



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master Thesis

Efficient Multi-Party Computation Protocols

Micha Riser
riserm@student.ethz.ch

Department of Computer Science
Swiss Federal Institute of Technology Zurich

14th March 2005

Supervisors:
Zuzana Beerliova
Martin Hirt
Prof. Ueli Maurer

Abstract

We present a way to modularize multi-party computation protocols that use the player-elimination technique ([HMP00]) by introducing player-elimination modules. Such modules can be composed sequentially in general and often an efficient parallel self-composition is possible. Player-elimination modules can be used to compose protocols for the information-theoretical model that provide maximal security, i.e., active corruption of up to $t < n/3$ of the n players, or, assuming broadcast channels are available, up to $t < n/2$.

In the second part, we present an efficient, secure multi-party computation protocol for the secure-channels model with broadcast for $t < n/2$. The communication complexity for evaluating a function is $\mathcal{O}(n^3)$ field elements per multiplication and a number of invocations of the broadcast primitive that depends only on the number of inputs and number of players. The presented protocol combines techniques from [CDD⁺99] and [HM01]. For the specification of the protocol, we make extensive use of the previously introduced player-elimination modules.

Contents

1	Introduction	3
1.1	Models and Definitions	3
1.2	Previous Work	4
1.3	Contribution	5
1.4	Notation	5
1.5	Overview	6
2	Player-Elimination	7
2.1	PE-Modules	8
2.2	Segment as Module	10
2.3	Sequential Composition	10
2.4	Parallel Composition	12
3	IC-Signatures	15
3.1	Information Checking (IC)	15
3.2	IC in the PE-Framework	15
3.3	IC-Signatures	18
3.4	Complexity Analysis	20
4	Player-Elimination Secret Sharing	21
4.1	(Verifiable) Secret Sharing	21
4.2	Pre-Sharing	22
4.3	Player-Elimination Secret Sharing	27
4.4	Making PE-Sharings Robust	33
4.5	Complexity Analysis	35
5	Efficient MPC-Protocol for $t < n/2$	37
5.1	Protocol Overview	37
5.2	Generating Random Triples	38
5.3	Giving Input	45
5.4	Circuit Evaluation	45
5.5	Main Segment	46
5.6	Reconstructing Output	49
5.7	Complexity Analysis	49
6	Conclusions	53
6.1	Summary	53
6.2	Future Work	53

A Module Index	55
B Task Description	57
Bibliography	59

Chapter 1

Introduction

Secure *multi-party computation (MPC)* allows a set of players to perform some computation even if they do not trust each other. MPC can be seen as generalization of multi-party problems like joint coin flipping or anonymous secure electronic voting. In its general version, it allows the players to compute an arbitrary agreed function. Thereby, each player gives an input to the function and wants to keep his input secret. This means that, during the execution of the MPC, the other players should gain not more information about the input than what the function output reveals to them. All players must learn the output of the function and the computation has to be correct even if some players deviate from the protocol.

The concept of general secure MPC was introduced by Yao in the year 1982 [Yao82]. Since then, it has attracted many researchers and has generated a large body of literature. Solutions to this problem have been given in various models. Major characteristics of the model are the type of the adversary, which tries to interfere the computation or to gain information about the players' secret inputs, and the type of channels that are available to the players to communicate with each other.

1.1 Models and Definitions

1.1.1 Adversary Models

We assume the presence of a central adversary which can corrupt some of the players involved in the MPC. Such a corruption can be either *passive* or *active*. A passively corrupted player still sticks to the protocol which specifies the communication between the players but reveals all his available information to the adversary. Hence, the adversary can pool the information of all corrupted players to try to gain information about the secrets of the non-corrupted (we call them *honest*) players. If the adversary corrupts a player actively, it learns not only all his values but can also control which values the player sends in the on-going computation.

The possible sets of players which the adversary can corrupt is described by the so-called *adversary structure*. The adversary structure is a set of subsets of the player set out of which the adversary can freely choose one set and corrupt all the players in this set. The players do not get informed about the adversary's

choice. See [HM97] for a formal definition of the adversary structure. The most common adversary structure is the threshold structure, which means that there is some threshold parameter t and the adversary can corrupt any set of players as long as the size of the set does not exceed t . While a *static* adversary has to choose the set of corrupted players at the beginning of the computation, an *adaptive* adversary can corrupt further players during the protocol execution, as long as the total set of corrupted players stays admissible.

Furthermore, the adversary can either have polynomially bounded (cryptographic model) or unbounded (information theoretical model) computational resources.

1.1.2 Communication Models

The players are pairwise connected by synchronous, reliable communication channels (see [Can01] for a definition of synchronous channels). In the context of a computationally unbounded adversary, we consider the secure-channels model, i.e., all communication channels are perfectly secure. Additionally, broadcast channels can be available, which ensure that when a player sends a value to a broadcast channel, all players receive the same value.

1.2 Previous Work

In the year 1987, Goldreich, Micali and Wigderson presented the first general solution to the MPC-problem for the cryptographic model and active corruption of up to $t < n/2$ out of n players [GMW87]. Later, information theoretically secure MPC protocols were presented for $t < n/3$ by Ben-Or, Goldwasser, Wigderson and by Chaum, Crépeau, Damgård [BGW88, CCD88]. Presuming the availability of broadcast channels, information theoretically secure protocols that allow the active corruption of $t < n/2$ players were given by Rabin and Ben-Or and by Beaver [RB89, Bea91].

In the last years, the efficiency of MPC-protocols has successively been improved. For the information theoretically secure model, Hirt and Maurer showed that for active corruption of $t < n/3$ the same communication complexity, $\mathcal{O}(n^2\kappa)$ bits per multiplication, as in the best known passive protocol for $t < n/2$ is possible, where κ denotes the security parameter [HM01]. For the cryptographic model, the most efficient protocols known today communicate $\mathcal{O}(n\kappa)$ and $\mathcal{O}(n^2\kappa)$ bits per multiplication for a passive and active adversary, respectively [HN04]. The core technique which lead in these models to efficient MPC protocols was the so-called *player-elimination (PE)*, introduced in [HMP00].

In the secure-channels model with broadcast and active corruption of up to $t < n/2$ players, however, the most efficient protocol so far was the one presented in [CDD⁺99] which requires to broadcast $\Omega(n^5\kappa)$ bits per multiplication (and additionally a number of bits in the same order is sent over the pairwise channels).

1.3 Contribution

A thorough study of the player-elimination technique has been performed to find a way to apply this technique to MPC in the secure-channels model with broadcast. We introduce the definition of PE-modules, which are building blocks that can be composed to build complex MPC-protocols. Using this concept, we present a protocol that consequently uses player-elimination and gains efficiency by reducing the number of broadcasted bits dramatically. We prove the upper bound of $\mathcal{O}(n^3\kappa)$ bits per multiplication sent over the private channels and $\mathcal{O}(n^3n_I\kappa + n^2\kappa)$ bits broadcast (where n_I is the number of function inputs) to perform secure MPC in the secure-channels model with broadcast and active corruption of $t < n/2$ players.

1.4 Notation

In the subsequent Chapters, we will present numerous n -player protocols. To make the notation of the protocols easier, we use the following conventions:

- Whenever a player receives no or an invalid value during the execution of a protocol, he uses a default value instead. E.g., when he receives a polynomial of too high degree he chooses the constant 0-polynomial instead.
- We will assign the indices $1, \dots, n$ to the n players, i.e., the player set \mathcal{P} is $\{P_1, \dots, P_n\}$. When we consider subsets $\mathcal{P}' \in \mathcal{P}$, we assume without loss of generality that the players in \mathcal{P}' are always indexed from 1 to $n' = |\mathcal{P}'|$, i.e., $\mathcal{P}' = \{P_1, \dots, P_{n'}\}$. This can always be achieved by renumbering the player indices.
- Our computations are performed in some Galois field K , where we require that the size of the field is at least by one larger than the number of players, i.e., $|K| \geq n + 1$. We assume that the players can efficiently compute multiplicative inverses of field elements and that there exists an efficient computable injective mapping of the numbers $\{0, \dots, n\}$ to field elements in K where 0 maps to the neutral element of the field's addition.
- When we use a binary variable b , we consider $b = 1$ as Boolean state *true* and $b = 0$ as *false*. Consequently, we perform the Boolean operations AND (“ \wedge ”) and OR (“ \vee ”) on binary variables.
- We will use Lagrange interpolation to reconstruct polynomials from data points. When we say that we reconstruct a polynomial from the values y_1, \dots, y_k , this means to interpret each $y_i \in \{y_1, \dots, y_k\}$ as the point (i, y_i) . So, we look for the polynomial $p(x)$ of smallest degree that satisfies the condition $p(i) = y_i \forall i \in \{1, \dots, k\}$. This polynomial is uniquely defined and can be found using Lagrange interpolation [Sha79].

To describe communication complexities, we use the following notation: We say that $f(x) = \mathcal{O}(g(x))$ if

$$\exists c_1, c_2 > 0 : \forall n > 0 : f(n) \leq c_1 g(n) + c_2$$

and that $f(x) = \Omega(g(x))$ if

$$\exists c, n_0 > 0 : \forall n > n_0 : f(n) \geq c g(n)$$

To distinguish between the number of bits that are sent over the pairwise channels and the number of bits that the players have to broadcast, we denote the costs for broadcasting a k -bit message with $\text{BC}(k)$.

1.5 Overview

In the next Chapter, we will recapitulate the technique of player-elimination and define what a PE-module is. Then, we will use this definition to present solutions to the MPC problems *signing values* (Chapter 3) and *secret sharing* (Chapter 4). We will use the resulting protocols in Chapter 5 to present an efficient secure general MPC-protocol for the secure-channels model with broadcast. In the last Chapter, a summary of the thesis' results is given and we discuss possible feature work.

Chapter 2

Player-Elimination

In this Chapter, we present the technique of *player-elimination* in *multi-party computation* protocols. Throughout the Chapter, we will consider an active, adaptive threshold adversary which can corrupt at most t out of n players.

Player-elimination is a technique to reduce the message complexity of MPC-protocols. Hirt et al. describe in [HMP00] a *framework for efficient resilient protocols* where they present the general procedure to take advantage of the player-elimination technique. We will repeat the structure introduced by them and extend it with the definition of *modules*, which allow a finer structuring and modularization of multi-party computation protocols.

Protocols secure against an active adversary generally have a higher communication complexity than their private, non-resilient counterparts, because expensive checks and fault-recovery steps are necessary to ensure that the output is correct even if a corrupted player deviates from the protocol. Player-elimination reduces the number of fault-recoveries that must be performed in the worst case: Whenever a deviation of the protocol is detected, a set of players with a certain number of corrupted players and possibly some honest players is localized and all players of the given set are eliminated. I.e., the eliminated players will not take part in the on-going computation. Fault-recovery is then done by repeating the part of the protocol where the fault was detected. As always at least one corrupted player is eliminated before the fault-recovery, the number of repetitions is bounded by the maximal number of corrupted players. Because we know that fault-recovery will occur infrequently, and it is simply done by repeating part of the protocol, the frequency of the consistency checks can be reduced as well. However, we will have to ensure that delaying the fault-detection will not threaten the security of the protocol.

Following the framework for efficient resilient protocols [HMP00] we divide a protocol into segments of about equal communication complexities.

Definition 1 (Player-Elimination MPC-Protocol). A player-elimination multi-party protocol consists of the initialization I followed by k computation segments followed by the output segment O :

$$[I, S_1, \dots, S_k, O]$$

In the initialization, the *set of non-eliminated players* \mathcal{P}' is made to contain all n players: $\mathcal{P}' = \mathcal{P} = \{P_1, \dots, P_n\}$, $n' = n$, $t' = t$. Each computation segment consists of the following steps:

1. **Private computation.** The current segment of the protocol is computed. The computation must be verifiable, but not robust. This means, that at the end, it must be possible to check whether any faults occurred or not. But if faults occurred, the output is allowed to be incorrect. Furthermore, secrecy must be preserved even if players deviate from the protocol.
2. **Fault detection.** The goal of fault detection is to reach agreement on whether or not a fault occurred during the current segment. The following Steps 3 to 5 are performed only if a fault is detected.
3. **Fault localization.** The purpose of fault localization is to find a set $\mathcal{S} \in \mathcal{P}$ of size $|\mathcal{S}| = p$ guaranteed to contain at least r corrupted players and that the (honest) players agree on \mathcal{S} .
4. **Player elimination.** The players in \mathcal{S} are eliminated from \mathcal{P}' and the further computation is performed without them. (Eliminated players may still send and receive messages to give input and receive output but will never be allowed to complain about a misbehavior of some player again.) In general, the protocol cannot be continued immediately, but it must be transformed to capture the new setting with $n' - p$ players and at most $t' - r$ corrupted players.
5. **Fault correction.** Fault-recovery is done by repeating the current segment with the new setting.

The number and size of the segments is optimally chosen such that the fault detections and the overhead of the (up to t) repetitions do not dominate the total complexity of the protocol. Let k be the number of segments and c the total communication complexity of the private protocol. If all segments are of equal size $\mathcal{O}(c/k)$ then the cost of the repetitions is $\mathcal{O}(tc/k)$. Therefore if we choose $k = \Omega(t)$ and polynomial in n , the number of fault detection steps is independent of c and the cost for correcting faults by repeating segments is $\mathcal{O}(c)$.

2.1 PE-Modules

We introduce the concept of modules, which allow a further structuring of player-elimination protocols. A player-elimination module is a collection of protocols that stands for itself and can be used in the player-elimination framework to compose more complex protocols, similar to procedures of computer languages. A player-elimination module aims to compute the same as some given distributed, non-resilient protocol. It uses the techniques of player-elimination to provide resilience against active misbehavior of some players and be efficient in terms of communication complexity at the same.

Definition 2 (PE-Module). A *PE-module* (δ, γ) -secure against an active adaptive t -threshold adversary which provides resilience for a given private n -player protocol π is a collection of three n -player protocols [PC, WFD, FL]. These three protocols are parameterized by the set of non-eliminated players \mathcal{P}' . Only players in \mathcal{P}' do the actual computation. Players not in \mathcal{P}' may interact during PC but only in such a way that it cannot tamper the security of the protocol.

The protocols are non-deterministic: Each player has access to a random source which provides uniformly random bits. PC, WFD and FL have the following input and output:

PC (Private Computation) Protocol with input x_i for each player $P_i \in \mathcal{P}$. At the end of the protocol, each player $P_i \in \mathcal{P}$ has a string y_i as output and each player $P_i \in \mathcal{P}'$ has possibly some additional auxiliary string z_i . We say that the computation has been correct if the output of PC has the same characteristics as in the private protocol π .

WFD (Weak Fault Detection) Protocol with input x_i, y_i and z_i for player $P_i \in \mathcal{P}'$. P_i 's output of the protocol is the bit b_i and possibly some auxiliary string \bar{z}_i , where $b_i = 1$ indicates that player P_i has detected an inconsistency in the inputs, $b_i = 0$ that this has not occurred.

FL (Fault Localization) Protocol with input $x_i, y_i, b_i, z_i, \bar{z}_i$ and index v of some player $P_v \in \mathcal{P}'$ (who we call *complaining player*) for each player $P_i \in \mathcal{P}'$ and output a pair¹ of players $\mathcal{S} \subseteq \mathcal{P}'$ (known to all players in \mathcal{P}).

We consider the execution of PC(\mathbf{x}) on some module input $\mathbf{x} = [x_1, \dots, x_n]$ followed by WFD(PC(\mathbf{x})) and, if thereby exists an index j with $b_j = 1$, also FL(WFD(PC(\mathbf{x})), j). The protocols have to ensure the following properties irrespectively of the behavior of the adversary:

- *Secrecy*: PC must preserve secrecy except with probability γ and WFD must always preserve secrecy. I.e., the adversary must not get any information that a passive adversary in the private protocol could not get as well.
- *Verifiability*: If PC has incorrect computation then with probability at least $1 - \delta$ all players honest during WFD output the bit 1.
- *Completeness*: If no player is corrupted during PC then the output of PC must be correct. Furthermore, if also no player is corrupted during WFD then all players must output the bit 0 in WFD.
- *Blame assigning*: If FL gets executed, i.e., there is at least one bit $b_v = 1$ (so player P_v complains), then with probability at least $1 - \delta$ the output pair of FL contains at least one corrupted player.

