Realistic Failures in Secure Multi-party Computation*

Vassilis Zikas, Sarah Hauser, and Ueli Maurer

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

Abstract. In secure multi-party computation, the different ways in which the adversary can control the corrupted players are described by different corruption types. The three most common corruption types are active corruption (the adversary has full control over the corrupted player), passive corruption (the adversary sees what the corrupted player sees) and fail-corruption (the adversary can force the corrupted player to crash *irrevocably*). Because fail-corruption is inadequate for modeling recoverable failures, the so-called omission corruption was proposed and studied mainly in the context of Byzantine Agreement (BA). It allows the adversary to selectively block messages sent from and to the corrupted player, but without actually seeing the message.

In this paper we propose a modular study of omission failures in MPC, by introducing the notions of *send-omission* (the adversary can selectively block outgoing messages) and *receive-omission* (the adversary can selectively block incoming messages) corruption. We provide security definitions for protocols tolerating a threshold adversary who can actively, receive-omission, and send-omission corrupt up to t_a, t_ρ, and t_σ players, respectively. We show that the condition $3t_a + t_\rho + t_\sigma < n$ is necessary and sufficient for perfectly secure MPC tolerating such an adversary. Along the way we provide perfectly secure protocols for BA under the same bound. As an implication of our results, we show that an adversary who actively corrupts up to t_a players and omission corrupts (according to the already existing notion) up to t_ω players can be tolerated for perfectly secure MPC if $3t_a + 2t_\omega < n$. This significantly improves a result by Koo in TCC 2006.

1 Introduction

In secure multi-party computation (MPC) n players $p_1, \ldots, p_n$ wish to securely compute a function of their inputs. The computation should be secure, in the sense that the output is correct and the privacy of the players' inputs is not violated. The security should be guaranteed even when some of the players misbehave. The misbehavior of players is modeled by assuming a central adversary who corrupts players. The most typical corruption types are active corruption (the adversary has full control over the corrupted player), passive corruption (the adversary sees whatever the player sees), and fail-corruption (the adversary can make the player crash *irrevocably*).

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-00457-5_36

* This research was partially supported by the Swiss National Science Foundation (SNF), project no. 200020-113700/1. The full version of this paper is available at http://www.crypto.ethz.ch/pubs/ZiHaMa09

O. Reingold (Ed.): TCC 2009, LNCS 5444, pp. 274–293, 2009.

The study of MPC was initiated 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.

One of the most studied sub-problems of secure multi-party computation is Byzantine Agreement (BA). BA comes in two flavors, namely *consensus* and *broadcast*. Informally, consensus guarantees that n players, each holding an input, can agree on a common output without destroying pre-agreement. On the other hand, broadcast allows a dedicated player to consistently send his input to every player. BA serves as an important tool for the design of multi-party protocols.

Failures in MPC. For motivating the different corruption-types one typically thinks of MPC as each player running his protocol on his (local) computer, where the computers can communicate over some network (e.g., the Internet). Passive and active corruption correspond, for example, to (the adversary) planting a spyware or a virus, respectively, to the player's computer. Fail-corruption, however, can be criticized as being not so realistic due to the requirement that the crash is irrevocable. Indeed, in real-world scenarios computer-crashes are not irrevocable and are usually fixed soon after they are discovered, e.g., by replacing the computer.

Corruption types modeling more realistic failures than irrevocable computer-crashes have been studied in the literature. An example is the so-called *omission corruption* which allows the adversary to selectively block messages sent or received by the corrupted player, but without seeing the actual message. Omission corruption models failures that are apparent in many real-world applications, e.g., a computer which might lose messages while being restarted due to a hang of the operating system. It also models failures or temporary unavailability of the communication network, e.g., a router's buffer overflow, or instability of the links due to a thunderstorm. Partial asynchronity of the network, i.e., the adversary causing unexpected delays on messages sent from and to certain players, can also be modeled.

Omission corruption has been primarily studied in the context of fault-tolerant consensus [Had85, PT86, Ray02, PR03] and, recently, also in MPC [Koo06].

Summary of known results. In the seminal papers solving the general MPC problem, 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.

A similar development as in MPC can be observed in the area of Byzantine agreement protocols [LSP82, DS82, LF82, MP91, GP92, FM98].

The first to consider omission corruption were Perry and Tueg [PT86]. They considered a threshold adversary who can omission corrupt up to t players and showed that BA tolerating this adversary is possible if and only if $t < n$. However their consistency-guarantee is limited to the outputs of uncorrupted players, i.e., omission corrupted players are allowed to output arbitrary values. Raynal and Parvedy [Ray02, PR03] proved that if we require omission corrupted players to output either the correct value (i.e., consistent with the output of uncorrupted players) or no value, then consensus is possible if and only if $2t < n$.

In the context of general MPC, omission corruption was first studied, in combination with active corruption, by Koo [Koo06]. He considered a threshold adversary who can actively corrupt up to t_a players and, simultaneously, omission corrupt up to t_ω players,[1] and proved that the conditions $3t_a + 2t_\omega < n$ and $3t_a + 4t_\omega < n$ are sufficient for perfectly secure consensus and general MPC, respectively. However, as we show in Section 9, the condition $3t_a + 4t_\omega < n$ is far from optimal.

Our Contributions. We propose a modular study of realistic failures in multi-party computation, by introducing the notions of *send-omission* and *receive-omission* corruption. As the names suggest, send-omission (resp. receive-omission) corruption allows the adversary to selectively block only outgoing (resp. only incoming) messages of the corrupted player, but without seeing the messages (this is consistent with the existent omission-corruption literature). Note that a player who is omission corrupted according to the definitions of [PT86, Ray02, PR03, Koo06] can be thought of as a player who is both send- and receive-omission corrupted at the same time; for clarity we refer to this type of corruption as *full-omission* corruption.

We provide security definitions for the model where the adversary can actively, send-omission, and receive-omission corrupt players, simultaneously. We show that in this model, an adversary who can actively, receive-omission, and send-omission corrupt up to t_a, t_ρ, and t_σ players, respectively, can be tolerated for perfectly secure MPC if and only if $3t_a + t_\rho + t_\sigma < n$. Along the way, we also construct BA primitives for the same bound. Our bound implies that the condition $3t_a + 2t_\omega < n$ is sufficient for perfectly secure MPC.

The novelty of our approach is that, unlike past results on fault-tolerant MPC, we primarily deal with the omissions on the network-level instead of internally in the protocol. In particular, using the paradigm of layered communication (e.g., the OSI-model), first we engineer the actual network to build a new network-layer with better security guarantees, and then we design protocols in which the players communicate over this higher network-layer. This approach leads to simpler and more intuitive protocols. For the construction of our main protocol we also use ideas from the *player-elimination* technique [HMP00].

Outline of this paper. In Section 2 we define the model and introduce some notation. In Section 3 we discuss the security definitions and prove an impossibility result. In Sections 4 and 5 we show how to get an authenticated network with strong security guarantees and then build BA protocols over it. In Section 6 we provide tools that will

[1] In [Koo06], omission corrupted players are called *constrained* and actively corrupted are called *corrupted*.

be used as building blocks for the construction of the SFE and MPC protocols:[2] these protocols are described in Sections 7 and 8, respectively. In Section 9 we look at the case of full-omission corruption.

2 The Model

We consider the standard secure-channels model introduced in [BGW88, CCD88]: The players in $\mathcal{P} = \{p_1, \ldots, p_n\}$ are connected by a complete network of bilateral secure channels. The communication is synchronous, i.e., all players have synchronized clocks and there is a known upper bound on the delay of the network. The computation is described as an arithmetic circuit over some finite field $\mathbb{F}$, consisting of addition (or linear) and multiplication gates.

We look at the case of *perfect* security, i.e., information-theoretic without error probability. 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 [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 (privacy), and that he cannot influence the outputs other than by choosing the inputs of the corrupted players (correctness).

