocol* Share $(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p, s)$ *has the following properties. Correctness:* Share *either outputs a consistent sharing of some s', where $s' = s$ unless the dealer is actively corrupted, or it aborts with $B = \{p\}$; it does not abort if p is correct. Secrecy: No information on s leaks to the adversary.*

Proof. Correctness: The consistency of the sharing is guaranteed because correct players either hold the same value for a common summand, or they complain and get a consistent value for the summand by strong broadcast. Because all sent and broadcasted summands are s_k such that $s = \sum s_k$ it is clear that the shared value is s when the dealer is correct. Lastly, the protocol only aborts when the dealer is incorrect in an invocation of strong broadcast. Secrecy: Because $\mathcal{S}$ is $\mathcal{Z}$-private we know that the summands of corrupted players do not reveal information on s. On the other hand, the dealer only broadcasts summands for which a complaint is broadcast, i.e., two players (claim to) have different values for that summand. This only happens when the dealer or one of the disputing players is actively corrupted, or when the dealer has crashed. In the first case, the adversary is entitled to know the summand, and in the second case, the summand will not be broadcasted (the dealer is crashed). □

C Implications Among the Conditions

The following figure summarizes the implications between the conditions: An arrow from Condition 1 to Condition 2 means that 1 implies 2; an erased arrow means that there is an example (i.e., an adversary structure $\mathcal{Z}$) that strictly separates the two conditions.