Note that we do not require the protocol FL to preserve the secrecy of the module input. This is an additional property which most module will satisfy.

Definition 3. A player-elimination module has *secrecy-preserving fault localization* if during the execution of its protocol FL the adversary gets no further information about the players' module inputs or outputs.

If some module does not have secrecy-preserving fault localization however, it is the duty of the module user to make sure that the resulting overall protocol preserves secrecy even if the protocol FL opens some of the module inputs or outputs.

¹Depending on the communication model, it might also be meaningful to allow larger sets of players. However, all protocols known to us which make use of player-elimination always eliminate pairs of players.

The auxiliary strings z_i and \bar{z}_i , that player P_i computes, are used during the weak fault detection and fault localization, but are discarded once these protocols have finished. An honest player can output 1 in the WFD even though the computation has been correct. This can happen when a corrupted player tells an honest player that he detected an inconsistency even though this is not true.

2.2 Segment as Module

Using the definition of modules, we can define a segment of a protocol in the PE-framework by specifying the main module $[\text{PC}, \text{WFD}, \text{FL}]$ and the transformation protocol Trans . The main module must have secrecy-preserving fault localization as defined above. The execution of the individual segment steps works then as following:

1. **Private computation.** The protocol PC is executed on segment input $[x_1, \dots, x_n]$.
2. **Fault detection.** The protocol WFD is executed on the output of PC. Each player $P_i \in \mathcal{P}'$ broadcasts his output bit b_i of WFD. The following Steps 3 to 5 are executed if and only if at least one of the bits is non-zero.
3. **Fault localization.** Let $v \in \{1, \dots, n'\}$ be the smallest index such that b_v is non-zero. The protocol FL is executed on the output of PC and WFD and with complaining player P_v resulting in the player set \mathcal{S} .
4. **Player elimination.** The players in \mathcal{S} are eliminated: $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \mathcal{S}$, $n' \leftarrow n' - 2$, $t' \leftarrow t' - 1$. The protocol Trans is executed to capture the new setting.
5. **Fault correction.** The possibly incorrect values are discarded and the current segment of the protocol is repeated with the same inputs.

The protocol Trans is generally needed to do the transformation from the old setting to the new one with fewer players. For example, values that are secret-shared among the old set of players may have to be converted to be a proper sharing among the new set of players.

2.3 Sequential Composition

As mentioned earlier, we want to use PE-modules as building blocks for larger protocols. Therefore, it is important that a PE-module itself can invoke other modules, so-called sub-modules. Care must be taken not to violate the secrecy of the super-modules thereby.

Let us consider the following situation: We want to compose two modules $\text{Sub1} [\text{PC}_{\text{Sub1}}, \text{WFD}_{\text{Sub1}}, \text{FL}_{\text{Sub1}}]$ and $\text{Sub2} [\text{PC}_{\text{Sub2}}, \text{WFD}_{\text{Sub2}}, \text{FL}_{\text{Sub2}}]$ sequentially, that is, the input of Sub2 depends on the output of Sub1, to get a new PE-module Super . We will describe how this can be achieved. It is sufficient to only consider the composition of two sub-modules as the composition of an arbitrary number of sub-modules can be realized as iterated composition of two

sub-modules. Furthermore, any additional computation that we might want to perform in module Super can be wrapped in a separate sub-module.

The weak fault detection of module Super must invoke the weak fault detection of both sub-modules to verify the correctness of their outputs. Furthermore, in the protocol FL_{Super} , it must be ensured that the players find out where the *first* inconsistency occurred before trying to localize a set of corrupted players, because an incorrect output of sub-protocol Sub1 can cause inconsistencies in the computation of the honest players in Sub2.

In general, the secrecy of PC_{Sub2} depends on the correctness of the execution of protocol PC_{Sub1} . Therefore, WFD_{Sub1} has to be executed immediately after PC_{Sub1} (or at the latest before the output of PC_{Sub1} is used in further computations). Additionally, it must be ensured that when an honest player detects an inconsistency, all honest players continue the computation with dummy values as the inconsistent state could reveal information about the secrets if the players continued the computation with the real values. This can be achieved as outlined in the following protocols of module Super.

Module Super. This PE-module is a collection of protocols $[\text{PC}_{\text{Super}}, \text{WFD}_{\text{Super}}, \text{FL}_{\text{Super}}]$ which sequentially composes the sub-modules Sub1 and Sub2 .

Protocol PC_{Super}

1. The players execute the protocol PC_{Sub1} .
2. The players execute the protocol WFD_{Sub1} to check whether the output of PC_{Sub1} is correct and each $P_i \in \mathcal{P}'$ player sends his output bit b_i of the protocol to all players in \mathcal{P}' .
3. If a player P_i gets informed about an inconsistency, i.e., he receives some bit $b_v = 1$ from player P_v , he sets the bit $b_i^{(1)} = 1$ and memorizes the index v . If P_i detected the inconsistency himself, he “informs” himself, i.e., sets $b_i^{(1)} = 1$ and $v = i$. In either case, P_i , “aborts” the computation then in the following way: Instead of further computing the values according to the protocol, he sends random values whenever the protocol tells him to send some value to an other player. On the other hand, if P_i does not get informed about any inconsistency he sets $b_i^{(1)} = 0$.
4. The players execute the protocol PC_{Sub2} .

Out: For each player $P_i \in \mathcal{P}'$, $b_i^{(1)}$, and if $b_i^{(1)} = 1$ also index v , belong to the auxiliary values.

Protocol $\text{WFD}_{\text{Super}}$

1. The players execute the protocol WFD_{Sub2} to check whether the output of PC_{Sub2} is correct. Let the output bit of the protocol be $b_i^{(2)}$ for each player $P_i \in \mathcal{P}'$.
2. Player $P_i \in \mathcal{P}'$ outputs $b_i = b_i^{(1)} \vee b_i^{(2)}$ where $b_i^{(1)}$ is the bit computed during PC_{Super} .

Protocol FL_{Super} Let P_v be the complaining player.

1. Each player $P_i \in \mathcal{P}'$ broadcasts the smallest index $k \in \{1, 2\}$ with $b_i^{(k)} = 1$ or \perp if there is no such index. The players choose the index v' of the player who broadcasted the smallest value k . (If all players broadcasted \perp then P_v is certainly corrupted as he is suddenly not complaining about an inconsistency any more and $\mathcal{S} = \{P_v\}$).
2. $P_{v'}$ broadcasts the index v'' of the player $P_{v''}$ who has informed him about the inconsistency. Player $P_{v''}$ then either accepts or rejects this by broadcast. If he accepts, the players continue with protocol FL_{Subk} and complaining player $P_{v''}$. Otherwise, $P_{v'}$ and $P_{v''}$ are disagreeing about the existence of an inconsistency and the output of FL_{Super} is $\mathcal{S} = \{P_{v'}, P_{v''}\}$.

Lemma 1. *Assuming sub-module Sub1 and Sub2 satisfy Definition 2 for PE-modules with security parameters (δ_1, γ_1) and (δ_2, γ_2) , then module Super satisfies the definition as well, with $\delta' = \max\{\delta_1, \delta_2\}$ and $\gamma' = \gamma_1 + \max\{\gamma_2, \delta_1\}$. Furthermore, if both sub-modules have secrecy-preserving fault localization, then module Super has this property as well.*

Proof. Secrecy: The probability that the secrecy is violated during PC_{Sub1} is at most γ_1 and, if the input of PC_{Sub2} is correct, secrecy is violated during PC_{Sub2} with probability at most γ_2 , resulting in the probability at most $\gamma_1 + \gamma_2$ that either of this happens. On the other side, if the output of PC_{Sub1} is incorrect – and therefore the input of PC_{Sub2} incorrect – this gets detected with probability at least $1 - \delta_1$ and during the execution PC_{Sub2} the secrecy cannot be violated then, as all honest players execute the protocol dummy values. So, in this case, the probability that the secrecy is violated by the executing PC_{Super} is at most $\gamma_1 + \delta_1$. WFD_{Super} always preserves secrecy by the assumption that WFD_{Sub2} does this. Clearly, when FL_{Sub1} and FL_{Sub2} reveal no further information about the secrets to the adversary, FL_{Super} does this neither.

Verifiability: If the adversary tries to manipulate the output of PC_{Sub1} this will be detected with probability at least $1 - \delta_1$ by the execution of WFD_{Sub1}. Otherwise, if the adversary tries to cheat in PC_{Sub2} this is detected by the execution of WFD_{Sub2} with probability at least $1 - \delta_2$, resulting in an overall probability at most $\max\{\delta_1, \delta_2\}$ for WFD_{Super} to overlook an incorrect output.

Completeness: Follows from the completeness of the sub-modules.

Blame assigning: By inspecting the protocols, we see that if FL_{Sub1} or FL_{Sub2} is invoked, this always happens for the first inconsistency detected and with a complaining player that actually detected an inconsistency, so the blame assigning follows from the properties of the sub-modules. Otherwise, if FL_{Super} outputs $\mathcal{S} = \{P_{v'}, P_{v''}\}$, $P_{v'}$ or $P_{v''}$ must be corrupted as either $P_{v'}$ untruly claims that he got informed by $P_{v''}$ about an inconsistency or $P_{v''}$ disputes an inconsistency previously reported to $P_{v'}$. \square

2.4 Parallel Composition

Often, we want to execute the same protocol on different inputs in parallel, i.e., the input of each run does not depend on the output of the other runs. In this case, it is likely that the weak fault detection can be performed more efficient

for all executions at once than for each one individually. Therefore, we give a separate definition for a player-elimination module whose weak fault detection can be executed after an arbitrary number of private protocol executions.

Following definition describes the parallel self-composition of a PE-module with merged weak fault detection.

Definition 4 (Parallel PE-Module). A *parallel player-elimination module* (δ, γ) -secure against an active adaptive t -threshold adversary which provides resilience for a given private n -player protocol π is a collection of three n -player protocols $[\text{PC}, \text{WFD}^m, \text{FL}^m]$. Protocols WFD^m and FL^m are executed (if at all) once after m parallel executions of PC . Protocol PC has the same input and output as in the non-parallel version, protocols WFD^m and FL^m are modified as following:

WFD^m (Weak Fault Detection) Protocol with input $x_i^{(1)}, \dots, x_i^{(m)}, y_i^{(1)}, \dots, y_i^{(m)}$ and $z_i^{(1)}, \dots, z_i^{(m)}$ for player $P_i \in \mathcal{P}'$. P_i 's output of the protocol is the bit b_i and possibly some auxiliary string \bar{z}_i where $b_i = 1$ indicates that player P_i has detected an inconsistency in the inputs, $b_i = 0$ that this has not occurred.

FL^m (Fault Localization) Protocol with input $x_i^{(1)}, \dots, x_i^{(m)}, y_i^{(1)}, \dots, y_i^{(m)}, b_i, z_i^{(1)}, \dots, z_i^{(m)}, \bar{z}_i$ and index v of some player $P_v \in \mathcal{P}'$ (who we call *complaining player*) for each player $P_i \in \mathcal{P}'$ and output a pair of players $\mathcal{S} \subseteq \mathcal{P}'$ (known to all players in \mathcal{P}).

We consider the parallel execution of $\text{PC}(\mathbf{x}^{(k)})$ on k inputs $\mathbf{x}^{(k)} = [x_1^{(k)}, \dots, x_n^{(k)}]$, $k = 1 \dots m$, followed by the merged weak fault detection $\text{WFD}^m(\text{PC}(\mathbf{x}^{(1)}), \dots, \text{PC}(\mathbf{x}^{(m)}))$ and, in the case that thereby exists an index j with $b_j = 1$, also $\text{FL}^m(\text{WFD}^m(\text{PC}(\mathbf{x}^{(1)}), \dots, \text{PC}(\mathbf{x}^{(m)})), j)$. As in the non-parallel version, the protocols have to ensure the properties secrecy, verifiability and blame assigning irrespectively of the behavior of the adversary. Thereby, the verifiability means to detect incorrect computation in *any* of the private computations with the probability $1 - \delta$.

Chapter 3

IC-Signatures

In this Chapter, we specify protocols that allow players to create information theoretically secure signatures in a multi-player setting. As we want to use them in our PE-framework, we will present them as PE-modules.

3.1 Information Checking (IC)

Information checking is an information theoretically secure method for authenticating data ([CDD⁺99]). An IC-scheme is a collection of two protocols `DistVal` and `RevealVal` which describe the communication between the dealer P_d , the proofer P_q and the verifier P_v . In `DistVal` the dealer hands some secret to P_q and some auxiliary data to both P_q and P_v . Players P_q and P_v then verify the consistency of this auxiliary data. The purpose of protocol `RevealVal` is that player P_q can proof to P_v that he correctly opens the secret that P_d has sent him before – even if P_d has been corrupted in `DistVal`.

The two protocols `RevealVal` and `DistVal` must have the following properties correctness and secrecy where δ is the security parameter.

Correctness:

1. If P_d , P_q and P_v are honest, and P_d has secret s , then P_v will accept s in `RevealVal`.
2. If P_q and P_v are honest, then after `DistVal`, P_q knows a value s such that P_v will accept s in `RevealVal` (except with probability δ).
3. If P_d and P_v are honest, then in `RevealVal`, player P_v will reject every value s' different from s with probability at least $1 - \delta$.

Secrecy: If P_d and P_q are honest and before `RevealVal` has been initiated, P_v has no information about the secret s .

3.2 IC in the PE-Framework

To integrate the idea of an IC-scheme into the player-elimination framework, i.e. to create PE-modules, we drop the robustness of `DistVal` and allow this protocol to fail. So, the players execute a fault detection protocol at the end of `DistVal`

to check whether the distribution of the shares and auxiliary values succeeded or not. If an inconsistency is detected thereby, the players abort and a set of two players with at least one of them being corrupted is localized. If the check succeeds, however, the same properties as in a robust IC-scheme hold.

Summarizing, this means that the correctness will only hold when the fault localization detects no inconsistencies but the player-elimination IC-scheme additionally has the following properties (which we already know from the definition of PE-modules and are adapted to the IC-scheme here):

Verifiability: If P_d tries to distribute inconsistent auxiliary data, then honest players P_q and P_v will detect this during the fault detection with probability at least $1 - \delta$.

Completeness: If no player is corrupted then correctness is satisfied and the fault detection always succeeds.

Blame assigning: If the fault detection finds some inconsistency then there is a fault localization protocol which outputs a set of two players with at least one of them being corrupted.

We specify protocols to perform information checking in the PE-framework. Thereby, P_d , P_q and P_v are players of the set \mathcal{P}' . All computations are done in some Galois field $K = GF(q)$.

The idea behind the construction is that P_d chooses a random line which interpolates the secret at position 0. The dealer hands two points on the line to P_q , and one point on the line to P_v at some position α unknown to P_q (Figure 3.1). Like this, P_v gets no information about the secret but receives some verification value which is unknown to P_q .

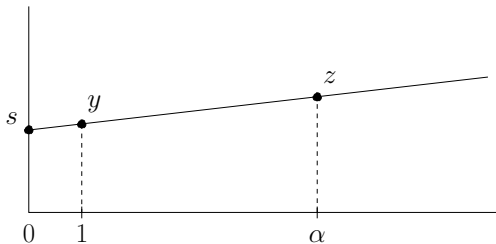


Figure 3.1: Illustration of IC-checking.

Definition 5. A tuple $[x, y, z] \in K^3$ is 1_α -consistent if there exists a degree-1 polynomial p over K such that $p(0) = x$, $p(1) = y$ and $p(\alpha) = z$.

Module ICDistVal. This PE-module is a collection of protocols $[\text{PC}_{ICDistVal}, \text{WFD}_{ICDistVal}, \text{FL}_{ICDistVal}]$. It transfers a secret $s \in K$ given by player $P_d \in \mathcal{P}'$ to player $P_q \in \mathcal{P}'$ in such a way that later on, using the protocol ICRevealVal , P_q can transfer s to player $P_v \in \mathcal{P}'$ and proof to him that this was the value which P_d had given him.

Protocol $\text{PC}_{\text{ICDistVal}}$

In: Secret s for player P_d .