We consider a rushing[4] threshold adversary who can actively, receive-omission, and send-omission corrupt up to t_a, t_ρ, and t_σ players, respectively. The adversary chooses the players to corrupt non-adaptively, i.e., before the beginning of the protocol.[5]

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.

Every $p_i \in \mathcal{P}$ can be in one of the following two internal states: *alive* or *zombie*. At the beginning of the computation every player is alive, which means that he correctly executes all the protocol instructions (unless he is actively corrupted). If p_i realizes that he is receive-omission corrupted, e.g., by receiving fewer messages than what he should in some round, then p_i sets his internal state to zombie (we say that p_i becomes a zombie). Once the state is set to zombie it never switches back. A zombie behaves in the

[2] SFE stands for Secure Function Evaluation, i.e., multi-party computation of *non-reactive* functionalities.

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

[4] A *rushing* adversary is an adversary who, in each round of the protocol, first sees all the messages sent to actively corrupted players in this round and then decides how the corrupted players should behave in this round.

[5] In contrast, an *adaptive* adversary can corrupt more and more players during the protocol execution, subject only to the constraint that the number of corrupted players of each type is upper-bounded by the corresponding threshold. We do not consider the adaptive setting in this paper, but our results could be generalized to it.

protocols as a player who has crashed, i.e., sends and receives no messages and has no outputs. However, there are two conceptual differences between zombies and crashed players: (1) Being a zombie is a self-imposed state and corresponds to a correct behavior, i.e., players become zombies when the protocol (and not the adversary) instructs them to; (2) zombie-players are "aware of the actual time", as they have clocks which are synchronized with the clocks of the alive players; this will be useful in the context of reactive computation (Section 8) where time plays an important role.

The sets $\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, $\mathcal{SR}$, and $\mathcal{H}$. To simplify the description we denote the sets of actively, send-omission only, receive-omission only, and full-omission[6] (but not actively) corrupted players by $\mathcal{A}$, $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{SR}$, respectively, and the set of uncorrupted players by $\mathcal{H}$ ($\mathcal{H}$ stands for "honest"). Note that these sets are a partition of the player set $\mathcal{P}$, they are not known to the players and appear only in the security analysis.

3 Security Definition

Intuitively, the security definition for our model should not allow the adversary to do more with send- and receive-omission corrupted players than to decide which of them give input to and receive output from the computation, respectively. The strongest security one can hope for is to require that the adversary's decision is taken independently of the inputs of non actively corrupted players and before seeing the outputs of actively corrupted players. More precisely one would be interested in securely realizing the functionality STRONG SFE (see below).[7]

STRONG SFE - IDEAL MODEL. *Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is $f(\cdot)$. The adversary decides which of the send-omission (resp. receive-omission) corrupted players give input to (resp. receive output from) the trusted party* before *seeing the outputs of actively corrupted players.*

1. Every $p_i \in \mathcal{H} \cup \mathcal{R}$ sends his input to the trusted party (TP). Actively corrupted players might send TP arbitrary inputs as instructed by the adversary. For each $p_i \in \mathcal{SR} \cup \mathcal{S}$ the adversary decides (without seeing p_i's input) whether p_i sends TP his input or a default value from $\mathbb{F}$ (e.g., 0). TP denotes the received values by $x'_1, \ldots, x'_n$.
2. TP computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ (if f is randomized then TP internally generates the necessary random coins). TP asks the adversary which of the players $p_i \in \mathcal{R} \cup \mathcal{SR}$ should receive their output y_i (without revealing any information about y_i).
3. For each $p_i \in \mathcal{H} \cup \mathcal{S} \cup \mathcal{A}$, TP sends y_i to p_i. For each $p_i \in \mathcal{R} \cup \mathcal{SR}$, TP sends y_i to p_i if the adversary allowed that p_i receives output in the previous step, otherwise TP sends nothing to p_i.

[6] Recall that a full-omission corrupted player is one who is both send- and receive-omission corrupted at the same time.

[7] We assume that the reader is familiar with the ideal-world/real-world paradigm for defining security of multi-party protocols [Bea91a, MR91, Can00, DM00, BPW03].

We say that a protocol Π *strongly* (t_a, t_ρ, t_σ)-securely evaluates the function f if it securely realizes the functionality STRONG SFE in the presence of an adversary who can actively, receive-omission, and send-omission corrupt up to t_a, t_ρ, and t_σ players, respectively.

Unfortunately, as stated in the following lemma, when the adversary is rushing then for any non-trivial choice for t_a and t_ρ there exist functions which cannot be perfectly strongly (t_a, t_ρ, t_σ)-securely evaluated. In fact our impossibility result is inherent in any setting where we have a threshold adversary with active (or even just passive) and receive-omission corruption, simultaneously. In particular it also applies to the (non-adaptive) case of active and full-omission corruption [Koo06].[8] The idea is the following: the adversary might, with non-zero probability, corrupt the player p_i who is the first (or among the first) to get the output, e.g., by randomly choosing whom to corrupt. In this case, as she is rushing, she can decide, depending on the output, whether the receive-omission corrupted players get full information on the output or not. However, the simulator has to take this decision without seeing the outputs of corrupted players, and hence he is not able to perfectly simulate this behavior. Due to space restrictions the proof of the lemma is deleted from this extended abstract.

Lemma 1. *If $t_a > 0$ and $t_\rho > 0$ and the adversary is rushing, then there exist functions which cannot be perfectly strongly $(t_a, t_\rho, \cdot)$-securely evaluated. The statement holds even when we have passive instead of active corruption.*

We relax the definition of the functionality to allow the adversary to decide which receive-omission corrupted players receive output, even after having seen the outputs of actively corrupted players (and possibly depending on those outputs). Our relaxation is minimal as Lemma 1 suggests. We call the resulting functionality SFE (see below).

SFE – IDEAL MODEL. *Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is $f(\cdot)$. The adversary decides which of the receive-omission corrupted players receive output from the trusted party* after *receiving the outputs of actively corrupted players.*

1. Every $p_i \in \mathcal{H} \cup \mathcal{R}$ sends his input to the trusted party (TP). Actively corrupted players might send TP arbitrary inputs as instructed by the adversary. For each $p_i \in \mathcal{SR} \cup \mathcal{S}$ the adversary decides (without seeing p_i's input) whether p_i sends TP his input or a default value from $\mathbb{F}$ (e.g., 0). TP denotes the received values by $x'_1, \ldots, x'_n$.
2. TP computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ (if f is randomized then TP internally generates the necessary random coins). For each $p_i \in \mathcal{H} \cup \mathcal{S} \cup \mathcal{A}$, TP sends y_i to p_i.
3. For $p_i \in \mathcal{R} \cup \mathcal{SR}$, TP asks the adversary if p_i should receive his output y_i (without revealing any information about y_i), if the answer is yes then TP sends y_i to p_i, otherwise it sends nothing to p_i.

[8] In [Koo06] the assumed adversary is also rushing and the (non-adaptive) ideal-world functionality requires the adversary to decide which omission corrupted players receive output before seeing the outputs of actively corrupted players.

Definition 1. *We say that a protocol Π (t_a, t_ρ, t_σ)-*securely *evaluates the function f if Π securely realizes the functionality* SFE *in the presence of an adversary who can actively, receive-omission, and send-omission corrupt up to t_a, t_ρ, and t_σ players, respectively.*

4 Engineering the Network – Authenticated Channels

A source of difficulties in designing protocols tolerating both active cheaters and omissions is that a player p_j who receives $\perp$ when expecting a message from a player p_i cannot decide whether p_i is send-omission or actively corrupted, or himself (i.e., p_j) is receive-omission corrupted. In [Koo06] the following straight-forward approach was taken in order to overcome this difficulty in the context of p_i sharing a secret: Every player complains when he received no share from the dealer p_i. If more players complain than the number of potentially corrupted players, p_i is disqualified. Otherwise, the players who did not complain pairwise check the consistency of their shares (as in [BGW88, FHM98]), where inconsistencies are publicly reported and resolved by the dealer. This approach, however, leads to thresholds on the number of actively and (full) omission corrupted players which are far from optimal, as discussed in the introduction.

Our approach is different. We deal with this difficulty outside the protocol, on the network level. In particular, using the paradigm of layered communication (e.g., the OSI-model), first we engineer the actual network to get a new network-layer with stronger guarantees, and then we invoke the actual protocol over this layer.

The protocol which is used to build the new network-layer is called FixReceive. It works on the channels of the actual network (the lowest layer), i.e., the secure channels with omissions, and builds on top of them a network of *authenticated* channels (the higher layer), where for any receive-omission corrupted p_i the adversary has to choose *either* to allow p_i to receive all messages that are sent to him *or* to let p_i know that he is receive-omission corrupted. More precisely, FixReceive guarantees that when some p_i sends a message x to a receive-omission corrupted p_j then either p_j receives it, as if he were uncorrupted, or p_j finds out that he is receive-omission corrupted (and becomes a zombie). If p_j becomes a zombie in FixReceive then he notifies every $p_k \in \mathcal{P}$ about this by sending a bilateral message; this information will be used by the players in future invocations of FixReceive. The protocol FixReceive is described in the following. For the proof of the lemma we refer to the full version of this paper.

Protocol FixReceive $(\mathcal{P}, t_a, t_\rho, t_\sigma, p_i, p_j, x)$

1. p_i sends his input x to every $p_k \in \mathcal{P}$.
2. Each $p_k \in \mathcal{P}$ forwards x to p_j (if p_k received no value, he sends a special symbol "n/v" to p_j); p_j denotes the received value as x_k (if p_k has become a zombie in the past then p_j sets $x_k =$ "n/v").
3. If $|\{p_k : x_k \in \mathbb{F} \cup \{\text{"n/v"}\}\}| < n - t_a - t_\sigma$ then p_j becomes zombie (and notifies all players). Otherwise, if there exists $x' \notin \{\perp, \text{"n/v"}\}$ such that $|\{p_k : x_k = x'\}| > t_a$ then p_j outputs x', otherwise p_j outputs $\perp$.

Lemma 2. *If $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, protocol* FixReceive *has the following properties. If p_j is alive at the end of the protocol then he outputs a value x', where $x' \in \{x, \perp\}$ unless $p_i \in \mathcal{A}$, and $x' = x$ when $p_i \in \mathcal{H} \cup \mathcal{R}$. Moreover, p_j might become a zombie only when $p_j \in \mathcal{R} \cup \mathcal{SR}$ and when he becomes a zombie every player notices.*

5 Byzantine Agreement

In this section we build primitives solving the Byzantine Agreement (BA) problem, which we will later use as tools for constructing the main SFE protocol. BA comes in two flavors, namely *consensus* and *broadcast*. Informally, consensus guarantees that n players, each holding an input, can decide on a common output y, where $y = x$ if all non-actively corrupted players had (the same) input x. On the other hand, broadcast allows a dedicated player p_s holding input x_s, to consistently send x_s to every player.

In our BA protocols, the players communicate over the strengthened authenticated network which is constructed using FixReceive. More precisely, whenever $p_i \in \mathcal{P}$ is instructed to bilaterally send a message to $p_j \in \mathcal{P}$, the protocol FixReceive is invoked. Because alive players might become zombies only within FixReceive, all the designed protocols have the following property: *Only receive-omission corrupted players might become zombies.* The proofs of the lemmas can be found in the full version of the paper.

5.1 Consensus

For constructing a consensus protocol, we use the standard approach [BGP89, FM00]: We construct weaker consensus primitives, and then compose them in a clever way to construct the desired consensus primitive. We construct three such weaker primitives called *Weak Consensus, Graded Consensus*, and *King Consensus*.

Weak Consensus. Informally, weak consensus guarantees that there are no inconsistencies among the outputs of the non-actively corrupted players, but some of them (even alive) might have no output (we say that they output $\perp$). However, we get the guarantee that if the players *pre-agreed* on some value x, i.e., all non-actively corrupted players had input (the same) x, then we get *post-agreement* on x, i.e., all non-actively corrupted players output x.[9] In the following we describe protocol WeakConsensus which achieves weak consensus in our model. The input of each $p_i \in \mathcal{P}$ is denoted as x_i

Protocol WeakConsensus $(\mathcal{P}, t_a, t_\rho, t_\sigma, \vec{x} = (x_1, \ldots, x_n))$

1. Each $p_i \in \mathcal{P}$ sends x_i to every $p_j \in \mathcal{P}$, by invoking FixReceive; p_j denotes the received value by $x_j^{(i)}$.
2. Each $p_j \in \mathcal{P}$ sets

$$y_j := \begin{cases} x & \text{, if } (|\{p_i : x_j^{(i)} = x\}| \geq n - t_a - t_\sigma - t_\rho) \ \bigwedge \\ & \qquad (|\{p_i : x_j^{(i)} \notin \{x, \perp\}\}| \leq t_a) \\ \perp & \text{, otherwise} \end{cases}$$

Lemma 3. *If $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, the protocol* WeakConsensus *has the following properties. Weak Consistency: Every (alive) $p_j \in \mathcal{P} \setminus \mathcal{A}$ outputs $y_j \in \{x', \perp\}$ for*

[9] Recall that the zombies send no values in any protocol and receive no output.

some $x' \in \mathbb{F}$. *Correctness: If every* $p_i \in \mathcal{P} \setminus \mathcal{A}$ *who is alive at the beginning of* WeakConsensus *has input* $x_i = x$, *then* $x' = x$.

Graded Consensus. In Graded Consensus each $p_i \in \mathcal{P}$ outputs a pair (y_i, g_i), where y_i is p_i's actual output-value and $g_i \in \{0, 1\}$ is a bit, called p_i's *grade*. The grade g_i has the meaning of the confidence level of p_i on the fact that all non-actively corrupted players also output y_i. In particular, if $g_i = 1$ for some non-actively corrupted p_i then $y_j = y_i$ for every (alive) non-actively corrupted $p_j \in \mathcal{P}$. Moreover, when the non-actively corrupted players pre-agreed on a value x, then they all output x with grade 1.

In the following we describe the protocol GradedConsensus. The idea is to have the players first invoke the protocol WeakConsensus and then exchange their outputs of WeakConsensus to decide on the actual output and the corresponding grade.

Protocol GradedConsensus $\left(\mathcal{P}, t_a, t_\rho, t_\sigma, \vec{x} = (x_1, \ldots, x_n)\right)$

1. Invoke WeakConsensus $(\mathcal{P}, t_a, t_\rho, t_\sigma, \vec{x}\,)$; p_i denotes his output by x'_i.
2. Each $p_i \in \mathcal{P}$ sends x'_i to every $p_j \in \mathcal{P}$ by invocation of FixReceive; p_j denotes the received value by $x_j^{(i)}$.
3. Each $p_j \in \mathcal{P}$ sets $y_j := \begin{cases} x & \text{, if there exists } x \in \mathbb{F} \text{ s.t. } |\{p_i :\ x_j^{(i)} = x\}| > t_a \\ 0 & \text{, otherwise} \end{cases}$

   and sets $g_j := \begin{cases} 1 & \text{, if } (|\{p_i :\ x_j^{(i)} \in \{y_j, \perp\}\}| \geq n - t_a) \ \bigwedge \\ & \qquad (|\{p_i :\ x_j^{(i)} = y_j\}| \geq n - t_a - t_\rho - t_\sigma) \\ 0 & \text{, otherwise} \end{cases}$

Lemma 4. *If* $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, *protocol* GradedConsensus *has the following properties. Graded Consistency: If some* $p_j \in \mathcal{P} \setminus \mathcal{A}$ *outputs* $(y_j, g_j) = (y, 1)$ *for some* $y \in \mathbb{F}$, *then every (alive)* $p_k \in \mathcal{P} \setminus \mathcal{A}$ *outputs* $(y_k, g_k) = (y, g_k)$, *where* $g_k \in \{0, 1\}$. *Graded Correctness: If every* $p_i \in \mathcal{P} \setminus \mathcal{A}$ *who is alive at the beginning of* GradedConsensus *has input* $x_i = x$, *then every (alive)* $p_j \in \mathcal{P} \setminus \mathcal{A}$ *outputs* $(y_j, g_j) = (x, 1)$.