1. P_d chooses randomly a value $\alpha \in K \setminus \{0\}$ and sends α to P_v .
2. P_d chooses randomly a 1_α -consistent triple $[s, y, z] \in K^3$ and sends s, y to P_q and z to P_v .

Out: Values s, y for player P_q and α, z for player P_v .

During the fault detection, player P_v acts as verifier. The idea is to reconstruct a random scaling of the line that shares the secret, blinded by the addition of a random line chosen by P_d .

Protocol $\text{WFD}_{\text{ICDistVal}}$

1. P_d chooses randomly a 1_α -consistent triple $[s', y', z'] \in K^3$ and sends s', y' to P_q and z' to P_v .
2. P_q chooses randomly a value $d \in K$ and sends $d, s' + ds, y' + dy$ to P_v .
3. P_v checks whether $[s' + ds, y' + dy, z' + dz]$ is 1_α -consistent and outputs $b_v = 0$ if yes or $b_v = 1$ if not. All other players $P_i \in \mathcal{P}'$ have fixed output $b_i = 0$.

Protocol $\text{FL}_{\text{ICDistVal}}$ P_v is the complaining player.

1. P_q broadcasts the values $d, s' + ds$ and $y' + dy$.
2. P_v checks then whether what P_q broadcasted is the same as what P_q sent to him before. If not, he broadcasts the bit 1 and $\mathcal{S} = \{P_q, P_v\}$. Else, P_v broadcasts 0 and it is up to the dealer P_d to decide if the accusation of P_v was correct, i.e., if the broadcast values of P_q were invalid. P_d broadcasts his decision. If P_d gives right to P_v then $\mathcal{S} = \{P_d, P_q\}$ otherwise $\mathcal{S} = \{P_d, P_v\}$.

Protocol ICRevealVal

In: Values s, y for player P_q and α, z for player P_v .

1. P_q sends the value s and “proof” y to player P_v .
2. P_v verifies that $[s, y, z]$ is 1_α -consistent and accepts or rejects accordingly.

Out: P_v either accepts or rejects P_q 's proof.

Lemma 2. *Module ICDistVal satisfies Definition 2 for a PE-module and together with protocol ICRevealVal , it builds an IC-scheme in the PE-framework, where $\delta = 1/(q - 1)$ and $\gamma = 0$. The module has additionally secrecy-preserving fault localization.*

Proof. Secrecy: We assume that P_d and P_q are honest, as otherwise, the adversary already knows the secret. Protocol $\text{PC}_{\text{ICDistVal}}$ preserves the secrecy as the value z sent to player P_v is independent of the secret s . During $\text{WFD}_{\text{ICDistVal}}$, player P_v learns the values z' , d , $s' + ds$ and $y' + dy$. Since P_d and P_q are honest, the triple $[s' + ds, y' + dy, z' + dz]$ is 1_α -consistent and P_v can compute $y' + dy$ from $s' + ds$ and $z' + dz$. Therefore $y' + dy$ does not give P_v additional information. However, it is clear that $z, z', d, s' + ds$ has distribution independent of s . During $\text{FL}_{\text{ICDistVal}}$, player P_v learns no value new value, so $\text{FL}_{\text{ICDistVal}}$ clearly preserves secrecy as well.

Verifiability: Assume player P_d distributed a triple $[s, y, z]$ which is not 1_α -consistent. Then, for each triple $[s', y', z']$ there is exactly one value d such that $[s' + ds, y' + dy, z' + dz]$ is 1_α -consistent. As P_q chooses d independent of s', y', z' , the probability that $[s' + ds, y' + dy, z' + dz]$ is 1_α -consistent is at most $1/q$. Therefore, an honest P_v will complain with probability $1 - 1/q > 1 - \delta$.

Completeness: It is easy to see that if all players stick to the protocol the verification always succeeds and P_v always accepts in ICRevalVal .

Correctness: We need to show that the correctness holds when P_v outputs $b_v = 0$ in $\text{WFD}_{\text{ICDistVal}}$. The first point of correctness is already proven by the completeness. For the second point, remember that $[s, y, z]$ is 1_α -consistent with probability at least $1 - \delta$ if $\text{WFD}_{\text{ICDistVal}}$ succeeds. So, an honest P_v will accept s, y with the same probability. The third point follows from the fact that P_q does not know α : If P_q sends values \tilde{s}, \tilde{y} where $\tilde{s} \neq s$ then P_v will accept if $[\tilde{s}, \tilde{y}, z]$ is 1_α -consistent. We know that $[s, y, z]$ is 1_α -consistent by its definition, so $[s - \tilde{s}, y - \tilde{y}, 0]$ is as well. This gives a non-trivial degree-1 equation from which α can be computed. In other words, P_q must guess α to have P_v accept a false value which he can do with probability at most $1/(q - 1) = \delta$.

Blame assigning: If P_q broadcasts different values than he has sent privately to P_v , he is certainly corrupted and the set $\mathcal{S} = \{P_q, P_v\}$ contains a corrupted player if P_v complains about this. Otherwise, the dealer can assume that the broadcast values are the ones that P_q sent to P_v during $\text{WFD}_{\text{ICDistVal}}$ and as he knows all correct values he can decide if P_q sent incorrect values or if P_v 's accusation was unjustified. So an honest P_d always finds a corrupted player $P_i \in \{P_v, P_q\}$ and the set $\mathcal{S} = \{P_d, P_i\}$ satisfies the requirement. \square

3.3 IC-Signatures

Information checking can be used to create information theoretically secure signatures in a multi-party setting. This means that the dealer can send some secret s to a player and sign it in such a way that the receiver can use the received signature later on to proof to the other players that the dealer has sent him the value s . The difference to an ordinary signature is that the signing procedure is a multi-party computation where besides the dealer and the receiver all player which should be able to act as verifier later on are involved – which also means that the set of potential verifiers must already be known at the time of signing. In our player-elimination framework this set will be the set of non-eliminated players \mathcal{P}' .

A player-elimination IC-signature of dealer P_d for the value s given to P_q simply consists of the parallel execution of n' IC-checkings where always P_d is

the dealer, P_q the proofer and each player in \mathcal{P}' acts as verifier once. When the signature is to be opened, the protocol **ICRevealVal** is invoked for all IC-checkings and each player accepts or rejects the signature accordingly.

Module ICSign. This PE-module is a collection of protocols [PC_{ICSign} , WFD_{ICSign} , FL_{ICSign}] which allows a dealer $P_d \in \mathcal{P}'$ to send some secret $s \in K$ to player $P_q \in \mathcal{P}'$, together with an IC-signature for s .

Protocol PC_{ICSign}

In: Secret s for player P_d .

1. For $i = 1, \dots, n'$, the protocol $\text{PC}_{ICDistVal}$ is executed with dealer P_d , proofer P_q , verifier P_i and secret s .

Out: Value s for P_q and auxiliary values for all player in \mathcal{P}' .

Protocol WFD_{ICSign}

1. For $i = 1, \dots, n'$, the protocol $\text{WFD}_{ICDistVal}$ is executed with dealer P_d , proofer P_q , verifier P_i and secret s to verify all executions of $\text{PC}_{ICDistVal}$.
2. P_q checks whether P_d always performed the IC-checking for the same secret s and outputs $b_q = 0$ if this holds or $b_q = 1$ else. Each other player $P_i \in \mathcal{P}'$ outputs the bit b_i of the i -th execution of protocol $\text{WFD}_{ICDistVal}$ in Step 1. (In all the other executions, P_i had fixed output $b_i = 0$.)

Protocol FL_{ICSign} Let P_i be the complaining player.

1. If $i = q$ then $\mathcal{S} = \{P_d, P_q\}$. Otherwise, $\text{FL}_{ICDistVal}$ is invoked for the i -th execution of $\text{PC}_{ICDistVal}$ and $\text{WFD}_{ICDistVal}$ (where P_i acted as verifier).

Lemma 3. *The module ICSign satisfies Definition 2 for a PE-module with $\delta = 1/(q - 1)$, $\gamma = 0$ and has secrecy-preserving fault localization.*

Proof. Secrecy: The secrecy of the three protocols that define the module ICSign follows directly from the secrecy of the used sub-module ICDistVal as no further values are sent.

Verifiability: By the properties of sub-module ICDistVal, each receiver will detect an inconsistency in the output of $\text{PC}_{ICDistVal}$ during $\text{WFD}_{ICDistVal}$ with probability at least $1 - \delta$.

Completeness: Follows directly from the completeness of sub-module ICDistVal.

Blame assigning: If the output of FL_{ICSign} is $\mathcal{S} = \{P_d, P_q\}$ this means that P_d complains about a corrupted dealer P_d , so \mathcal{S} contains at least one corrupted player. Otherwise, protocol $\text{FL}_{ICDistVal}$ is only invoked for instances where the receiver was complaining. Therefore, blame assigning follows from the properties of the sub-module.

□

Protocol ICOpenSig Allows a player $P_q \in \mathcal{P}'$ to open an IC-signature for some secret s which he has received from some player $P_d \in \mathcal{P}'$ and to convince all honest players in \mathcal{P}' that s is indeed the perviously received value.

1. For $i = 1, \dots, n'$ the protocol ICRevealVal is executed with proofer P_q , verifier P_i and secret s , whereby P_i either accepts or rejects the proof.

It follows from the correctness of the IC-scheme that each honest player will always accept the value s and only the value s in ICOpenSig (except with probability δ).

3.4 Complexity Analysis

From inspecting the protocols, it follows that the modules and protocols presented in this Chapter have following communication complexities where $\kappa = \log_2 q$ and we use that $\kappa \geq \log_2 n'$. The numbers for total do not include the n' bits broadcast during the fault detection when the module is used as a segment in a player-elimination protocol (see Section 2.2). All values are upper bounds:

	ICDistVal
PC	4κ
WFD	6κ
FL	$\text{BC}(3\kappa) + 2 \text{BC}(1)$
total	$10\kappa + \text{BC}(3\kappa) + 2 \text{BC}(1)$

Table 3.1: Communication complexities of module ICDistVal.

	ICSign
PC	$4n'\kappa$
WFD	$6n'\kappa$
FL	$\text{BC}(3\kappa) + 2 \text{BC}(1)$
total	$10n'\kappa + \text{BC}(3\kappa) + 2 \text{BC}(1)$

Table 3.2: Communication complexities of module ICSign.

ICRevealVal	2κ
ICOpenSig	$2n'\kappa$

Table 3.3: Communication complexities of protocols ICRevealVal and ICOpenSig.

Chapter 4

Player-Elimination Secret Sharing

4.1 (Verifiable) Secret Sharing

The concept of secret sharings was introduced by Shamir [Sha79]. A secret-sharing scheme allows a dealer to distribute some secret among a set of players by privately sending a share to each player. Later on, only qualified subsets of the player set can reconstruct the secret by pooling their shares. The set of qualified subsets is called access structure and is just the complement of the previously introduced adversary structure. There are secret-sharing schemes which allow general access structures but in this thesis we will only consider threshold structures. This means that a player set is qualified if its size is at least $t + 1$ where t is the threshold parameter. Such a secret-sharing scheme must ensure *secrecy*, i.e., an adversary who sees up to t shares learns no information about the secret and *correctness*, which means, the secret can always be reconstructed from a set of at least $t + 1$ shares.

In the following, we will focus on the secure-channels model with broadcast and an active threshold adversary which may corrupt any minority of the players (as defined in Section 1.1). In presence of such an adversary, the corrupted players may contribute incorrect shares during the reconstruction phase. As a consequence of this, it is not possible to construct a secret-sharing scheme where the correct secret gets always reconstructed [CDF01], so we allow a small error probability. Furthermore, as also the dealer can be corrupted, we need a stronger definition of the correctness which ensures consistency of the shared value. More formally, this leads to

Definition 6 (verifiable secret sharing (VSS)). Assume the presence of an active adaptive adversary that can corrupt at most t out of n players. A $(1 - \delta)$ -secure VSS scheme for sharing a secret s (an element of some field K) is a pair of protocols [Sh, Rec] that satisfies the following secrecy, correctness and completeness properties:

- *Secrecy:* If the dealer is honest and before Rec has been started, the adversary has no information about the secret s .

- *Correctness*: Once all players complete **Sh**, there exists a fixed value $r \in K$ such that with probability at least $1 - \delta$, during **Rec**, each uncorrupted player outputs r .
- *Completeness*: If the dealer was honest then, with probability at least $1 - \delta$, each uncorrupted player outputs $r = s$ during **Rec**.

4.2 Pre-Sharing

We present protocols for secret sharing adapted to the player-elimination framework. In contrast to an ordinary VSS scheme, the protocols for sharing and reconstructing a secret can fail. On success, the protocols ensure the same properties as a VSS. However, on failure, only the secrecy is preserved and the output of the protocols is a pair of players with at least one of them being corrupted (agreed on by all players). The idea is to take a passive secret sharing and to do additional checks to ensure correctness in the case that the checks succeed or abort the protocol otherwise.

First, we present Shamir's simple threshold scheme [Sha79] which allows the dealer to share elements of some Galois field $K = GF(q)$ which must have size at least $q \geq n + 1$ where n is the number of players.

Definition 7 (pre-sharing). We call a secret $s \in K$ *pre-shared* among the player set $\mathcal{P} = \{P_1, \dots, P_n\}$ (where at most $t < n/2$ players are corrupted) when the following holds: There exists a degree- t polynomial $f(x)$ with $f(0) = s$ and each player $P_i \in \mathcal{P}$ knows share $s_i = f(i)$.

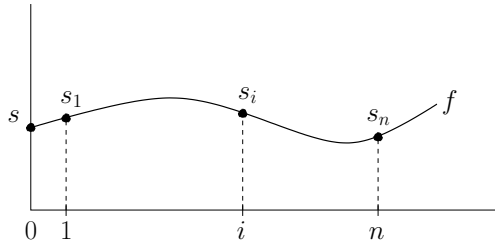


Figure 4.1: Pre-shared secret s with polynomial f and shares s_1, \dots, s_n .

A pre-shared secret s is secure against a passive adversary which can see up to t shares: Given any set of t shares and some secret candidate s' , there is exactly one polynomial that interpolates both the t shares and the secret. As the dealer chooses the polynomial which shares the secret uniform at random, each polynomial is equally likely and the adversary gets no information about the secret s at all.

We give PE-modules to pre-share and reconstruct secrets:

Module PreShare. This parallel PE-module is a collection of protocols $[PC_{PreShare}, WFD_{PreShare}^m, FL_{PreShare}^m]$. It allows a dealer $P_d \in \mathcal{P}'$ to pre-share values $s^{(1)} \in K, \dots, s^{(m)} \in K$ using the sharing described in Definition 7.

The protocol $\text{PC}_{\text{PreShare}}$ pre-shares a single value and is executed m times in parallel before verifying all m sharings at once using $\text{WFD}_{\text{PreShare}}^m$.

Protocol $\text{PC}_{\text{PreShare}}$

In: Value $s \in K$ for the dealer $P_d \in \mathcal{P}'$.

1. P_d chooses a random degree- t' polynomial $f(x)$ with $f(0) = s$ and sends the share $s_i = f(i)$ to player P_i , $i = 1, \dots, n'$.

Out: Polynomial f for P_d and share s_i for each player $P_i \in \mathcal{P}'$.

During the weak fault detection, each player in \mathcal{P}' acts as verifier. The idea of the verification is to reconstruct a random linear combination of the shared values towards each verifier, who can check like this with probability $1 - \delta$ that all sharings are correct. Thereby, a blinding value must be added to the linear combination such that the verifier gets no information about the secrets.

Protocol $\text{WFD}_{\text{PreShare}}^m$

In: Polynomials $f^{(k)}$ for P_d and values $s_i^{(k)}$ for $P_i \in \mathcal{P}'$, $k = 1, \dots, m$.

1. The dealer shares n' blinding values: P_d chooses uniform random $s^{(m+1)}, \dots, s^{(m+n')}$ and shares them using the protocol $\text{PC}_{\text{PreShare}}$.
2. Each player $P_v \in \mathcal{P}'$ performs the following steps to verify that all distributed shares are consistent:
 - (a) P_v selects a random vector $r_v = [r_v^{(1)}, \dots, r_v^{(m)}] \in K^m$ and sends it to all players in \mathcal{P}' .
 - (b) Each player $P_j \in \mathcal{P}'$ computes the linear combinations of his shares

$$z_j^{(v)} = s_j^{(m+v)} + \sum_{k=1}^m r_v^{(k)} s_j^{(k)}$$

and sends it to P_v .