King Consensus. In King Consensus there is a distinguished player $p_k \in \mathcal{P}$, called the *king*. King Consensus guarantees that if the king is uncorrupted, then all non-actively corrupted players output the same value. Additionally, independent of the king's corruption, if the non-actively corrupted players pre-agreed on a value x, then they all output x. The protocol KingConsensus is described in the following.

Protocol KingConsensus $(\mathcal{P}, t_a, t_\rho, t_\sigma, \vec{x} = (x_1, \ldots, x_n), p_k)$

1. Invoke GradedConsensus$(\mathcal{P}, t_a, t_\rho, t_\sigma, \vec{x}\,)$; p_i denotes his output by (x'_i, g_i).
2. The king p_k sends x'_k to every $p_j \in \mathcal{P}$ by invocation of FixReceive.
3. Each $p_j \in \mathcal{P}$ sets

   $y_j; = \begin{cases} x'_j & \text{, if } (g_j = 1) \text{ or } (p_k \text{ sent } x'_k = \perp) \\ x'_k & \text{, otherwise} \end{cases}$

Lemma 5. *If* $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, *the protocol* KingConsensus *has the following properties. King Consistency: If the king* p_k *is uncorrupted, then every* $p_j \in \mathcal{P} \setminus \mathcal{A}$ *outputs* $y_j = y$. *Correctness: If every* $p_i \in \mathcal{P} \setminus \mathcal{A}$ *who is alive at the beginning of* KingConsensus *has input* $x_i = x$ *then they all output* $y = x$.

Consensus. Building a consensus protocol from king consensus is straight-forward: Invoke KingConsensus with $t_a + t_\rho + t_\sigma + 1$ different players as king, where the input of the i-th iteration is the output of the $(i-1)$-th iteration. As there are at most $t_a + t_\rho + t_\sigma$ corrupted players, at least one of the kings will be uncorrupted, hence consistency on the output value will be achieved in the corresponding iteration; the correctness of KingConsensus guarantees that this value will not be changed in any future iteration. Note that when we have pre-agreement on some value then consistency on this value is achieved from the first iteration independent of the king.

Lemma 6. *If $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, the protocol* Consensus *has the following properties. Consistency: All (alive) $p_i \in \mathcal{P} \setminus \mathcal{A}$ output (the same) $y \in \mathbb{F}$. Correctness: If every $p_i \in \mathcal{P} \setminus \mathcal{A}$ who is alive at the beginning of* Consensus *has input $x_i = x$ then $y = x$.*

5.2 Broadcast

The standard approach for achieving broadcast when consensus is given, is to have the sender p_s send his input to every player, and then run consensus on the received values. Unfortunately, this generic approach does not work in our setting, as it provides no guarantees when a send-omission corrupted p_s fails to send his input to some uncorrupted players.

To guarantee that a non actively corrupted p_s never broadcasts a wrong value we extend the above generic protocol by adding the following steps: p_s sends a confirmation bit to every player, i.e., a bit b where $b = 1$ if p_s agrees with the output of the consensus and $b = 0$ otherwise; subsequently, the players invoke consensus on the received bits to establish a consistent view on the confirmation-bit and they accept the output of the generic broadcast protocol only if this bit equals 1, otherwise they output $\perp$. This results in the protocol Broadcast (see below).

Protocol Broadcast $(\mathcal{P}, t_a, t_\rho, t_\sigma, p_s, x_s)$

1. p_s sends x to every $p_j \in \mathcal{P}$ (by FixReceive), who denotes the received value by x_j ($x_j = 0$ if p_j received $\perp$).
2. The players invoke Consensus on the received values. Let y_j denote p_j's output.
3. Each p_j sends y_j to p_s (by FixReceive).
4. p_s sends a confirmation bit b to every $p_i \in \mathcal{P}$ (by FixReceive), where $b = 1$ if p_s received $y_j = x$ from more that t_a players in the previous step and $b = 0$ otherwise; p_i denotes the received bit by b_i ($b_i = 0$ if p_i received $\perp$).
5. Invoke Consensus $(\mathcal{P}, t_a, t_\sigma, t_\rho, (b_1, \ldots, b_n))$. For each $p_i \in \mathcal{P}$, if p_i's output in Consensus is 1 then p_i outputs y_i, otherwise he outputs $\perp$.

Lemma 7. *If $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, protocol* Broadcast *has the following properties. Consistency: All (alive) $p_j \in \mathcal{P} \setminus \mathcal{A}$ output the (same) value $y_j = y$. Correctness: $y \in \{x, \perp\}$ when $p_s \in \mathcal{P} \setminus \mathcal{A}$, where $y = x$ when $p_s \in \mathcal{H} \cup \mathcal{R}$ and he is alive at the end of the protocol, and $y = \perp$ when p_s has been a zombie from the beginning of the protocol.*

6 Tools

In this section we describe sub-protocols that will be used as building-blocks in the construction of the main SFE and MPC protocols. Some of the sub-protocols are non-robust, and might abort with a non-empty set $B \subseteq \mathcal{P}$. When they abort, then all (alive) players in $\mathcal{P}$ notice it and they also learn the set B. As in the case of BA, some alive players might become zombies during the invocation of the sub-protocols, but only when they are receive-omission corrupted.

6.1 Secret Sharing

A secret sharing scheme allows a player, called the *dealer*, to distribute his input among the players in some player set $\mathcal{P}$, so that only qualified sets of players can reconstruct it. As usual in the threshold adversary literature, we use Shamir-sharings for sharing values: With each $p_i \in \mathcal{P}$ a unique publicly known $\alpha_i \in \mathbb{F}$ is associated. A secret s is t-*shared* among the players in $\mathcal{P}$ when there exists a degree-t polynomial $q(\cdot)$ with $q(0) = s$, and every non actively corrupted $p_i \in \mathcal{P}$ holds $s_i \in \{q(\alpha_i), \perp\}$, where $s_i = q(\alpha_i)$ unless p_i is receive-omission corrupted. The value s_i is p_i's *share* of s. We refer to the vector of shares, denoted by $\langle s \rangle = (s_1, \ldots, s_n)$, as a t-*sharing* of s.

We say that $\langle s \rangle$ is a t-*consistent* sharing of s among the players in $\mathcal{P}$ if there exists a degree-t polynomial $q(\cdot)$ such that each non actively corrupted $p_i \in \mathcal{P}$ holds share $s_i \in \{q(\alpha_i), \perp\}$. We say that $\langle s \rangle$ is a t-*valid* sharing of s among the players in $\mathcal{P}$, if $\langle s \rangle$ is t-consistent and for some degree-t polynomial $q(\cdot)$ with $q(0) = s$, each uncorrupted $p_i \in \mathcal{P}$ holds share $s_i = q(\alpha_i)$.

Protocol Share allows a dealer p to t-share his input among the players in any set $\mathcal{P}$. Essentially it is a passive Shamir-sharing protocol: p picks a degree-t uniformly random polynomial $q(\cdot)$ and sends $q(\alpha_i)$ to p_i. The following lemma states the achieved security.

Lemma 8. *Protocol* Share$(\mathcal{P}, t, p, s)$ *has the following properties. Correctness: When* $p \in \mathcal{P} \setminus \mathcal{A}$ *then* Share *outputs a* t*-consistent sharing* $\langle s \rangle$ *of* s *among the players in* $\mathcal{P}$*, where* $\langle s \rangle$ *is even* t*-valid unless* $p \in \mathcal{A} \cup \mathcal{S} \cup \mathcal{SR}$ *or unless* p *is a zombie. Privacy: The players in any set* $\mathcal{P}' \subseteq \mathcal{P}$ *with* $|\mathcal{P}'| \leq t$ *get no (joint) information on* s*.*

In the following we describe the protocols PublicReconstruct and Reconstruct used to reconstruct a shared value publicly and towards some output player p, respectively. The protocols take as input a sharing of a value among the players in some $\mathcal{P}'$ ($\mathcal{P}'$ might be different than $\mathcal{P}$). In protocol Reconstruct (resp. PublicReconstruct) every $p_i \in \mathcal{P}'$ sends his share to p (resp. broadcasts his share to $\mathcal{P}$) and then p (resp. every $p_j \in \mathcal{P}$) reconstructs the shared value using standard error correction. Due to their similarity we only describe protocol Reconstruct and state the security of both protocols in a joint lemma.

Protocol Reconstruct $(\mathcal{P}', t, t', p, \langle s \rangle)$

1. Each $p_i \in \mathcal{P}'$ sends his share s_i to p.
2. p finds, using standard polynomial interpolation techniques, a degree t polynomial $f(\cdot)$ with the property that more than $t + t'$ of the received shares lie on $f(\cdot)$ and outputs $s' = f(0)$. If no such polynomial exists then p_j outputs $\perp$.

Lemma 9. *Assume that there exists t_c such that there are at most t_c corrupted players in $\mathcal{P}'$, of whom at most t' are actively corrupted and the condition $t + t' + t_c < |\mathcal{P}'|$ holds. Then the protocol* Reconstruct *(resp.* PublicReconstruct*)*[10] *reconstructs a value s' towards player p (resp. towards every $p_j \in \mathcal{P}$), where $s' \in \{s, \perp\}$ if $\langle s \rangle$ is a t-consistent sharing of s among the players in $\mathcal{P}'$, and $s' = s$ if $\langle s \rangle$ is t-valid.*

6.2 Engineering the Network - Secure Channels

The trick of engineering the network allowed us to reduce the effect of receive-omission corruption. However, because the channels which we achieve provide no privacy guarantees, we cannot use the resulting network directly to build a perfectly secure SFE protocol. In the following, we show how to engineer the initial network of secure channels to get a new network-layer (also of secure channels) with stronger security guarantees.

The new network layer will allow any $p_j \in \mathcal{P}$ who receives $\perp$ instead of a message x from $p_i \in \mathcal{P}$ to decide whether he (i.e., p_j) is receive-omission corrupted or the sender p_i is corrupted. Additionally, when the reception fails because of p_i, then every (alive) player will recognize that p_i is (actively or send-omission) corrupted. Given Broadcast and a uniformly random key $k_{i,j} \in \mathbb{F}$ known exclusively to p_i and p_j, this can be achieved as follows: For p_i to privately send s to p_j, p_i uses $k_{i,j}$ as a one time pad to perfectly blind s, and broadcasts the blinded value $s + k_{i,j}$. Because only p_i and p_j know $k_{i,j}$, only p_j can unblind the broadcasted message and any other player gets no information about it. As syntactic sugar, we denote this protocol as PrivBroadcast.

In the remaining of this section we concentrate on enabling two players p_i and p_j to establish a secret key $k_{i,j}$ (to use in PrivBroadcast). We design two protocols, called WeakExchangeKey and ExchangeKey, which achieve the following: WeakExchangeKey uses the bilateral secure channels and allows any pair $p_i, p_j \in \mathcal{P}$ to exchange a key as long as *one of them* is at most receive-omission corrupted (i.e., is in $\mathcal{H} \cup \mathcal{R}$) and *the other one* is at most send-omission corrupted (i.e., is in $\mathcal{H} \cup \mathcal{S}$). Protocol ExchangeKey uses protocols WeakExchangeKey and Broadcast and allows p_i and p_j to exchange a key, even when *each of them* is *either* at most receive-omission *or* at most send-omission corrupted. Both protocols work in a publicly detectable way, i.e., all (alive) players notice whether or not the key-exchange worked. In the following we describe the protocols WeakExchangeKey and ExchangeKey in more detail.

Protocol WeakExchangeKey is based on the observation that when p_i is at most send-omission and p_j is at most receive-omission corrupted, then p_j can always securely send messages to p_i through the bilateral secure channel. The protocol works as follows: p_i and p_j choose uniformly random values $k_i \in \mathbb{F}$ and $k_j \in \mathbb{F}$, respectively, and exchange them over their bilateral channel. Subsequently, each of them publicly announces, by Broadcast, whether or not he received a value from the other. If any of them confirms reception of a value then this value is used as the secret key and the protocol succeeds; otherwise the protocol fails. WeakExchangeKey is non-robust and might abort with a set $B \in \{\{p_i\}, \{p_j\}\}$, but only when p_i and/or p_j broadcast $\perp$ (if they both broadcast $\perp$ take the one with the smallest index). The detailed description of WeakExchangeKey and the proof of the following lemma can be found in the full version.

[10] For PublicReconstruct we need to assume a broadcast primitive, which when $3t_a + t_\sigma + t_\rho < |\mathcal{P}|$ we can instantiate by Broadcast.

Lemma 10. *If $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$, protocol* WeakExchangeKey *has the following properties. Correctness: Either it succeeds in p_i and p_j exchanging a uniformly random key k, or it fails, or it aborts with set $B \in \{\{p_i\}, \{p_j\}\}$. It might abort with B only when $B \subseteq \mathcal{R} \cup \mathcal{S} \cup \mathcal{SR} \cup \mathcal{A}$. When it does not abort then the following hold: (1) Every alive $p_k \in \mathcal{P}$ sees whether the protocol succeded or failed, and (2) it always succeeds when $p_i \in \mathcal{H} \cup \mathcal{R}$ and $p_j \in \mathcal{H} \cup \mathcal{S}$ or vice versa (i.e., when $p_i \in \mathcal{H} \cup \mathcal{S}$ and $p_j \in \mathcal{H} \cup \mathcal{R}$). Privacy: The adversary gets no information on k (unless p_i or p_j is actively corrupted).*

We describe the protocol ExchangeKey (see below) and state its achieved security in a lemma. The protocol is non-robust and might abort with set $B \in \{\{p_i\}, \{p_j\}, \{p_i, p_j\}\}$. However, from the fact that it aborted the players can deduce useful information on the corruption of the players in B.

Protocol ExchangeKey $(\mathcal{P}, t_a, p_i, p_j)$

1. For $\ell \in \{i, j\}$: p_ℓ invokes WeakExchangeKey with every $p_r \in \mathcal{P}$. If WeakExchangeKey aborts with B, then ExchangeKey also aborts with B. Denote by $P^\ell_{\text{"ok"}} \subseteq \mathcal{P}$ the set of players who successfully exchanged keys with p_ℓ, and by $P_{\text{"ok"}} := (P^i_{\text{"ok"}} \cap P^j_{\text{"ok"}})$. If $|P_{\text{"ok"}}| \leq 2t_a$ then ExchangeKey aborts with $B = \{p_i, p_j\}$.
2. For $\ell \in \{i, j\}$: p_ℓ picks a value $k_\ell \in_R \mathbb{F}$ uniformly at random and a degree t_a random polynomial $f_\ell(\cdot)$ with $f_\ell(0) = k_\ell$. For each $p_r \in P_{\text{"ok"}}$, p_ℓ sends, by invoking PrivBroadcast with the exchanged keys, the share $f_\ell(\alpha_r)$ to p_r, who denotes the received value as $s^{(\ell)}_r$. If p_ℓ broadcast $\perp$ then ExchangeKey aborts with $B = \{p_\ell\}$ (if both p_i and p_j broadcast $\perp$ take the one with the smallest index).
3. The players in $P_{\text{"ok"}}$ compute a sharing of the sum $k_i + k_j$, by each player (locally) adding his shares, and then publicly reconstruct it by PublicReconstruct. If PublicReconstruct outputs $\perp$ then ExchangeKey aborts with $B = \{p_i, p_j\}$.