- (c) P_v verifies that the received shares $z_1^{(v)}, \dots, z_{n'}^{(v)}$ lie on a polynomial of degree at most t' . If so, P_v 's output bit b_v is 0, otherwise it is 1.

Out: Auxiliary values are $r_j, z_j^{(i)}, s_i^{(m+j)}$, held by player P_i , $1 \leq i, j \leq n'$ and additionally $f^{(m+1)}, \dots, f^{(m+n')}$ for P_d .

Protocol $\text{FL}_{\text{PreShare}}^m$ Let $P_v \in \mathcal{P}'$ be the complaining player.

1. The dealer P_d sends the correct linear combination of the polynomials to P_v :

$$g(x) = f^{(m+v)}(x) + \sum_{k=1}^m r_v^{(k)} f^{(k)}(x)$$

2. P_v selects the index $i \in \{1, \dots, n'\}$ of some share $z_i^{(v)}$ which does not lie on the polynomial g which he received from P_d (i.e. $z_i^{(v)} \neq g(i)$) and broadcasts i .

3. For $k = 1, \dots, m, m + v$, P_d and P_i now send the values $f^{(k)}(i)$ and $s_i^{(k)}$, respectively, to P_v . P_v verifies that the linear combination of these values is the same as P_d and P_i have sent to him earlier. If not, P_v adjusts the blinding value $f^{(m+v)}(i)$ and $s_i^{(m+v)}$ accordingly (he is allowed to do this because in this case P_d or P_i is clearly corrupted and could have sent the adjusted values just as well).
4. P_v selects one index $k \in \{1, \dots, m, m + v\}$ where P_d 's value $f^{(k)}(i)$ and P_i 's value $s_i^{(k)}$ differ and broadcasts k .
5. P_d and P_i broadcast $f^{(k)}(i)$ and $s_i^{(k)}$, respectively. If the two values differ, then $\mathcal{S} = \{P_d, P_i\}$.
6. Otherwise, P_v sets either $j = i$ if what P_i broadcasted is different to the value $s_i^{(k)}$ that P_v got in Step 3, or else $j = d$. Then, P_v broadcasts j and $\mathcal{S} = \{P_j, P_v\}$.

Lemma 4. *The module $PreShare$ fulfills Definition 4 for a PE-module with $\delta = 1/q$ and $\gamma = 0$.*

Proof. Secrecy: The protocol $PC_{PreShare}$ fulfills the secrecy trivially as only output values are sent to the players (which they must also learn in the private protocol). The random blinding sharings in $WFD_{PreShare}^m$ make sure that the adversary does not get any further information about the shared values during its execution: The values $z_i^{(j)}$, $1 \leq i, j \leq n'$ give no joint information about the secrets $s^{(1)}, \dots, s^{(m)}$. A corrupted dealer could choose to share non-uniform random blinding values. In this case however, the adversary has learned all secrets anyway by corrupting the dealer.

Verifiability: If the dealer P_d remains honest during $PC_{PreShare}$ the output is always correct. A corrupted dealer however could send shares that do not lie on a polynomial of degree at most t' . $WFD_{PreShare}^m$ checks that the shares lie on degree- t' polynomials in all sharings:

Let $\mathcal{P}_h \subseteq \mathcal{P}'$ be the set of players honest at the end of $WFD_{PreShare}^m$. We show that if the dealer P_d has distributed shares $\{s_i^{(j)} \mid i \in \mathcal{P}_h\}$ for some $j \in \{1, \dots, m\}$ such that these shares interpolate a polynomial of too high degree $l > t$, then each honest player $P_v \in \mathcal{P}_h$ will complain in $WFD_{PreShare}^m$ with probability at least $1 - 1/q$. We consider the polynomial $p(x)$ that interpolates the values $\{z_i^{(v)} \mid i \in \mathcal{P}_h\}$ and calculate the probability that the coefficient of the monomial x^l in $p(x)$ is zero. This is a necessary condition that P_v does not complain about a too high degree and therefore a lower bound for the probability. P_v does not know which values he received from honest players but interpolates the polynomial from all values $\{z_1^{(v)}, \dots, z_{n'}^{(v)}\}$. However, the polynomial interpolating all values cannot be of lower degree than the polynomial interpolating only the values of the honest players.

We can see $p(x)$ as the linear combination of two polynomials $p(x) = p'(x) + r_v^{(j)} p^{(j)}(x)$ where $p'(x)$ is interpolated from the values

$$\left\{ s_i^{(m+v)} + \sum_{k=1, k \neq j}^m r_v^{(k)} s_i^{(k)} \mid i \in \mathcal{P}_h \right\}$$

and $p^{(j)}(x)$ from $\{s_i^{(j)} \mid i \in \mathcal{P}_h\}$. The l -th coefficient of $p'(x)$ is a random variable X depending on P_v 's challenge vector. The l -th coefficient of $p^{(j)}(x)$ is non-zero by assumption. The random variable Y of the l -th coefficient in the polynomial $r_v^{(j)} p^{(j)}(x)$ is uniformly distributed and independent of X as P_v chooses $r_v^{(1)}, \dots, r_v^{(m)}$ uniformly and independently of each other. The sum $X + Y$ in K which describes the l -th coefficient in $p(x)$ is therefore uniformly distributed as well. With that, the probability that P_v does not detect that $p^{(j)}(x)$ has a too high degree is at most $1/q = \delta$.

Correctness: Follows from inspecting the protocol.

Blame assigning: It is sufficient to show that $\text{FL}_{\text{PreShare}}^m$ never outputs a pair of honest players. We distinguish two cases of its output:

1. $P_v \notin S$. This only happens when P_d 's and P_i 's broadcast values in Step 5 differ. This cannot happen when both P_d and P_i are honest, because if so, they will both broadcast the value which P_d has sent to P_i during $\text{PC}_{\text{PreShare}}$.
2. $P_v \in S$. P_d 's and P_i 's broadcast values $f^{(k)}(i)$ and $s_i^{(k)}$ are the same. If P_v is corrupted, the requirement is clearly fulfilled. So, assume that P_v is honest. Player P_v always finds an index i in Step 2 with $z_i^{(v)} \neq g(i)$ as he has detected that $z_1^{(v)}, \dots, z_{n'}^{(v)}$ lie on a polynomial of degree $> t'$ but g is of degree at most t' . Also in Step 4, he can always select an index k where $f^{(k)}(i)$ and $s_i^{(k)}$ differ as otherwise, if all values matched also the linear combinations would match. In Step 6, P_v always finds the index j of some corrupted player, as either P_d or P_i has broadcast something different than P_v got in Step 3.

□

For the reconstruction, we do not present a module to reconstruct a single pre-shared value, but a module which reconstructs a linear combination of m pre-shared values (over the field K) to a single player. We will need this computation as sub-module later on. Even though the operation on m sharings makes the module seem to be a parallel one, this is not true. For each execution of the private computation there is exactly one execution of the weak fault detection.

Module PreRec. This PE-module is a collection of protocols $[\text{PC}_{\text{PreRec}}, \text{WFD}_{\text{PreRec}}, \text{FL}_{\text{PreRec}}]$ that reconstructs the linear combination of pre-shared secrets $s^{(\Sigma)} = \sum_{k=1}^m w^{(k)} s^{(k)}$ to player $P_r \in \mathcal{P}'$ who chooses the coefficient vector $[w^{(1)}, \dots, w^{(m)}] \in (K \setminus \{0\})^m$. All secrets must have been pre-shared by the same player P_d .

Protocol $\text{PC}_{\text{PreRec}}$

In: Share $s_i^{(k)}$ for player $P_i \in \mathcal{P}'$, additionally polynomial $f^{(k)}(x)$ for player P_d and $w^{(k)}$ for player P_r , $k = 1, \dots, m$.

1. Player P_r sends the vector $\mathbf{w} = [w^{(1)}, \dots, w^{(m)}]$ to all players in \mathcal{P}' .

- Each player $P_i \in \mathcal{P}'$ computes and sends the value

$$s_i^{(\Sigma)} = \sum_{k=1}^m w^{(k)} s_i^{(k)}$$

to P_r .

- P_r interpolates the secret $s^{(\Sigma)}$ from the values $s_1^{(\Sigma)}, \dots, s_{n'}^{(\Sigma)}$ using Lagrange interpolation.

Out: Value $s^{(\Sigma)}$. Auxiliary values are the vector \mathbf{w} for each player in \mathcal{P}' and $s_1^{(\Sigma)}, \dots, s_{n'}^{(\Sigma)}$ for player P_r .

Protocol WFD_{PreRec}

- P_r checks whether the interpolated polynomial is of degree at most t' and outputs $b_r = 0$ if yes, $b_r = 1$ otherwise. All other players $P_i \in \mathcal{P}'$ have fixed output $b_i = 0$.

Protocol FL_{PreRec} Only P_r can be complaining.

- P_d , who is accused to have distributed wrong shares, sends the correct linear combination of the polynomials to P_r :

$$f^{(\Sigma)}(x) = \sum_{k=1}^m w^{(k)} f^{(k)}(x)$$

- P_r selects and broadcasts an index $i \in \{1, \dots, n'\}$ with $s_i^{(\Sigma)} \neq f^{(\Sigma)}(i)$.
- For $k = 1, \dots, m$, P_d and P_i now send the values $f^{(k)}(i)$ and $s_i^{(k)}$, respectively, to P_r . P_r verifies that the linear combination of these values is the same as P_d and P_i have sent to him earlier. If not, P_r sets $f^{(1)}(i) = f^{(\Sigma)}(i)/w^{(1)}$, $f^{(2)}(i) = \dots = f^{(m)}(i) = 0$ and $s_i^{(1)} = s_i^{(\Sigma)}/w^{(1)}$, $s_i^{(2)} = \dots = s_i^{(m)} = 0$, respectively (he is allowed to do this because in this case P_d or P_i is clearly corrupted and could have sent the adjusted values just as well).
- P_r selects one index $k \in \{1, \dots, m\}$ where P_d 's value $f^{(k)}(i)$ and P_i 's value $s_i^{(k)}$ differ and broadcasts k .
- P_d and P_i broadcast $f^{(k)}(i)$ and $s_i^{(k)}$, respectively. If the two values differ, then $\mathcal{S} = \{P_d, P_i\}$.
- Otherwise, P_r sets either $j = i$ if what P_i broadcasted is different to the value $s_i^{(k)}$ that P_r got in Step 3, or else $j = d$. P_r broadcasts j and $\mathcal{S} = \{P_j, P_r\}$.

Lemma 5. *The module $PreRec$ fulfills Definition 2 for a PE-module with $\delta = \gamma = 0$.*

Proof. Secrecy: Protocols PC_{PreRec} and WFD_{PreRec} preserve secrecy as only P_r receives shares to reconstruct $s^{(\Sigma)}$ which he is to know according to the private protocol.

Verifiability: At least $t' + 1$ honest players send the correct linear combination of shares to player P_r . These values already uniquely define a degree- t' polynomial and therefore P_r will always notice if any corrupted player sends an incorrect share to him.

Completeness: The correct secret can always be interpolated from the correct shares and the verification in WFD_{PreRec} succeeds.

Blame assigning: It is sufficient to show that FL_{PreRec} never outputs a pair of honest players. We distinguish two cases of its output:

- $P_r \notin \mathcal{S}$: This only happens when players P_d and P_i broadcast different values in Step 5. This clearly does not happen if both are honest as the shares of the honest players are consistent by assumption.
- $P_r \in \mathcal{S}$: P_d 's and P_i 's broadcast values $f^{(k)}(i)$ and $s_i^{(k)}$ are the same. If P_r is corrupted, the requirement is fulfilled. So, we assume that P_r is honest. Player P_r always finds an index i in Step 2 with $s_i^{(\Sigma)} \neq f^{(\Sigma)}(i)$ as he has detected that $s_1^{(\Sigma)}, \dots, s_{n'}^{(\Sigma)}$ lie on a polynomial of degree $> t'$ but $f^{(\Sigma)}$ is of degree at most t' . Also in Step 4, he can always select an index k where $f^{(k)}(i)$ and $s_i^{(k)}$ differ as otherwise, if all values matched also the linear combinations would match. In Step 6, P_r always finds the index j of some corrupted player, as either P_d or P_i has broadcast something different than P_r got in Step 3.

□

4.3 Player-Elimination Secret Sharing

The secret-sharing scheme presented in the previous section is linear: Given two sharings of values a and b from the same dealer, the players can calculate a sharing of $a + b$ without communication. The dealer simply adds the two polynomials which he used to share a and b and each player adds his share for a and b to get a sharing of the sum. However, this does not work for two values shared by different dealers because the WFD and FL of the module $PreRec$ rely on the fact that there is one single player who knows the polynomial that is used for the sharing.

To overcome this problem, we can use the following two-dimensional secret sharing that is composed from pre-shared values. The module $ComputeSharing$ creates such a secret sharing as linear combination of values $x_1, \dots, x_{n'}$ where each player $P_i \in \mathcal{P}'$ can contribute the value x_i . The resulting secret sharing is linear and can be reconstructed without the assistance of a dealer.

Definition 8 (player-elimination secret sharing). We call a secret s *player-elimination secret shared among the player set* $\mathcal{P} = \{P_1, \dots, P_n\}$ (where at most $t < n/2$ players are corrupted) when the following holds: The secret s is pre-shared and each share s_i is again pre-shared by the polynomial $f_i(x)$ where player $P_j \in \mathcal{P}$ knows $f_j(x)$ and $s_{ij} = f_i(j)$ for $i = 1, \dots, n$.

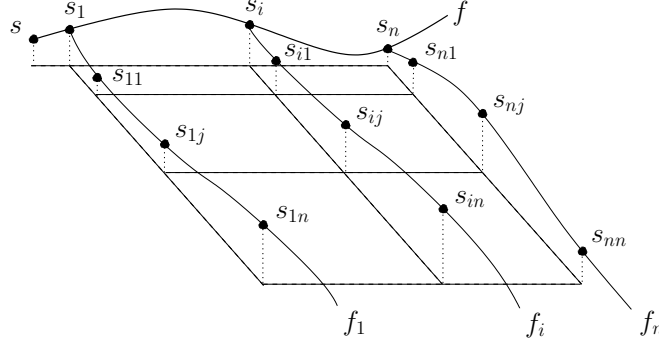


Figure 4.2: PE-shared secret s with 2-dimensional sharing structure.

We give PE-modules to share and reconstruct secrets with the PE secret-sharing scheme.

Module ComputeSharing. This parallel PE-module is a collection of protocols $[\text{PC}_{\text{ComputeSharing}}, \text{WFD}_{\text{ComputeSharing}}^m, \text{FL}_{\text{ComputeSharing}}^m]$. It computes a PE-secret sharing of the values $s^{(1)}, \dots, s^{(m)}$ in the player set $\mathcal{P}' = \{P_1, \dots, P_{n'}\}$. Thereby, for $k = 1, \dots, m$, $s^{(k)} = \sum_{l=1}^{n'} w_l x_l^{(k)}$ where $x_l^{(k)}$ is contributed by player $P_l \in \mathcal{P}'$ and $w_1, \dots, w_{n'}$ are constants known to all players.

The protocol $\text{PC}_{\text{ComputeSharing}}$ shares a single value and is executed m times in parallel before verifying all m sharings at once using $\text{WFD}_{\text{ComputeSharing}}^m$.

Protocol $\text{PC}_{\text{ComputeSharing}}$

In: Value x_i for player $P_i \in \mathcal{P}'$.

1. Each player $P_i \in \mathcal{P}'$ acts as dealer and pre-shares the value x_i with the sub-protocol $\text{PC}_{\text{PreShare}}$. Let $f_i(x)$ be the polynomial that P_i chooses thereby and x_{ij} P_j 's share of x_i
2. Each player $P_j \in \mathcal{P}'$ calculates

$$s_j = \sum_{l=1}^{n'} w_l x_{lj}.$$

3. Each player $P_i \in \mathcal{P}'$ pre-shares the value s_i with the sub-protocol $\text{PC}_{\text{PreShare}}$. Let $g_i(x)$ be the polynomial that P_i chooses thereby and s_{ij} P_j 's share of s_i .

Out: Shares s_i , $g_i(x)$ and $s_{1i}, \dots, s_{n'i}$ of the secret s for player $P_i \in \mathcal{P}'$. Auxiliary values are $f_i(x)$ and $x_{1i}, \dots, x_{n'i}$ for player $P_i \in \mathcal{P}'$.

During the weak fault detection, each player in \mathcal{P}' acts as verifier. The idea of the verification is to reconstruct a random linear combination over $k = 1, \dots, m$ of each input pre-sharings $x_1^{(k)}, \dots, x_{n'}^{(k)}$ as also of the second dimension pre-sharings of the $s^{(k)}$ and then, each verifier checks the consistency of the reconstructed values.

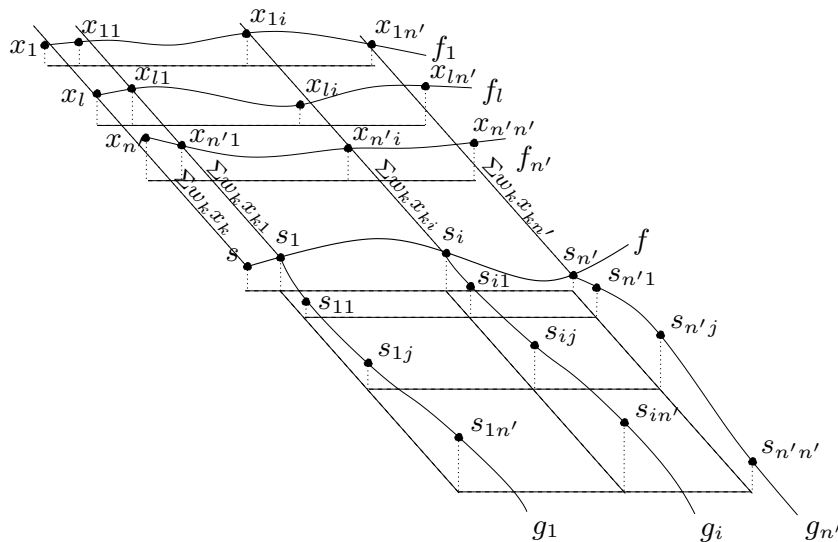


Figure 4.3: Illustration of the PE-sharing composition in $\text{PC}_{\text{ComputeSharing}}$.

The following protocol uses the auxiliary values that result during the execution of $\text{WFD}_{\text{PreShare}}^m$ for this purpose and we require that the players use the same random vectors for all verification in $\text{WFD}_{\text{PreShare}}^m$. This conflicts to the encapsulation idea of modules and is only done for efficiency reasons. Instead of using these auxiliary values, a new linear combination of the pre-shares could be reconstructed using module PreRec , where again new blinding shares were necessary.

Protocol $\text{WFD}_{\text{ComputeSharing}}^m$

In: Values $s_i^{(k)}, f_i^{(k)}(x), g_i^{(k)}(x), x_{1i}^{(k)}, \dots, x_{n'i}^{(k)}$ and $s_{1i}^{(k)}, \dots, s_{n'i}^{(k)}$ for player $P_i \in \mathcal{P}'$, $k = 1, \dots, m$.

1. All pre-sharings are verified, whereby each verifier P_v chooses the same random vector r_v in all executions of $\text{WFD}_{\text{PreShare}}^m$. For $i = 1, \dots, n'$, the following two steps are performed:
 - (a) $\text{WFD}_{\text{PreShare}}^m$ is executed on input $f_i^{(k)}, x_{i1}^{(k)}, \dots, x_{in'}^{(k)}$, $k = 1, \dots, m$ (P_i has been the dealer for all these sharings). We denote the output of player $P_v \in \mathcal{P}'$ as $b_i^{(v)}$ and his auxiliary values received from P_j as $z_{ij}^{(v)}$ ($1 \leq j \leq n'$).
 - (b) $\text{WFD}_{\text{PreShare}}^m$ is executed on input $g_i^{(k)}, s_{i1}^{(k)}, \dots, s_{in'}^{(k)}$, $k = 1, \dots, m$ (P_i has been the dealer for all these sharings). We denote the output of player $P_v \in \mathcal{P}'$ as $b_{n'+i}^{(v)}$ and his auxiliary values received from P_j as $\bar{z}_{ij}^{(v)}$ ($1 \leq j \leq n'$).

2. Each player $P_v \in \mathcal{P}'$ verifies that the sharings of the values $s^{(1)}, \dots, s^{(m)}$ are correct:

- (a) For $i = 1, \dots, n'$, P_v uses the values $\bar{z}_{i1}^{(v)}, \dots, \bar{z}_{in'}^{(v)}$ to compute $\bar{z}_i^{(v)}$ using Lagrange interpolation.
- (b) P_v verifies that

$$\bar{z}_i^{(v)} = \sum_{l=1}^{n'} w_l z_{li}^{(v)}$$

for each $i = 1, \dots, n'$.

If so, P_v sets $b_{2n'+1}^{(v)} = 0$ else $b_{2n'+1}^{(v)} = 1$.

3. Output for $P_i \in \mathcal{P}'$ is

$$b_i = \bigvee_{j=1}^{2n'+1} b_j^{(i)}.$$

Out: Auxiliary values are $z_{ij}^{(v)}, \bar{z}_i^{(v)}, b_i^{(v)}, b_{n'+i}^{(v)}, b_{2n'+1}^{(v)}, 1 \leq i, j \leq n'$, for player $P_v \in \mathcal{P}'$.

Protocol $\text{FL}_{\text{ComputeSharing}}^m$ Let $P_c \in \mathcal{P}'$ be the complaining player.

1. Each player $P_j \in \mathcal{P}'$ broadcasts the smallest index $k \in \{1, \dots, 2n'+1\}$ with $b_k^{(j)} = 1$ or \perp if there is no such index. Let i be the smallest broadcast index and P_v the player that broadcast i . (If all players broadcast \perp then $\mathcal{S} = \{P_c\}$.)
2. Depending on i the player do
 - $1 \leq i \leq n'$: Continue with $\text{FL}_{\text{PreShare}}^m$ for the sharings of $x_i^{(1)}, \dots, x_i^{(m)}$ and complaining player P_v .
 - $n' < i \leq 2n'$: Continue with $\text{FL}_{\text{PreShare}}^m$ for the sharings of $s_{i-n'}^{(1)}, \dots, s_{i-n'}^{(m)}$ and complaining player P_v .
 - $i = 2n'+1$: P_v selects and broadcasts some index j such that $\bar{z}_j^{(v)} \neq \sum_{l=1}^{n'} w_l z_{lj}^{(v)}$ and the output set is $\mathcal{S} = \{P_j, P_v\}$.

Lemma 6. *The module ComputeSharing fulfills Definition 4 for a parallel PE-module with $\delta = 1/q$ and $\gamma = 0$.*

Proof. Secrecy: The secrecy of $\text{PC}_{\text{ComputeSharing}}$ follows directly from the secrecy of the protocol $\text{PC}_{\text{PreShare}}$. The secrecy of $\text{WFD}_{\text{ComputeSharing}}^m$ relies on the secrecy of $\text{WFD}_{\text{PreShare}}^m$, no further values are sent.

Verifiability: If any pre-sharing has incorrect degree this will be detected with probability at least $1 - 1/q$ during the two invocations of $\text{WFD}_{\text{PreShare}}^m$. Due to the linearity of the pre-sharings this also holds for the polynomials that share $s^{(1)}, \dots, s^{(m)}$. What remains to prove is that each player gave the correct values as input to the pre-sharings for the second dimension. Assume some corrupted player P_j shares in Step 3 of the p -th execution $\text{PC}_{\text{ComputeSharing}}$ a value $s_j^{(p)}$ not computed as specified in Step 2. We show that every honest player will detect this during $\text{WFD}_{\text{ComputeSharing}}$ with probability at least $1 - 1/q = 1 - \delta$:

Let P_v be an honest verifier. In Step 2b, P_v checks whether $\bar{z}_j^{(v)}$ is the same as the sum $\sum_{l=1}^{n'} w_l z_{lj}^{(v)}$. The reconstruction values $z_{ij}^{(v)}$ and $\bar{z}_{ij}^{(v)}$, $1 \leq i, j \leq n'$ have been verified to be the correct linear combination of the players' shares during the execution of $\text{WFD}_{PreShare}^m$. Consider the random variable X which describes the difference

$$\begin{aligned} \bar{z}_j^{(v)} - \sum_{l=1}^{n'} w_l z_{lj}^{(v)} &\stackrel{(1)}{=} \sum_{k=1}^m r_v^{(k)} \tilde{s}_j^{(k)} - \sum_{l=1}^{n'} w_l \sum_{k=1}^m r_v^{(k)} x_{lj}^{(k)} \\ &\stackrel{(2)}{=} \sum_{k=1}^m r_v^{(k)} \tilde{s}_j^{(k)} - \sum_{k=1}^m r_v^{(k)} \sum_{l=1}^{n'} w_l x_{lj}^{(k)} \\ &\stackrel{(3)}{=} \sum_{k=1}^m r_v^{(k)} \tilde{s}_j^{(k)} - \sum_{k=1}^m r_v^{(k)} s_j^{(k)} \\ &\stackrel{(4)}{=} \sum_{k=1}^m r_v^{(k)} (\tilde{s}_j^{(k)} - s_j^{(k)}) \end{aligned}$$

where the values with tilde are the ones that P_j shared and those without the ones that he should have shared according to Step 3 of $\text{PC}_{ComputeSharing}$. In (3) we use the assumption that $\text{WFD}_{PreShare}^m$ checked all pre-sharings for correctness. Because by assumption $\tilde{s}_j^{(p)} - s_j^{(p)}$ is non-zero, X is uniformly distributed for independent and uniform $r_v^{(1)}, \dots, r_v^{(m)}$. Therefore, P_v will complain with probability $1 - 1/q$ if the linear combination $\bar{z}_j^{(v)}$ of the values that P_j shared and the ones that he should have shared are not the same.

Correctness: From inspecting the protocols, it follows that if all players are honest, there is always a correct secret sharing computed and the verification always succeeds.

Blame assigning: If in Step 1 of $\text{FL}_{ComputeSharing}^m$ the selected index i is not $2n' + 1$ then the invoked $\text{FL}_{PreShare}^m$ outputs a correct set \mathcal{S} . Otherwise, always $P_v \in \mathcal{S}$ and we need to show that an honest P_v always selects the index j of some corrupted player. We know that for an honest player P_j , it always holds that $\bar{z}_j^{(v)} = \sum_{l=1}^{n'} w_l z_{lj}^{(v)}$ because he computed and shared all $s_j^{(1)}, \dots, s_j^{(m)}$ correctly and the correct linear combination is reconstructed therefore as well. So, an honest P_v only selects indices j of corrupted players. \square

Module Reconstruct. This PE-module is a collection of protocols $[\text{PC}_{Reconstruct}, \text{WFD}_{Reconstruct}, \text{FL}_{Reconstruct}]$ that reconstruct a secret s , which is shared according to Definition 8, to all players in the set $\mathcal{R} \subseteq \mathcal{P}'$.

Protocol $\text{PC}_{Reconstruct}$

In: Polynomial $f_i(x)$ and $s_{ji} = f_j(i)$, $j = 1, \dots, n'$, for player $P_i \in \mathcal{P}'$.

1. Each player $P_i \in \mathcal{P}'$ sends $s_i = f_i(0)$ to all players in \mathcal{R} .
2. Each player $P_v \in \mathcal{R}$ interpolates a polynomial $p(x)$ from the received shares and outputs $s^{(v)} = p(0)$.

Out: Auxiliary values are the received values $s_1^{(v)}, \dots, s_{n'}^{(v)}$ for each player $P_v \in \mathcal{R}$.

Protocol WFD_{Reconstruct}

1. Each player $P_v \in \mathcal{R}$ verifies that the received shares $s_1^{(v)}, \dots, s_{n'}^{(v)}$ lie on a polynomial $p(x)$ of degree at most t' . If so, then P_v sets $b_v = 0$, otherwise $b_v = 1$. All other players $P_i \in \mathcal{P}' \setminus \mathcal{R}$ have fixed output $b_i = 0$.

Protocol FL_{Reconstruct} Let $P_v \in \mathcal{R}$ be the complaining player.

1. Each player $P_i \in \mathcal{P}'$ sends all his shares of s to P_v , i.e. the polynomial $f_i(x)$ and $s_{ji} = f_j(i)$ for $j = 1, \dots, n'$. P_v verifies that $f_i(0)$ is equal to $s_i^{(v)}$ (which he has received from P_i during $\text{PC}_{\text{Reconstruct}}$) for all $i = 1, \dots, n'$. If this does not hold for some f_i then P_v sets $f_i(x)$ to be the constant function $f_i(x) = s_i^{(v)}$.
2. P_v selects one pair of indices (i, j) , $1 \leq i, j \leq n'$ where the values received from P_i and P_j for the crosspoints of the sharing differ ($f_i(j) \neq s_{ij}$) and broadcasts (i, j) .
3. Player P_i and P_j now broadcast $f_i(j)$ and s_{ij} , respectively. If the two broadcasted values differ, then $\mathcal{S} = \{P_i, P_j\}$. Otherwise, P_v sets $k = i$ if what P_i broadcasted is different to what he had sent to P_v in Step 1 or $k = j$ otherwise. Then, P_v broadcasts k and $\mathcal{S} = \{P_k, P_v\}$.

Lemma 7. *The module Reconstruct fulfills Definition 2 for a PE-module with $\delta = \gamma = 0$.*

Proof. Secrecy: Only players in \mathcal{R} receive values during $\text{PC}_{\text{Reconstruct}}$ and $\text{WFD}_{\text{Reconstruct}}$ and these players are to know the secret.

Verifiability: At least $t' + 1$ honest players open the correct shares in Step 1 of $\text{PC}_{\text{Reconstruct}}$. These shares already uniquely define a degree- t' polynomial. Therefore, all players in \mathcal{R} will always notice if any corrupted player sends them an incorrect share.

Blame assigning: It is sufficient to show that $\text{FL}_{\text{Reconstruct}}$ never outputs a pair of honest players. We distinguish two cases of its output:

- $P_v \notin \mathcal{S}$: This happens only when players P_i and P_j broadcast different values in Step 3. This clearly does not happen if both are honest as the shares of the honest players are consistent by assumption.
- $P_v \in \mathcal{S}$: If P_v is corrupted, then the requirement is fulfilled. So, we assume that P_v is honest. There is always a pair of indices (i, j) with $f_i(j) \neq s_{ij}$ in Step 2 of $\text{FL}_{\text{Reconstruct}}$: P_v has recognized that $s_1^{(v)}, \dots, s_{n'}^{(v)}$ lie on a polynomial on degree $> t'$. Therefore, in Step 1 at least one of the functions f_i is not the correct one and not all shares $\{s_{ih} \mid P_h \text{ honest}\}$ lie on f_i as these interpolate the correct polynomial and there are at least $t' + 1$ of them. In Step 3, P_v always finds the index k of some corrupted player, as either P_i or P_j has broadcast something different than he has sent to P_v in Step 1.

□

Lemma 8. *If the module `Reconstruct` is used to publicly open the secret, i.e., for $\mathcal{R} = \mathcal{P}'$, then the module has secrecy-preserving fault localization.*

Proof. In this case, the purpose of the reconstruction is to open the secret to all players and therefore no module input has to be kept private. \square

4.4 Making PE-Sharings Robust

Later on, we will need to reconstruct PE-shared values in a robust way as we cannot tolerate that the adversary can gain information about the secret and provoke the fault detection of the reconstruction to fail which forces a repetition of the computation. Therefore, we use in this section the IC-signatures introduced in Chapter 3 and create protocols which add robustness to PE-sharings using these signatures. The resulting PE-module `MakeRobust` itself is non-robust. However, even if its computation fails, no information about the secret is revealed to the adversary. If its weak fault detection succeeds on the other hand, there is a protocol `RobustRec` which robustly reconstructs the sharing that results from `MakeRobust`.