Lemma 11. *If $3t_a + t_\sigma + t_\rho < |\mathcal{P}|$, the protocol* ExchangeKey *has the following properties. Correctness: Either p_i and p_j succeed in exchanging a uniformly random key k (and all players notice) or the protocol aborts with a set $B \in \{\{p_i\}, \{p_j\}, \{p_i, p_j\}\}$. It might abort with set B only if one of the following two cases holds: (1) $|B| = 1$ and $B \subseteq \mathcal{R} \cup \mathcal{S} \cup \mathcal{SR} \cup \mathcal{A}$ and (2) $|B| = 2$ and $B \cap (\mathcal{SR} \cup \mathcal{A}) \neq \emptyset$. Privacy: The adversary gets no information on k (unless p_i or p_j is actively corrupted).*

6.3 Protocol $\mathsf{SFE}^{(\mathsf{BC})}$

The last tool is a protocol, called $\mathsf{SFE}^{(\mathsf{BC})}$, which perfectly securely evaluates any given function f without fairness but with unanimous abort [GL02]. In particular, protocol $\mathsf{SFE}^{(\mathsf{BC})}$ either perfectly (t_a, t_ρ, t_σ)-securely evaluates the function f, or it aborts with set $B \in \{\{p_i\}, \{p_j\}, \{p_i, p_j\}\}$ for some $p_i, p_j \in \mathcal{P}$. The adversary might force the protocol to abort even after receiving the outputs of actively corrupted players. However, when it aborts every player learns useful information about the corruption of the players in B.

The idea is the following: Let $\Pi_{\mathcal{P},t}(\cdot)$ denote a protocol which perfectly t-securely evaluates any given function, in the presence of an adversary who can (only) actively corrupt up to t players.[11] Such a protocol is known to exist if $3t < n$ [BGW88]. Also, let C_f denote the arithmetic circuit which computes a given function f. To securely evaluate C_f, protocol $\mathsf{SFE}^{(\mathsf{BC})}$ invokes protocol $\Pi_{\mathcal{P},t}(C_f)$ over the engineered network of secure channels. More precisely, each $p_i \in \mathcal{P}$ executes the instructions of $\Pi_{\mathcal{P},t}(C_f)$ with the following modification: whenever p_i is instructed to bilaterally send a message x to some $p_j \in \mathcal{P}$, protocol $\mathsf{ExchangeKey}(\mathcal{P}, p_j, p_j)$ is invoked to have p_i and p_j exchange a uniformly random key, and then the message x is sent using $\mathsf{PrivBroadcast}$ with the established key; whenever p_i is instructed to broadcast a message, he invokes $\mathsf{Broadcast}$. If some invocation of $\mathsf{ExchangeKey}$ aborts with B or some $p_i \in \mathcal{P}$ broadcasts $\perp$ (in this case we set $B = \{p_i\}$) then $\mathsf{SFE}^{(\mathsf{BC})}$ aborts with B.

In the following lemma we state the security of $\mathsf{SFE}^{(\mathsf{BC})}$. The proof follows directly from the perfect t-security of protocol $\Pi_{\mathcal{P},t}(\cdot)$ and the perfect security of protocols $\mathsf{ExchangeKey}$ and $\mathsf{Broadcast}$. $\mathsf{SFE}^{(\mathsf{BC})}$ is parametrized by a single threshold, namely t, but it assumes as given the primitives $\mathsf{Broadcast}$ and $\mathsf{ExchangeKey}$ as specified in Lemmas 7 and 11, respectively.[12]

Lemma 12. *Given* $\mathsf{Broadcast}$ *and* $\mathsf{ExchangeKey}$*, assuming that the condition* $3t < |\mathcal{P}|$ *holds protocol* $\mathsf{SFE}^{(\mathsf{BC})}(\mathcal{P}, t, C_f)$ *has the following properties. Correctness: Either it perfectly* (t, t_σ, t_ρ)*-securely evaluates the circuit* C_f *among the players in* $\mathcal{P}$ *for any* $t_\sigma, t_\rho < n$*, or it aborts with a set* $B \subseteq \mathcal{P}$*. It might abort with set* B *only when one of the following two cases holds: (1)* $|B| = 1$ *and* $B \subseteq \mathcal{R} \cup \mathcal{S} \cup \mathcal{SR} \cup \mathcal{A}$ *and (2)* $|B| = 2$ *and* $B \cap (\mathcal{SR} \cup \mathcal{A}) \neq \emptyset$*. Privacy: The adversary gets no more information than what he can compute from the specified inputs and outputs of actively corrupted players (i.e., from the inputs and outputs she should get when the protocol does not abort).*

7 SFE

In this section we prove the necessary and sufficient condition for perfectly (t_a, t_ρ, t_σ)-securely evaluating any given function $f(\cdot)$, namely we prove the following theorem:

Theorem 1. *Perfectly* (t_a, t_ρ, t_σ)*-secure SFE is possible if and only if* $3t_a + t_\rho + t_\sigma < n$*.*

The necessity of the condition follows, with some additional work, from the necessity of the conditions $3t_a < n$ for SFE [BGW88]; we state the necessity in the following lemma which is proved in the full version of this paper.

Lemma 13. *If* $3t_a + t_\rho + t_\sigma \geq n$ *then there are functions which cannot be perfectly* (t_a, t_ρ, t_σ)*-securely evaluated.*

[11] Here, t-secure evaluation is according to any of the standard security definition (with fairness and guaranteed output delivery) of protocols tolerating an active-only adversary [MR91, Can00, DM00, BPW03].

[12] In slight abuse of notation here, we write $\mathsf{Broadcast}$ and $\mathsf{ExchangeKey}$ to refer *not* to the protocols but to primitives achieving the security specified in Lemmas 7 and 11 (independent of pre-conditions). To be able to instantiate them with our protocols we will have to guarantee that the pre-conditions of the lemmas are satisfied.

The sufficiency is proved by constructing an SFE protocol for computing any given function f. For simplicity, we assume that f takes one input per player and has one global output. Using standard techniques, we can obtain a protocol for computing functions with multiple inputs and/or multiple or even private outputs.

On a high level, the evaluation of the function f proceeds in three stages: In the first stage, called the *input stage*, every $p_i \in \mathcal{P}$ t_a-shares his input to the players in $\mathcal{P}$. Next, in the *computation stage*, the players use $\mathsf{SFE}^{(\mathsf{BC})}$ to compute a random t_a-sharing of the output of the function f. Finally, in the *output stage*, this sharing is reconstructed towards every player using Reconstruct. In the remaining of this section we describe in detail the three stages, and give a detailed description of protocol SFE.

The input stage. In this stage protocol Share is invoked to have each $p_i \in \mathcal{P}$ t_a-share his input $s^{(i)}$ to the players in $\mathcal{P}$. Denote the resulting sharing by $\langle s^{(i)} \rangle$. The security of Share guarantees that for any non actively corrupted p_i $\langle s^{(i)} \rangle$ is a t_a-consistent sharing of $s^{(i)}$, where $\langle s^{(i)} \rangle$ is even t-valid when $p_i \in \mathcal{H} \cup \mathcal{R}$.

The computation stage. The goal is to securely compute, using $\mathsf{SFE}^{(\mathsf{BC})}$, *a uniformly random t_a-valid sharing* of the output of f on input the values that where shared in the input stage. This stage is non-robust and might abort with a player set $B \subseteq \mathcal{P}$, when $\mathsf{SFE}^{(\mathsf{BC})}$ aborts with B. When it aborts, the players use the information about the set B, which is provided by Lemma 12, to repeat this stage in a smaller setting, i.e., among the players in $\mathcal{P}' := \mathcal{P} \setminus B$. The security of $\mathsf{SFE}^{(\mathsf{BC})}$ guarantees that, even when it aborts, the adversary learns at most the outputs of actively corrupted players, which, as they are shares of a (uniformly random) t_a-sharing, give her no information on the input-sharings. Hence, in the successful iteration of $\mathsf{SFE}^{(\mathsf{BC})}$, both the inputs of actively corrupted players and the decision of which send-omission corrupted players give their inputs are independent of the inputs of non actively corrupted players.

Initially $\mathcal{P}' := \mathcal{P}$ and $t'_a := t_a$. Protocol $\mathsf{SFE}^{(\mathsf{BC})}$ is invoked with player set $\mathcal{P}'$ and threshold t'_a, to compute the circuit $C^{t_a}_{\langle f \rangle}$ which does the following: $C^{t_a}_{\langle f \rangle}$ takes as input from each $p_j \in \mathcal{P}'$ his share of each of the input-sharings $\langle s^{(1)} \rangle, \ldots, \langle s^{(n)} \rangle$. For each such sharing $\langle s^{(i)} \rangle$: $C^{t_a}_{\langle f \rangle}$ attempts, exactly as in protocol Reconstruct, to reconstruct the shared value; if the reconstruction succeeds it sets $\hat{s}_i$ to the reconstructed value, otherwise it sets $\hat{s}_i$ to a default value (e.g., $\hat{s}_i := 0$). Note that for $t = t'_a, t' = t'_a$, and $t_c = t'_a + t_\sigma + t_\rho$ all the sufficient conditions for Reconstruct are satisfied; therefore, $C^{t_a}_{\langle f \rangle}$ correctly reconstructs the input of every $p_i \in \mathcal{H} \cup \mathcal{R}$ (which is t-valid), and for every $p_i \in \mathcal{S} \cup \mathcal{SR}$ it either reconstructs p_i's input or it takes a default value (since the sharing of p_i is a t-consistent sharing of his input). Having computed the values $\hat{s}_1, \ldots, \hat{s}_n$, $C^{t_a}_{\langle f \rangle}$ inputs them to the circuit computing f; denote the output by y. Finally, $C^{t_a}_{\langle f \rangle}$ computes and outputs a uniformly random t_a-valid sharing of y among the players in $\mathcal{P}'$. We point out that the circuit $C^{t_a}_{\langle f \rangle}$ can be efficiently computed from the circuit which computes the function f [IKLP06].

To be able to re-invoke $\mathsf{SFE}^{(\mathsf{BC})}$ in $\mathcal{P}' = \mathcal{P}' \setminus B$ when it aborts with B, we need to guarantee that in the updated $\mathcal{P}'$: (1) the condition $3t'_a < |\mathcal{P}'|$, which is sufficient for $\mathsf{SFE}^{(\mathsf{BC})}$, holds and (2) no inputs of non actively corrupted players are lost. To ensure Property (1), we use the idea of *player elimination* [HMP00].[13] The security of $\mathsf{SFE}^{(\mathsf{BC})}$

[13] To our knowledge, this is the first work which uses the idea of player elimination not for improving efficiency but rather for arguing about feasibility of protocols.

guarantees that when it aborts with set B, then either $|B| = 1$ and $B \subseteq \mathcal{R} \cup \mathcal{S} \cup \mathcal{SR} \cup \mathcal{A}$ or $|B| = 2$ and $B \cap (\mathcal{SR} \cup \mathcal{A}) \neq \emptyset$. Therefore, by eliminating the players in B we might only change the ratio of uncorrupted vs. actively corrupted players in $\mathcal{P}'$ in favor of the uncorrupted players. However, as the set $\mathcal{P}'$ becomes smaller, the players might have to reduce the actual threshold t'_a. To be on the safe side, t'_a is reduced only when at least as many players as there can be send-/receive-omission corrupted have been eliminated. Property (2) is guaranteed because, first, the t_a-consistency and t_a-validity of input sharings cannot be destroyed by deleting players and, second, the newly computed t'_a satisfies, as we show, the sufficient condition for Reconstruct.

The output stage. The players invoke Reconstruct with the (latest) t'_a to reconstruct the sharing created in the successful iteration of $\mathsf{SFE}^{(\mathsf{BC})}$. Because the protocol $\mathsf{SFE}^{(\mathsf{BC})}$ outputs a t_a-valid sharing of the output, and, as we will show, t'_a satisfies the sufficient condition for protocol Reconstruct, the reconstruction is robust. For completeness we describe the protocol SFE (see below) and state the achieved security in the following lemma.

Protocol SFE $(\mathcal{P}, t_a, t_\rho, t_\sigma, f)$

0. Set $\mathcal{P}' := \mathcal{P}$, and $t'_a := t_a$.
1. For each $p_i \in \mathcal{P}$ invoke Share$(\mathcal{P}, t_a, p_i, x_i)$. Each $p_j \in \mathcal{P}$ denotes the vector of all shares he received by $\vec{x}^{\,(j)}$.
2. The players in $\mathcal{P}'$ invoke $\mathsf{SFE}^{(\mathsf{BC})}(\mathcal{P}', t'_a, C^{t_a}_{\langle f \rangle})$,where each $p_i \in \mathcal{P}'$ has input $\vec{x}^{\,(j)}$.[a] If $\mathsf{SFE}^{(\mathsf{BC})}$ aborts with B, then set $\mathcal{P}' = \mathcal{P}' \setminus B$, set $t'_a := t_a - \max\{0, |\mathcal{P} \setminus \mathcal{P}'| - (t_\sigma + t_\rho)\}$ and repeat this step; otherwise denote by $\langle f \rangle$ the output sharing.
3. For each $p_j \in \mathcal{P}$ invoke Reconstruct$(\mathcal{P}', t_a, t'_a, p_j, \langle f \rangle)$.

[a] The required invocations of Broadcast and ExchangeKey are done in the player set $\mathcal{P}$.

Lemma 14. *Protocol* SFE *is perfectly* (t_a, t_ρ, t_σ)*-secure if* $3t_a + t_\rho + t_\sigma < |\mathcal{P}|$.

Proof (sketch). Termination is guaranteed because Step 2 is repeated at most $t_a + t_\sigma + t_\rho$ times (in each repetition at least one corrupted player is removed from $\mathcal{P}'$). Correctness follows from the security of the invoked sub-protocols; however one needs to verify that the corresponding sufficient conditions hold whenever they are invoked. This follows from a player-elimination argument, which, due to space restrictions, is deleted from this proceedings version. Privacy follows also from the security of the invoked subprotocols and from the fact that all the sharings that we do are of degree t_a (except of those done internally in $\mathsf{SFE}^{(\mathsf{BC})}$ whose privacy is guaranteed by the security of $\mathsf{SFE}^{(\mathsf{BC})}$), therefore they leak no information to the adversary about the inputs. □

As already mentioned, when the adversary is rushing there are functions that cannot be strongly (t_a, t_ρ, t_σ)-securely evaluated, except in trivial corruption scenarios (i.e., if $t_a = 0$ or $t_\sigma = 0$). However, when the adversary is non-rushing the above protocol can be used to achieve strong security. Indeed, before the output stage, the adversary gains no useful information. As protocol Reconstruct is single round, if, within the output stage, we run it in parallel for every $p_i \in \mathcal{P}$, then a non-rushing adversary has to choose

which receive-omission corrupted players do not get enough messages to reconstruct the output before getting any information about the output. This implies strong security. We point out that the necessity of condition $3t_a + t_\rho + t_\sigma < n$ for SFE is independent of whether or not the adversary is rushing.

Corollary 1. *Assuming that the adversary in non-rushing, perfectly* strongly (t_a, t_ρ, t_σ)*-secure SFE is possible if and only* $3t_a + t_\rho + t_\sigma < n$.

8 Computing Reactive Circuits (MPC)

In this section we show how to compute reactive functionalities, i.e., functionalities that receive inputs from and give outputs to the players several times during the computation (an output can depend on all previous inputs). An important consideration when computing a reactive functionality, is to make sure that the players can keep a secret joint state.