The concept behind the protocols is to add signatures to all the players' shares. With that, for the shares that an honest and a corrupted player have in common, it is impossible for the corrupted player to open a wrong share during the reconstruction of the secret as this would require to forge the signature of the honest player.

Module `MakeRobust`. This PE-module is a collection of protocols [`PCMakeRobust`, `WFDMakeRobust`, `FLMakeRobust`] which adds signatures to the shares of a PE-sharing such that, on success, the secret can be robustly reconstructed.

Protocol `PCMakeRobust`

In: Each player in \mathcal{P}' has PE-shares of secret s where s_{ij} describes the share that players P_i and P_j have in common, $1 \leq i, j \leq n'$.

1. The players in \mathcal{P}' give each other pairwise IC-signatures for the PE-shares that they have in common: For each pair (P_i, P_j) , $1 \leq i, j \leq n'$, the player P_i creates an IC-signatures for the share s_{ij} using the protocol `PCICSign` on input s_{ij} with dealer P_i and prover P_j .

Out: Each player $P_i \in \mathcal{P}'$ has IC-signatures for his PE-shares $s_{1i}, \dots, s_{n'i}$ of secret s .

Protocol `WFDMakeRobust`

1. The players check whether all executions of `PCICSign` have been successful by executing `WFDICSign` for each created IC-signature. Furthermore, each player P_j checks for $i = 1, \dots, n'$ whether he received the signatures for the correct values s_{ij} from player P_i .
2. $P_v \in \mathcal{P}'$ outputs the bit $b_v = 1$ if he has detected any inconsistency in Step 1 or $b_v = 0$ else.

Protocol $\text{FL}_{\text{MakeRobust}}$ Let P_v be the complaining player.

1. If P_v has received a signature for an incorrect value s_{iv} from P_i , then he broadcasts the index i and $\mathcal{S} = \{P_i, P_v\}$.
2. Otherwise, P_v selects and broadcasts some pair of indices (i, j) where the execution of $\text{WFD}_{\text{ICSign}}$ to verify the IC-signature creation for s_{ij} showed him an inconsistency. Protocol $\text{FL}_{\text{ICSign}}$ is invoked for the IC-signature of s_{ij} with P_v as complaining player.

Lemma 9. *The module MakeRobust satisfies Definition 2 for a PE-module with the same (δ, γ) as the module ICSign and has secrecy-preserving fault localization.*

Proof. Secrecy: The secrecy of the three protocols follows directly from the secrecy of the used sub-module ICSign.

Verifiability: Each player verifies that he receives the signatures for the correct values. By the properties of sub-modules ICSign an invalid signature procedure is detected in $\text{WFD}_{\text{ICSign}}$, and therefore also in $\text{WFD}_{\text{MakeRobust}}$, with probability at least $1 - \delta$.

Completeness: Follows directly from the completeness of sub-module ICSign.

Blame assigning: Protocol $\text{FL}_{\text{ICDistVal}}$ is only invoked for instances where the complaining player P_v indeed detected an inconsistency (or claimed to do so). Therefore blame assigning follows from the properties of the sub-module ICSign. □

Protocol RobustRec Reconstructs a PE-sharing with IC-signatures (as it results from module MakeRobust) to player $P_r \in \mathcal{P}'$.

1. By executing the protocol ICRevealVal , each player $P_i \in \mathcal{P}'$ sends the shares $s_{1i}, \dots, s_{n'i}$ to the receiver P_r and reveals the IC-checkings for the shares to P_r .
2. For each $i = 1, \dots, n$, the receiver P_r does the following: P_r checks whether, out of the set $\{s_{i1}, \dots, s_{in'}\}$, he received at least $t' + 1$ shares with correct signatures and whether these lie on a polynomial of degree at most t' or not. If yes, he uses these values to reconstruct s_i . Let \mathcal{I} be the set of indices where he can do this.
3. P_r interpolates the polynomial $f(x)$ from the set $\{s_i \mid i \in \mathcal{I}\}$ using Lagrange interpolation and the reconstructed value is $s = f(0)$.

Lemma 10. *For an honest receiver P_r , the output of RobustRec is the secret s that is uniquely defined by the PE-shares of the honest players (except with probability at most n'^2/q).*

Proof. We assume that no corrupted player manages to forge an IC-signature. This assumption holds with probability at least $1 - n'^2/q$ as there are a total number of n'^2 signatures that are opened and the error probability of ICSigOpen is at most $1/q$. Then, for the index h of an honest player $P_h \in \mathcal{P}'$, out of set $\{s_{h1}, \dots, s_{hn'}\}$, P_r will only accept correct shares and at least $t' + 1$ of them,

as he receives so many from honest players. Therefore, P_r will reconstruct s_h for all indices h of honest players and again, there are at last $t' + 1$ of them, i.e., $|\mathcal{I}| \geq t' + 1$. What remains to show is that P_r never reconstructs some incorrect s_i for the index i of a corrupted player P_i . This cannot happen as the $t' + 1$ shares of the honest players in the set $\{s_{i_1}, \dots, s_{i_{t'+1}}\}$ already uniquely define a degree- t' polynomial p and therefore P_r does not use s_i if any corrupted player P_j sends him some s_{ij} with correct IC-signature that does not lie on the polynomial p . □

4.5 Complexity Analysis

From inspecting the protocols, it follows that the modules and protocols presented in this Chapter have following communication complexities where $\kappa = \log_2 q$ and we use that $\kappa \geq \log_2 n'$. The numbers for total do not include the n' bits broadcast during the fault detection when the modules are used as a section in a player-elimination protocol (see Section 2.2). All values are upper bounds.

	PreShare
PC	$n'\kappa$
WFD ^m	$mn'^2\kappa + 2n'^2\kappa$
FL ^m	$2(m+1)\kappa + n'\kappa + \text{BC}(\log_2 m) + 4\text{BC}(\kappa)$
total	$m(n'^2 + n')\kappa + 2(m+1)\kappa + (2n' + 1)n'\kappa + \text{BC}(\log_2 m) + 4\text{BC}(\kappa)$

Table 4.1: Communication complexities of module PreShare.

	ComputeSharing
PC	$2n'^2\kappa$
WFD ^m	$2mn'^2\kappa + 4n'^3\kappa$
FL ^m	$2(m+1)\kappa + n'\kappa + \text{BC}(\log_2 m) + n'\text{BC}(\kappa+1) + 4\text{BC}(\kappa)$
total	$4mn'^2\kappa + 2(m+1)\kappa + 4n'^3\kappa + n'\kappa + \text{BC}(\log_2 m) + n'\text{BC}(\kappa+1) + 4\text{BC}(\kappa)$

Table 4.2: Communication complexities of module ComputeSharing.

	PreRec
PC	$mn'\kappa + n'\kappa$
WFD	–
FL	$2m\kappa + n'\kappa + \text{BC}(\log m) + 4\text{BC}(\kappa)$
total	$m(n' + 2)\kappa + 2n'\kappa + \text{BC}(\log m) + 4\text{BC}(\kappa)$

Table 4.3: Communication complexities of module PreRec.

	Reconstruct ($\mathcal{R} = \mathcal{P}'$)
PC	$n'^2\kappa$
WFD	–
FL	$2n'^2\kappa + 5\text{BC}(\kappa)$
total	$3n'^2\kappa + 5\text{BC}(\kappa)$

Table 4.4: Communication complexities of module Reconstruct.

	MakeRobust
PC	$4n'^3\kappa$
WFD ^m	$6n'^3\kappa$
FL ^m	$6\text{BC}(\kappa) + 2\text{BC}(1)$
total	$10n'^3\kappa + 6\text{BC}(\kappa) + 2\text{BC}(1)$

Table 4.5: Communication complexities of module MakeRobust.

RobustRec	$2n'^2\kappa$
-----------	---------------

Table 4.6: Communication complexity of protocol RobustRec.

Chapter 5

Efficient MPC-Protocol for $t < n/2$

In this Chapter, we present a protocol for general secure multi-party computation in the secure-channels model with broadcast which makes use of the player-elimination framework. The protocol provides security against an active, adaptive threshold adversary which can corrupt up to $t < n/2$ out of the n players and is more efficient than the previously known protocols for the same model ([RB89, Bea91, CDD⁺99]).

The protocol allows a set \mathcal{P} of n players to securely compute an agreed function F (given as circuit) with M multiplication gates, R random gates and any number of addition gates. All computations are performed in a Galois field $K = GF(q)$ with $q > \max\{2^\kappa, n\}$ where κ is the security parameter. Thereby, each player can give a number of inputs to the function and receives the function output at the end of the protocol. We denote the total number of inputs (in field elements) with n_I and the number of outputs with n_O . The protocol satisfies the properties correctness and secrecy except with probability $2^{-\kappa + \mathcal{O}(\log n)}$. Correctness means, that the protocol always gives the correctly computed function output to all players. Secrecy means, that the adversary learns nothing about the inputs of the honest players (those who remain honest during the entire protocol execution), other than the function output.

5.1 Protocol Overview

Following MPC-protocol is a PE-protocol as specified in Definition 1. It consists of a single computation segment S_{Main} followed by the output segment. S_{Main} has as input the players' function inputs and as output a robust secret sharing of the function output. The output segment reconstructs the function output to all players after S_{Main} was successful. Having only one computation segment contradicts the considerations in Chapter 2 and a consequence of this choice is that the cost for the repetitions will be significant in the communication complexity. We will discuss this in Section 5.7.

The computation in S_{Main} proceeds in four phases: In the preparation phase, $M + R$ random triples $[a, b, c]$ with $c = ab$ are generated and shared among the players. In the second phase, the players create secret sharings of their function

inputs (using a robust protocol, such that also previously eliminated players can give input). Then, in the computation phase, the circuit is evaluated gate by gate using one random triple from the preparation phase for each multiplication [Bea91]. Finally, in the last phase, the output is made robust using signatures on the players' shares.

During the circuit evaluation itself, the players perform weak fault detection and at the end of the evaluation, it is verified that all computations were performed correctly, whereby an incorrect result is detected with probability at least $1 - 3 \cdot 2^{-\kappa}$. As the sharing of the output is made robust at the end of the computation segment, the output reconstruction is straight forward. Also players that have been eliminated by then can receive output.

In the following Sections we present the sub-modules that are necessary to compose the Main module which is given in Section 5.5. Finally, the output reconstruction protocol is specified in Section 5.6.

5.2 Generating Random Triples

The goal of the preparation phase is to generate $M + R$ shared random triples $[a^{(k)}, b^{(k)}, c^{(k)}] \in K^3$ with $c^{(k)} = a^{(k)}b^{(k)}$, $k = 1, \dots, M + R$. To do this, we use the following modules GenRandom, Multiply and GenTriple.

Module GenRandom. This parallel PE-module is a collection of protocols $[\text{PC}_{\text{GenRandom}}, \text{WFD}_{\text{GenRandom}}^m, \text{FL}_{\text{GenRandom}}^m]$. It creates a PE-sharing of m uniform random values.

The protocol $\text{PC}_{\text{GenRandom}}$ creates and pre-shares a single random value and is executed m times in parallel before verifying all m outputs at once using $\text{WFD}_{\text{GenRandom}}^m$.

Protocol $\text{PC}_{\text{GenRandom}}$

In: -

1. Each player $P_i \in \mathcal{P}'$ chooses a uniform random value $x_i \in K$.
2. Sub-protocol $\text{PC}_{\text{ComputeSharing}}$ is executed with $w_i = 1$ and input x_i , $i = 1, \dots, n'$. Denote the resulting shared value with r .

Out: Shares of r for each player in \mathcal{P}' .

Protocol $\text{WFD}_{\text{GenRandom}}^m$

In: Shares of $r^{(1)}, \dots, r^{(m)}$ for each player in \mathcal{P}' .

1. The players execute $\text{WFD}_{\text{ComputeSharing}}^m$ for $r^{(1)}, \dots, r^{(m)}$ and give the output thereof as output of this protocol.

Protocol $\text{FL}_{\text{GenRandom}}^m$

1. The players execute the sub-protocol $\text{FL}_{\text{ComputeSharing}}^m$.

Lemma 11. *The module GenRandom fulfills Definition 4 for a parallel PE-module with $\delta = 1/q$ and $\gamma = 0$.*

Proof. Secrecy: The secrecy of $\text{PC}_{\text{GenRandom}}$ and $\text{WFD}_{\text{GenRandom}}^m$ follows by the secrecy of the used sub-module ComputeSharing .

Verifiability: By invoking the protocol $\text{WFD}_{\text{ComputeSharing}}^m$, the players recognize an incorrectly computed sharing of r with probability at least $1 - 1/q$. If r is computed as specified

$$r = \sum_{i=1}^{n'} x_i$$

then r is certainly uniform random as there is at least one honest player $P_h \in \mathcal{P}'$ which chooses his contribution x_h uniform at random.

Blame assigning: Follows directly from the properties of the sub-module. \square

The multiplication in the module Multiply presented below is done as in [GRR98]: Each player $P_i \in \mathcal{P}'$ multiplies his shares of a and b and pre-shares the product $x_i = a_i b_i$ with the polynomial $h_i(x)$. The values $x_1, \dots, x_{n'}$ lie on the polynomial $f(x)g(x)$ which is of degree $2t'$ and therefore the x_i -s are not a proper pre-sharing as is. But as $2t' < n'$, the product of a and b can still be uniquely interpolated from $x_1, \dots, x_{n'}$ using the Lagrange interpolation polynomial:

$$L(0) = \sum_{i=0}^{n'} x_i l_i(0) = ab = c$$

with the Lagrange basis polynomials

$$l_i(x) = \prod_{\substack{k=1 \\ k \neq i}}^{n'} \frac{k-x}{k-i}.$$

For a fixed player set \mathcal{P}' , the values $l_i(0)$ are constant and the product is a linear combination of $x_1, \dots, x_{n'}$. Using this observation and the fact that our secret sharing allows to calculate linear combination of shares without communication, the players calculate a pre-sharing of the product as linear combination of the pre-sharings of $x_1, \dots, x_{n'}$: The polynomial

$$h(x) = \sum_{i=0}^{n'} h_i(x) l_i(0)$$

is of degree maximal t' (as all h_i are of degree maximal t') and shares the product, i.e., $h(0) = ab$, because each h_i shares x_i and following the consideration above.

The players have to verify that each player $P_i \in \mathcal{P}'$ has properly shared the value x_i as the product of the two previously shared secrets a_i and b_i . For that, we use a corrected version of the protocol presented in [CDD⁺99] (which showed to be vulnerable to an adaptive adversary)¹ and adapt it to our needs:

¹In Step 2 in Section 6.2, D chooses random β and shares β and $b\beta$ with a *one-dimensional* sharing without any checks for consistency of the shares. This allows a corrupted D to give inconsistent shares of β and $b\beta$ to the players and then to corrupt some of the players *after* the challenge r is exposed (Step 3) such that the shares of the remaining players are consistent. An attack which is described in Section 3 of the very same paper.

The resulting protocol is part of $\text{WFD}_{\text{Multiply}}^m$ and formulated in the view of a verifier. It makes use of player-elimination and was parallelized: Instead of verifying one triple, it verifies m triples at once.

Module Multiply. This parallel PE-module is a collection of protocols $[\text{PC}_{\text{Multiply}}, \text{WFD}_{\text{Multiply}}^m, \text{FL}_{\text{Multiply}}^m]$. It multiplies m times two given PE-shared values $a^{(k)}$ and $b^{(k)}$ and creates a PE-sharing of the product $c^{(k)} = a^{(k)}b^{(k)}$, $k = 1, \dots, m$.

The protocol $\text{PC}_{\text{Multiply}}$ multiplies two shared values and is executed m times in parallel before verifying all m products at once using $\text{WFD}_{\text{Multiply}}^m$.

Protocol $\text{PC}_{\text{Multiply}}$

In: Shares a_i and b_i of the secrets a and b for each player $P_i \in \mathcal{P}'$.

1. The players execute the sub-protocol $\text{PC}_{\text{ComputeSharing}}$ with input $x_i = a_i b_i$ for player $P_i \in \mathcal{P}'$ and weights $w_i = l_i(0)$. Denote the resulting PE-shared secret with c .
2. Each player $P_i \in \mathcal{P}'$ selects a random value \bar{a}_i and pre-shares the values \bar{a}_i and $\bar{x}_i = \bar{a}_i b_i$ by executing the sub-protocol $\text{PC}_{\text{PreShare}}$.

Out: Shares c_i of the secret c . Auxiliary values are the pre-shares of x_j , \bar{a}_j and \bar{x}_j , $j = 1, \dots, n'$, for each player $P_i \in \mathcal{P}'$.

Protocol $\text{WFD}_{\text{Multiply}}^m$

In: Values $a_i^{(k)}$, $b_i^{(k)}$, $x_i^{(k)}$, $c_i^{(k)}$, $\bar{a}_i^{(k)}$ and $\bar{x}_i^{(k)}$, $k = 1, \dots, m$, for each player $P_i \in \mathcal{P}'$.

1. The players in \mathcal{P}' execute, for each $i = 1, \dots, n'$, twice the sub-protocol $\text{WFD}_{\text{PreShare}}^m$ to verify the correctness of the $\bar{a}_i^{(1)}, \dots, \bar{a}_i^{(m)}$ - and $\bar{x}_i^{(1)}, \dots, \bar{x}_i^{(m)}$ -pre-sharings. Let $z_v^{(1,i)}$ and $z_v^{(2,i)}$ be P_v 's output bit thereof.
2. The players in \mathcal{P}' jointly generate and open a random value r using the sub-protocols $\text{PC}_{\text{GenRandom}}$, $\text{WFD}_{\text{GenRandom}}^1$, $\text{PC}_{\text{Reconstruct}}$ and $\text{WFD}_{\text{Reconstruct}}$. Let $z_v^{(3,1)}$ and $z_v^{(3,2)}$ be P_v 's output of $\text{WFD}_{\text{GenRandom}}^1$ and $\text{WFD}_{\text{Reconstruct}}$.
3. In the following, we will consider (for $1 \leq k \leq m$, $1 \leq i \leq n'$) the linear combinations of pre-shared values

$$\begin{aligned} p_i^{(k)} &= r a_i^{(k)} + \bar{a}_i^{(k)} \quad \text{and} \\ q_i^{(k)} &= p_i^{(k)} b_i^{(k)} - r x_i^{(k)} - \bar{x}_i^{(k)}. \end{aligned}$$

Each player $P_i \in \mathcal{P}$ computes and sends $p_i^{(1)}, \dots, p_i^{(m)}$ to all players in \mathcal{P}' .

4. Each player $P_v \in \mathcal{P}'$ selects a random vector $\mathbf{w}_v = [w_v^{(1)}, \dots, w_v^{(m)}] \in (K \setminus \{0\})^m$. The players reconstruct for $i = 1, \dots, n'$ the following random

linear combinations of pre-shared values to each player $P_v \in \mathcal{P}'$ using the sub-protocol PC_{PreRec} (with coefficient vector \mathbf{w}_v):

$$p_i^{(\Sigma)} = \sum_{k=1}^m w_v^{(k)} p_i^{(k)} \quad \text{and} \quad q_i^{(\Sigma)} = \sum_{k=1}^m w_v^{(k)} q_i^{(k)}$$

5. Each player $P_v \in \mathcal{P}'$ verifies for $i = 1, \dots, n'$ the reconstruction of $p_i^{(\Sigma)}$ and $q_i^{(\Sigma)}$ with the sub-protocol WFD_{PreRec} . Let $z_v^{(4,i)}$ and $z_v^{(5,i)}$ be P_v 's output thereof.
6. Each player $P_v \in \mathcal{P}'$ sets for $i = 1, \dots, n'$ the bit $z_v^{(6,i)} = 0$ if $p_i^{(\Sigma)}$ is the same as the linear combination of the values $p_i^{(k)}$ which he received from P_i in Step 3, and the bit $z_v^{(7,i)} = 0$ if $q_i^{(\Sigma)} = 0$.
7. Output for $P_v \in \mathcal{P}'$ is

$$b_v = \bigvee_{i,j} z_v^{(j,i)}$$

Out: Auxiliary values are $z_v^{(j,i)} \forall i, j$ for player $P_v \in \mathcal{P}$.

Protocol $\text{FL}_{Multiply}^m$ Let $P_c \in \mathcal{P}'$ be the complaining player.

1. Each player $P_j \in \mathcal{P}'$ selects the smallest index $k \in \{1, \dots, 7\}$ where there is some index l such that $z_j^{(k,l)} = 1$ and broadcasts (k, l) or \perp if there is no such index. Let (k', l') be the index with smallest k' among the broadcasted indices (and if there is more than one with the same k' , the one among these with smallest l') and P_v the player that broadcasted it. (If all players broadcast \perp then $\mathcal{S} = \{P_c\}$.)
2. Depending on (k', l') , the players continue as following (with P_v as complaining player)

$$\begin{aligned} k' = 1: & \quad \text{FL}_{PreShare}^m \text{ for } \bar{a}_{l'} \\ k' = 2: & \quad \text{FL}_{PreShare}^m \text{ for } \bar{x}_{l'} \\ k' = 3, l' = 1: & \quad \text{FL}_{GenRandom}^1 \text{ for } r \\ k' = 3, l' = 2: & \quad \text{FL}_{Reconstruct} \text{ for } r \\ k' = 4: & \quad \text{FL}_{PreRec}^m \text{ for } p_{l'}^{(\Sigma)} \\ k' = 5: & \quad \text{FL}_{PreRec}^m \text{ for } q_{l'}^{(\Sigma)} \\ k' \in \{6, 7\}: & \quad \mathcal{S} = \{P_{l'}, P_v\} \end{aligned}$$

Lemma 12. *The module Multiply fulfills Definition 4 for a parallel PE-module with $\delta = 3/(q-1)$ and $\gamma = 0$.*

Proof. Secrecy: During $\text{PC}_{Multiply}$, the players receive only shares of secret-shared values which give no information about the secrets as long as the shares of not more than t' players in \mathcal{P}' are pooled. In protocol $\text{WFD}_{Multiply}^m$ the players reconstruct the values $p_i^{(1)}, \dots, p_i^{(m)}$ and $q_i^{(1)}, \dots, q_i^{(m)}$ (or rather linear combinations thereof). This does not violate the secrecy of the secrets $a^{(k)}$, $b^{(k)}$ or $c^{(k)}$ ($k = 1, \dots, m$) because in $p_i^{(k)}$ the value $a_i^{(k)}$ is blinded by the addition of the random $\bar{a}_i^{(k)}$ and the players already know in advance that $q_i^{(k)}$ is zero. (The

value $q_i^{(k)}$ is only non-zero when the player P_i shares incorrect values – and is corrupted therefore. In this case however, the adversary learned $q_i^{(k)}$ already by corrupting P_i . Same argument for a non-random $\bar{a}_i^{(k)}$.)

Verifiability: As previously explained, a sharing of the product $c = ab$ can be computed as linear combination of the product shares $a_i b_i$. This is exactly what Step 1 of $\text{PC}_{\text{Multiply}}$ does by invoking $\text{PC}_{\text{ComputeSharing}}$ with the specified weights. What remains to be verified is that each player $P_i \in \mathcal{P}'$ indeed gives the product of his shares a_i and b_i as input x_i . P_i proofs this to each player $P_v \in \mathcal{P}'$ as explained in the following:

The idea of the proof is to reconstruct $a_i b_i - x_i$ and to verify that this is zero (which is equivalent to the check that $a_i b_i = x_i$). Of course, this cannot be reconstructed directly as this would require to multiply two pre-shared values a_i and b_i in a MPC. Instead, the players reconstruct a random multiple of the difference and write it as

$$r(a_i b_i - x_i) = (r a_i + \bar{a}_i - \bar{a}_i) b_i - r x_i = (r a_i + \bar{a}_i) b_i - r x_i - \bar{a}_i b_i$$

for a random blinding value \bar{a}_i chosen by P_i and a jointly generated random value r . The term $\bar{a}_i b_i$ can be computed and pre-shared by P_i which we denote by \bar{x}_i . The term $r a_i + \bar{a}_i$ corresponds to p_i and can be reconstructed to all players without violating the secrecy. With that, the term $q_i = p_i b_i - r x_i - \bar{x}_i$ is a linear combination of pre-shared values and can therefore be computed and reconstructed easily. Now the following claim holds: If $a_i b_i \neq x_i$ then for each \bar{x}_i there is exactly one value r such that $q_i = 0$. This is true, because

$$(r a_i + \bar{a}_i) b_i - r x_i - \bar{x}_i = 0 \quad \Leftrightarrow \quad r = (\bar{x}_i - \bar{a}_i b_i) / (a_i b_i - x_i)$$

On the other hand, if $a_i b_i = x_i$ then for $\bar{x}_i = \bar{a}_i b_i$ the value q_i is always zero for any r .

This means that the probability to overlook an incorrect x_i is $1 - 1/|K|$. (There could be more than one player trying to share an incorrect x_i . However, it is sufficient, then, to detect only one of them as this already causes the segment to be repeated.) It is important that the value r is opened to the players only after the sharings of $\bar{a}_i^{(1)}, \dots, \bar{a}_i^{(m)}$ and $\bar{x}_i^{(1)}, \dots, \bar{x}_i^{(m)}$ have been verified. The verification makes sure that the shares of the players honest at that time are consistent and therefore define a unique secret. Otherwise, an adaptive adversary could try to corrupt players dependent on r such that the shares of the remaining honest players define different values for different r s.

In $\text{WFD}_{\text{Multiply}}^m$, P_i has to give m such proofs to each player. It would be more communication needed than we want if the players reconstructed each value $q_i^{(1)}, \dots, q_i^{(m)}$. Instead, only the random linear combination

$$q_i^{(\Sigma)} = \sum_{k=1}^m w_v^{(k)} q_i^{(k)}$$

is reconstructed towards each player P_v where the $w_v^{(k)}$ are chosen by P_v . If all $q_i^{(1)}, \dots, q_i^{(m)}$ are zero, clearly the sum is zero as well. On the other hand, if for some $k \in \{1, \dots, m\}$, $q_i^{(k)}$ is non-zero, then the linear combination is zero with probability at most $1/(|K| - 1)$. In order to compute a pre-sharing of

$q_i^{(\Sigma)}$ however, all players have to know $p_i^{(1)}, \dots, p_i^{(m)}$. Again, we do not want to reconstruct all these values. Instead, P_i sends these values to all players and the random linear combination $p_i^{(\Sigma)}$ is reconstructed to verify that P_i sent the correct values. Therefore, if P_v gets for one or more indices $k \in \{1, \dots, m\}$ an incorrect value $p_i^{(k)}$ he will overlook this with probability at most $1/(|K| - 1)$ for each $i = 1, \dots, n'$.

So in total, the probability that P_v accepts an incorrect proof of P_i that the shared values $x_i^{(1)}, \dots, x_i^{(m)}$ are the products $a_i^{(1)}b_i^{(1)}, \dots, a_i^{(m)}b_i^{(m)}$ is at most $3/(|K| - 1) \leq \delta$ as this can happen when the corrupt P_i either gets the corresponding $\bar{x}_i^{(\Sigma)}$ right or the verification of the $p_i^{(1)}, \dots, p_i^{(m)}$ or $q_i^{(1)}, \dots, q_i^{(m)}$ overlooks an incorrect value.

Blame assigning: For index $1 \leq k' \leq 5$ the property follows from the properties of the sub-modules. Otherwise, player P_v is always in \mathcal{S} and it is sufficient to show that an honest P_v always broadcasted the index l' of some corrupted player. We assume that the weak fault detection of the used sub-modules has not missed any deviation from the protocol and therefore all sharings are consistent which holds with probability at least $1 - \delta$. If $k' = 5$ this means that for $j = l'$ the reconstructed value $p_j^{(\Sigma)}$ does not match the linear combination of the $p_j^{(1)}, \dots, p_j^{(m)}$ which P_v received from P_j . If $k' = 6$ this means that for $j = l'$ the reconstructed value $q_j^{(\Sigma)}$ is non-zero. Both cannot happen when P_j is honest as he is the dealer of this values (or rather of all the pre-shares which are used to compute them). \square

Note that the protocol $\text{FL}_{\text{Multiply}}^m$ does not preserve the secrecy of the module inputs. This is the reason why it cannot be used to multiply values during the circuit evaluation. We only use it to multiply random values which we discard when $\text{WFD}_{\text{Multiply}}^m$ detects an inconsistency and therefore the execution of $\text{FL}_{\text{Multiply}}^m$ may be necessary.

The communication complexity of the module Multiply can be reduced by requiring that in Step 1 of $\text{WFD}_{\text{Multiply}}^m$ for all n' execution of $\text{WFD}_{\text{PreShare}}^m$, each verifier always chooses the same random vector for the verification. This does not tamper the security of $\text{WFD}_{\text{Multiply}}^m$ as the random vector is still independent of the computations in $\text{PC}_{\text{Multiply}}$. Furthermore, note that in Step 4 of $\text{WFD}_{\text{Multiply}}^m$, the coefficient vector w is n' times the same and therefore must be only sent once in the sub-protocol $\text{PC}_{\text{PreRec}}$.

Module GenTriple. This parallel PE-module is a collection of protocols $[\text{PC}_{\text{GenTriple}}, \text{WFD}_{\text{GenTriple}}^m, \text{FL}_{\text{GenTriple}}^m]$. It creates a PE-sharing of m random triples $[a^{(k)}, b^{(k)}, c^{(k)}] \in K^3$ where $c^{(k)} = a^{(k)}b^{(k)}$, $k = 1, \dots, m$.

The protocol $\text{PC}_{\text{GenTriple}}$ creates and pre-shares a single random triple and is executed m times in parallel before verifying all m outputs at once using $\text{WFD}_{\text{GenTriple}}^m$.

Protocol $\text{PC}_{\text{GenTriple}}$ *In:* -

1. A random value a is generated and shared by executing the sub-protocol $\text{PC}_{\text{GenRandom}}$.
2. A random value b is generated and shared by executing the sub-protocol $\text{PC}_{\text{GenRandom}}$.
3. The shared values a and b are multiplied to get a sharing of $c = ab$ by executing the sub-protocol $\text{PC}_{\text{Multiply}}$.

Out: Shares for a , b and c for each player in \mathcal{P}' .**Protocol** $\text{WFD}_{\text{GenTriple}}^m$ *In:* Shares for $a^{(k)}$, $b^{(k)}$ and $c^{(k)}$, $k = 1, \dots, m$ for each player in \mathcal{P}' .

1. The sub-protocol $\text{WFD}_{\text{GenRandom}}^m$ is executed for both $a^{(1)}, \dots, a^{(m)}$ and $b^{(1)}, \dots, b^{(m)}$. Denote P_i 's output with $z_i^{(1)}$ and $z_i^{(2)}$.
2. The sub-protocol $\text{WFD}_{\text{Multiply}}^m$ is executed to verify $c^{(1)}, \dots, c^{(m)}$. Denote P_i 's output with $z_i^{(3)}$.
3. Player $P_i \in \mathcal{P}'$ outputs $b_i = z_i^{(1)} \vee z_i^{(2)} \vee z_i^{(3)}$.

Out: Auxiliary values are $z_i^{(1)}$, $z_i^{(2)}$ and $z_i^{(3)}$ for player $P_i \in \mathcal{P}'$.**Protocol** $\text{FL}_{\text{GenTriple}}^m$ Let $P_c \in \mathcal{P}'$ be the complaining player.

1. Each player $P_j \in \mathcal{P}'$ broadcasts the smallest index $k \in \{1, 2, 3\}$ with $z_j^{(k)} = 1$ or \perp if there is no such index. Let i be the smallest broadcast index and P_v the player that broadcast i . (If all players broadcast \perp then $\mathcal{S} = \{P_c\}$.)
2. Depending on i the players continue with $\text{FL}_{\text{GenRandom}}$ for a ($i = 1$) or for b ($i = 2$) or with $\text{FL}_{\text{Multiply}}$ ($i = 3$) with P_v as complaining player.