The circuit to be computed consists of input, output, addition, and multiplication gates.[14] We model the reactiveness of the computation by assigning to each gate a point in time in which the gate should be evaluated. The circuit is evaluated in a gate-by-gate fashion, using protocol SFE, where the evaluation of each gate (except for the output gates) yields a uniformly random t_a-valid sharing of the output of the gate among the players in $\mathcal{P}$. Keeping state is guaranteed by the fact that such a sharing is robustly reconstructible, e.g., by using protocol Reconstruct, given that the condition $3t_a + t_\sigma + t_\rho < n$ holds (Lemma 9). The privacy of the state is guaranteed, as there are at most t_a actively corrupted players.

To evaluate addition and multiplication gates, protocol $\mathsf{SFE}^{(\mathsf{BC})}$ is invoked to compute the circuits $C_{\langle Mult \rangle}$ and $C_{\langle Add \rangle}$, respectively, which on input t_a-valid sharings of the inputs x_1 and x_2 of the gate output a uniformly random t_a-valid sharing of the sum $x_1 + x_2$ and of the product $x_1 \cdot x_2$, respectively. For an output gate, protocol Reconstruct is invoked (with $\mathcal{P}' = \mathcal{P}$, and $t = t' = t_a$) to reconstruct the shared output towards the output player.

To evaluate an input gate, protocol SFE is invoked to evaluate the circuit $C_{\langle I \rangle}$ which takes as input the input of the corresponding player (and no value from other players) and computes a uniformly random t_a-valid sharing of it among the players in $\mathcal{P}$. *Exceptionally* in the evaluation of input gates, *even* the zombies are required to take part as if they were alive. This is possible as all players (including zombies) hold synchronized clocks, and are aware of when it is time to evaluate an input gate.[15] Instructing the zombies to "wake up" during the evaluation of input gates ensures that every $p_i \in \mathcal{H} \cup \mathcal{R}$, even if he is a zombie, is able to give input to the computation. When the evaluation of the gate finishes, all zombies "sleep" again, i.e., they stop playing (until the next input gate). The security of the MPC protocol follows from the security of protocols SFE and Reconstruct.

[14] This does not exclude probabilistic circuits, as a random gate can be simulated by having each player input a random value and taking the sum of the inputs as the output of the gate.

[15] A zombie might re-become zombie during the evaluation of the input gate, in which case he gives up the evaluation of the gate.

Theorem 2. *Perfectly* (t_a, t_ρ, t_σ)*-secure (reactive) MPC is possible if and only if* $3t_a + t_\sigma + t_\rho < n$.

As in the case of SFE, when the adversary is non-rushing, then by evaluating in parallel each tuple of output gates that are due to be evaluated at the same time, we get a strongly perfectly secure MPC protocol.

Corollary 2. *Assuming that the adversary in non-rushing, perfectly strongly* (t_a, t_ρ, t_σ)*-secure (reactive) MPC is possible if an only if* $3t_a + t_\rho + t_\sigma < n$.

9 (Full) Omission Corruption

Our results can be trivially used to obtain sufficient bounds for MPC and SFE in the presence of an adversary who can full-omission corrupt up to t_ω players and, simultaneously, actively corrupted t_a players (as in [Koo06]). Indeed, by setting $t_\sigma = t_\rho = t_\omega$ in our MPC protocols, we get a protocol which perfectly (t_a, t_ω)-securely realized any function when $3t_a + 2t_\omega < n$. Note that this bound is strictly better than the bound $3t_a + 4t_\omega < n$ which was proved sufficient in [Koo06].

Lemma 15. *Perfectly* (t_a, t_ω)*-secure (even reactive) MPC is possible if* $3t_a + 2t_\omega < n$.

10 Extensions

Our results can be extended to deal with adversaries who can, additionally, passively and fail-corrupt players; denote by t_p and t_f the corresponding thresholds. The proof of the following lemma is omitted, but we give some evidence of its validity: Fail-corruption comes almost "for free" as in our protocol a fail-corrupted players behaves exactly as a receive-omission corrupted player with the only difference that, instead of turning him into a zombie the adversary can make him crash. To incorporate passive corruption we need to do the following modifications: (1) the degree of the shares that are computed in SFE is increased by t_p; (2) for $\mathsf{SFE}^{(\mathsf{BC})}$, instead of invoking, over the engineered network, the protocol $\Pi_{\mathcal{P},t}(\cdot)$ [BGW88] which tolerates only actively-corruption, we use a protocol which tolerates both active and passive corruption, simultaneously. Such a protocol is known to exist if $3t_a + 2t_p < n$ [FHM98]. These modifications will guarantee privacy of our computation.

Lemma 16. *Perfectly* $(t_a, t_p, t_f, t_\rho, t_\sigma)$*-secure* MPC *is possible if and only if* $3t_a + 2t_p + t_\sigma + t_\rho + t_f < n$.

Using techniques from Secure Message Transmission [DDWY93], we can extend our results to allow every (even uncorrupted) $p_i \in \mathcal{P}$ to suffer from some message loss, as long as we have the following guarantee: in every round every $p_i \in \mathcal{H} \cup \mathcal{S}$ might lose at most t_a of the messages sent to him by players $p_j \in \mathcal{H} \cup \mathcal{R}$.

Acknowledgements. We would like to thank Martin Hirt for many useful discussions and comments.

References

[Bea91a] Beaver, D.: Foundations of secure interactive computing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 377–391. Springer, Heidelberg (1992)

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

[BGP89] Berman, P.J., Garray, J., Perry, J.: Towards optimal distributed consensus. In: FOCS 1989, pp. 410–415 (1989)

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

[BPW03] Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library (2003)

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

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

[DDWY93] Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. Journal of the ACM 40(1), 17–47 (1993)

[DM00] Dodis, Y., Micali, S.: Parallel reducibility for information-theoretically secure computation. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 74–92. Springer, Heidelberg (2000)

[DS82] Dolev, D., Strong, H.R.: Polynomial algorithms for multiple processor agreement. In: STOC 1982, pp. 401–407 (1982)

[FHM98] Fitzi, M., Hirt, M., Maurer, U.: Trading correctness for privacy in unconditional multi-party computation. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 121–136. Springer, Heidelberg (1998)

[FM98] Fitzi, M., Maurer, U.: Efficient byzantine agreement secure against general adversaries. In: Kutten, S. (ed.) DISC 1998. LNCS, vol. 1499, pp. 134–148. Springer, Heidelberg (1998)

[FM00] Fitzi, M., Maurer, U.: From partial consistency to global broadcast. In: STOC 2000, pp. 494–503 (2000)

[GL02] Goldwasser, S., Lindell, Y.: Secure computation without agreement. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 17–32. Springer, Heidelberg (2002)

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

[GP92] Garay, J.A., Perry, K.J.: A continuum of failure models for distributed computing. In: Segall, A., Zaks, S. (eds.) WDAG 1992. LNCS, vol. 647, pp. 153–165. Springer, Heidelberg (1992)

[Had85] Hadzilacos, V.: Issues of fault tolerance in concurrent computations (databases, reliability, transactions, agreement protocols, distributed computing). PhD thesis, Cambridge, MA, USA (1985)

[HMP00] Hirt, M., Maurer, U., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)

[IKLP06] Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: On combining privacy with guaranteed output delivery in secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 483–500. Springer, Heidelberg (2006)

[Koo06] Koo, C.-Y.: Secure computation with partial message loss. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 502–521. Springer, Heidelberg (2006)

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

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

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

[MR91] Micali, S., Rogaway, P.: Secure computation. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 392–404. Springer, Heidelberg (1992)

[PR03] Parvedy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: International Symposium on Parallel and Distributed Processing — IPDPS 2003, p. 212.2 (2003)

[PT86] Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Softw. Eng. 12(3), 477–482 (1986)

[PW01] Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: IEEE Symposium on Security and Privacy, pp. 184–200 (2001)

[Ray02] Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: Pacific Rim International Symposium on Dependable Computing — PRDC 2002, p. 221 (2002)

[RB89] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: STOC 1989, pp. 73–85 (1989)

[Yao82] Yao, A.C.: Protocols for secure computations. In: FOCS 1982, pp. 160–164 (1982)