Lemma 13. *The module GenTriple fulfills Definition 4 for a parallel PE-module with $\delta = 3/(q-1)$ and $\gamma = 0$.**Proof. Secrecy:* In $\text{PC}_{\text{GenTriple}}$, the secrecy of $a^{(1)}, \dots, a^{(m)}$ and $b^{(1)}, \dots, b^{(m)}$ is ensured by $\text{PC}_{\text{GenRandom}}$ and during the multiplication by $\text{PC}_{\text{Multiply}}$ the adversary learns nothing about the random values. $\text{WFD}_{\text{GenTriple}}^m$ preserves secrecy by the properties of the used sub-modules, no further values are sent.*Verifiability:* By invoking the corresponding weak fault detection protocols of the used sub-modules GenRandom and Multiply, the players recognize an incorrect $a^{(k)}$, $b^{(k)}$ or $c^{(k)}$ with probability at least $1 - \delta$ each.*Blame assigning:* For $i = 1$ and $i = 2$ the correctness of the fault localization follows directly from the correctness of $\text{FL}_{\text{GenRandom}}^m$. Assuming $\text{WFD}_{\text{GenRandom}}^m$ did not overlook any incorrect output (which happens with probability at least $1 - \delta$) this also holds for $i = 3$ and $\text{FL}_{\text{Multiply}}^m$. \square

5.3 Giving Input

The following robust protocol for sharing the inputs of the function works as in the input stage of [HMP00]. [HMP00] is designed for $t < n/3$ but the same arguments for the correctness of the input sharing protocol also hold for $t < n/2$.

Protocol ShareInput Allows a player $D \in \mathcal{P}$ to share a secret among the players in \mathcal{P}' .

In: Secret s for dealer D .

1. D selects at random a polynomial $p(x, y)$ of degree at most t' in both variables, where $p(0, 0) = s$ and sends the polynomials $f_i(x) = p(x, i)$ and $\tilde{f}_i(y) = p(i, y)$ to player P_i , $i = 1, \dots, n'$.
2. Each pair of players (P_i, P_j) $1 \leq i, j \leq n'$ checks whether $p(i, j) = p(j, i)$. For this, P_i sends $f_i(j)$ to P_j and P_j checks that the received value is equal to $\tilde{f}_j(i)$.
3. Each player in \mathcal{P}' broadcasts a bit vector where the j -th bit indicates whether or not the player has observed an inconsistency with player $P_j \in \mathcal{P}'$. The dealer has to answer the complaints by broadcasting the corresponding correct values.
4. If a player $P_i \in \mathcal{P}'$ observes that what the dealer broadcasts contradicts to what he sent him in Step 2 then he broadcasts an accusation. The dealer has to answer this by broadcasting P_i 's share, i.e. the polynomials $f_i(x)$ and $\tilde{f}_i(x)$. The published polynomials can cause new inconsistencies with the values held by some other players, who react again with accusations, and so on. If the dealer broadcasts inconsistent values or more than t' players have accused, then the dealer is disqualified and a fixed default sharing is taken.

Note that the resulting shares define a PE-secret sharing (Definition 8) with shares $f_j(x)$ and $s_{ij} = \tilde{f}_j(i)$, $i = 1, \dots, n'$ for player $P_j \in \mathcal{P}'$.

5.4 Circuit Evaluation

Taking advantage of the linearity of the used sharing, additions and multiplications with constants are evaluated without any communication. Multiplications of two variables are evaluated according to [Bea91]: To multiply the shared factors x and y , a multiplication-triple (a, b, c) with $c = ab$ is used and the differences $d_x := x - a$ and $d_y := y - b$ are opened to all players. Then, using the observation that the product xy can be written as

$$\begin{aligned} xy &= ((x - a) + a)((y - b) + b) \\ &= (x - a)(y - b) + (x - a)b + (y - b)a + c \\ &= d_x d_y + d_x b + d_y a + c \end{aligned}$$

the players can calculate shares of the product xy as a linear combination of a, b and c .

Module EvalMult. This PE-module is a collection of protocols $[\text{PC}_{\text{EvalMult}}, \text{WFD}_{\text{EvalMult}}, \text{FL}_{\text{EvalMult}}]$. It computes a PE-sharing of two shared values x and y using a shared random triple $(a, b, c = ab)$.

Protocol $\text{PC}_{\text{EvalMult}}$

In: Shares of secrets x, y, a, b and c for each player in \mathcal{P}' .

1. The players reconstruct $d_x = x - a$ and $d_y = y - b$ using the sub-protocol $\text{PC}_{\text{Reconstruct}}$.
2. Each player in \mathcal{P}' computes a share of the product $z = xy$ as linear combination of his shares of a, b and c .

$$z = d_x d_y + d_x b + d_y a + c$$

Out: Share of z for each player in \mathcal{P}' .

Protocol $\text{WFD}_{\text{EvalMult}}$

1. The players verify the reconstruction of the differences d_x and d_y with the sub-protocol $\text{WFD}_{\text{Reconstruct}}$. Let $b_i^{(x)}$ and $b_i^{(y)}$ be P_i 's output thereof.
2. Player $P_i \in \mathcal{P}'$ outputs $b_i = b_i^{(x)} \vee b_i^{(y)}$.

Protocol $\text{FL}_{\text{EvalMult}}$ Let $P_v \in \mathcal{P}'$ be the complaining player.

1. If $b_v^{(x)} = 1$ then P_v broadcasts $i = 1$ else $i = 2$. Depending on that, the players continue with sub-protocol $\text{FL}_{\text{Reconstruct}}$ for the reconstruction of either d_x ($i = 1$) or d_y ($i = 2$).

Lemma 14. *The module EvalMult fulfills Definition 2 for a PE-module with $\delta = \gamma = 0$ and has secrecy-preserving fault localization.*

Proof. Secrecy: As a and b are random values, the reconstruction of d_x and d_y gives no information about x and y to the adversary. The secrecy of the used sub-protocols is given by the properties of the module Reconstruct which has secrecy-preserving fault localization.

Verifiability: If a and b are random values, $c = ab$ and the reconstruction of d_x and d_y is correct then the output is clearly correct as well. The properties of (a, b, c) are precondition and if the reconstruction of d_x or d_y is incorrect, this is detected with probability 1 during $\text{WFD}_{\text{Reconstruct}}$.

Blame assigning: Follows directly from the properties of the sub-module Reconstruct.

□

5.5 Main Segment

The segment S_{Main} is defined by the PE-module Main and empty protocol Trans (see Section 2.2).

Module Main. This PE-module is a collection of protocols $[\text{PC}_{Main}, \text{WFD}_{Main}, \text{FL}_{Main}]$. It computes a secret sharing of the output of function F on input x_1, \dots, x_{n_I} in secure multi-party computation.

Protocol PC_{Main}

In: Inputs $x_i^{(1)}, \dots, x_i^{(l_i)}$ (where $\sum_{i=1}^n l_i = n_I$) for player $P_i \in \mathcal{P}$.

1. The players in \mathcal{P}' create sharings of random triples $[a^{(k)}, b^{(k)}, c^{(k)}]$ with $c^{(k)} = a^{(k)}b^{(k)}$, $k = 1, \dots, M + R$ using the sub-protocol $\text{PC}_{GenTriple}$.
2. Each player $P_i \in \mathcal{P}'$ sets $b_i = 0$. When, during the execution of this protocol, P_i gets informed about an inconsistency for the first time he memorizes the index $v_i = j$ of the player P_j who informed him, sets $b_i = 1$ and z_i to the index of the current computation. If he gets informed by more than one player in the same round, he chooses the one with smallest player index. Afterwards, instead of further computing the values according to the protocol, P_i sends random values whenever the protocol tells him to send some value to another player.
3. Protocol $\text{WFD}_{GenTriple}^{M+R}$ is executed to check the correctness of the triples generated in Step 1. Each player in \mathcal{P}' sends a bit to all players in \mathcal{P}' to inform them whether he detected an inconsistency or not.
4. Each player $P_i \in \mathcal{P}$ shares his inputs $x_i^{(1)}, \dots, x_i^{(l_i)}$ by executing sub-protocol ShareInput for each input.
5. The circuit is evaluated gate by gate. Additions and multiplications with constants are performed without communication using the linearity of the PE-secret sharing. Multiplications are computed with the sub-protocol $\text{PC}_{EvalMult}$ which takes each time – beside the two factors – an unused random triple from Step 1 as input. To evaluate random gates, a triple $[a, b, c]$ from Step 1 is taken as well and the sharing of a is used as random value, b and c are discarded.
6. After each multiplication, the players perform weak fault detection: Protocol $\text{WFD}_{EvalMult}$ is executed and each player $P_i \in \mathcal{P}'$ sends his output bit to all players in \mathcal{P}' .
7. After all operations have been evaluated, the sharings of the output is made robust by executing protocol $\text{PC}_{MakeRobust}$ for each sharing.
8. Protocol $\text{WFD}_{MakeRobust}$ is executed and each player sends his output bit to all players in \mathcal{P}' .

Out: Sharings of the output. Auxiliary values are b_i, z_i and, possibly, memorized index v_i for each player $P_i \in \mathcal{P}'$.

Protocol WFD_{Main}

1. Each player $P_i \in \mathcal{P}'$ outputs the bit b_i computed during PC_{Main} .

Protocol FL_{Main} Let $P_c \in \mathcal{P}'$ be the complaining player.

1. Each player $P_j \in \mathcal{P}'$ broadcasts \perp if b_j is zero, or (z_j, v_j) otherwise. If all players broadcast \perp then $\mathcal{S} = \{P_c\}$.
2. If all players broadcast the same pair of indices (z, v) then the players continue with the corresponding FL-protocol of round z with P_v as complaining player. Otherwise, let (i, j) be the pair of indices where i is the smallest round broadcast and j the corresponding player index (if multiple players broadcasted the same i with different player indices then let j be the smallest index). We distinguish two cases:
 - (a) A majority of the players broadcasted (i, j) . Then, the players select the smallest player index k among the players who broadcasted something different and $\mathcal{S} = \{P_j, P_k\}$.
 - (b) Otherwise, the players select the smallest player index k' among the players who broadcasted (i, j) and $\mathcal{S} = \{P_j, P_{k'}\}$.

Lemma 15. *The module Main satisfies Definition 2 for a PE-module with $\delta = \gamma = 3/(q-1)$ and has secrecy-preserving fault localization.*

Proof. Secrecy: The generation of the random triples itself does not threaten the secrecy as this is independent of the players' inputs. If, however, an incorrect triple gets generated, i.e., one that is either not uniformly random or one where the product is wrong, this can cause to leak information to the adversary during the circuit evaluation. As specified in module GenTriple, the protocol $\text{WFD}_{GenTriple}^{M+R}$ detects any incorrect triple with probability at least $1 - 3/(q-1)$. If all triples are correct, then the secrecy is preserved during the remainder of PC_{Main} : The input stage is unconditionally secure. Under the precondition of correct triples, the function evaluation during $\text{PC}_{EvalMult}$ does not give any information about the function inputs to the adversary as long as the random values remain secret. Each reconstruction is verified by $\text{WFD}_{EvalMult}$ which detects an incorrect result with probability 1. If so, the honest players continue with dummy values which prevents that the secrecy is violated in the further execution of the protocol. Module MakeRobust always preserves secrecy. Summarizing, the secrecy of PC_{Main} is only violated when one or more invalid triple is not detected in Step 3, which happens at most with probability γ .

Protocol WFD_{Main} trivially fulfills secrecy as no values are sent. Let us consider FL_{Main} : Fault-localization protocols of the sub-modules GenTriple, EvalMult and MakeRobust are only invoked when all players agree that they were informed about the same inconsistency during the computation. This means that from that point in the computation all honest players continued with dummy values. Therefore, as the fault-localization of the sub-modules EvalMult and MakeRobust are secrecy-preserving and the random triples were not used in the case where $\text{FL}_{GenTriple}^{M+R}$ is invoked, the secrecy is preserved during FL_{Main} .

Verifiability: Each honest player detects an incorrect triple of Step 1 with probability at least $3/(q-1) = \delta$ during $\text{WFD}_{GenTriple}^{M+R}$. Assuming the triples are correct, each player detects an incorrect function evaluation with probability 1 as $\text{WFD}_{EvalMult}$ always detects an incorrect computation step. Module MakeRobust detects with probability at least $1 - 1/(q-1) > 1 - \delta$ if any player tries to give an invalid signature.

Correctness: Follows from the correctness of the used sub-modules GenTriple, EvalMult and MakeRobust.

Blame assigning: If in Step 1 of FL_{Main} all players broadcast \perp then P_c is clearly corrupted as he is no longer complaining about an inconsistency. If all players agree on the inconsistency (z, v) then the first inconsistency certainly got detected in round z and P_v complained about it. So the fault-localization of the corresponding sub-module outputs a player set as required.

Otherwise, in Step 2a, either P_k is corrupted and ignored P_i 's information (remember that i is the smallest round broadcast, so an honest P_k cannot have been informed in an earlier round) or P_j is corrupted and did not inform all players about an inconsistency in round i . So, in both cases at least one player in $\mathcal{S} = \{P_j, P_k\}$ is corrupted. In Step 2b, either $P_{k'}$ untruly claims that he got informed by P_j about an inconsistency or P_j was corrupted and informed only parts of the players in round i . Therefore, at least one player in $\mathcal{S} = \{P_j, P_{k'}\}$ is corrupted. \square

5.6 Reconstructing Output

Each output y_1, \dots, y_{n_O} is reconstructed by executing RecOutput for its sharing.

Protocol RecOutput

In: PE-sharing of y in the player set \mathcal{P}' .

1. The players execute the protocol RobustRec n' times to reconstruct y to all players in \mathcal{P}' .
2. Each player in \mathcal{P}' sends his reconstructed value to all players in $\mathcal{P} \setminus \mathcal{P}'$. Then, each player in $\mathcal{P} \setminus \mathcal{P}'$ chooses as reconstructed output y the value that he received by more than $n'/2$ players.

Lemma 16. *The output reconstructed by RecOutput is the shared secret y with probability at least $1 - n^3/q$.*

Proof. This holds, as each honest player in \mathcal{P}' reconstructs y with probability at least $1 - n^2/q$. So, the probability that all honest players reconstruct y , which means that y is sent by a majority, is at least $1 - n^3/q$ as claimed. \square

5.7 Complexity Analysis

From inspecting the protocols, it follows that the protocols and modules presented in this Chapter have following communication complexities where $\kappa = \log_2 q$ and we use that $\kappa \geq \log_2 n'$. The numbers for total do not include the n' bits broadcast during the fault detection when the module is used as a section in a player-elimination protocol (see Section 2.2). All values are upper bounds.

	GenRandom
PC	$2n'^2\kappa$
WFD ^m	$2mn'^2\kappa + 4n'^3\kappa$
FL ^m	$2(m+1)\kappa + n'\kappa + \text{BC}(\log_2 m) + n'\text{BC}(\kappa+1) + 4\text{BC}(\kappa)$
total	$4mn'^2\kappa + 2(m+1)\kappa + 4n'^3\kappa + n'\kappa + \text{BC}(\log_2 m) + n'\text{BC}(\kappa+1) + 4\text{BC}(\kappa)$

Table 5.1: Communication complexities of module GenRandom.

	Multiply
PC	$4n'^2\kappa$
WFD ^m	$4mn'^2\kappa + 9n'^3\kappa + 5n'^2\kappa$
FL ^m	$2m\kappa + 2n'^2\kappa + \text{BC}(\log_2 m) + 2n'\text{BC}(k+4) + 5\text{BC}(k)$
total	$8mn'^2\kappa + 2m\kappa + 9n'^3\kappa + 7n'^2\kappa + \text{BC}(\log_2 m) + 2n'\text{BC}(k+4) + 5\text{BC}(k)$

Table 5.2: Communication complexities of module Multiply.

	GenTriple
PC	$8n'^2\kappa$
WFD ^m	$8mn'^2\kappa + 17n'^3\kappa + 5n'^2\kappa$
FL ^m	$2m\kappa + 2n'^2\kappa + \text{BC}(\log_2 m) + 2n'\text{BC}(k+4) + 5\text{BC}(k) + n'\text{BC}(2)$
total	$16mn'^2\kappa + 2m\kappa + 17n'^3\kappa + 7n'^2\kappa + \text{BC}(\log_2 m) + 2n'\text{BC}(k+4) + 5\text{BC}(k) + n'\text{BC}(2)$

Table 5.3: Communication complexities of module GenTriple.

	EvalMult
PC	$2n'^2\kappa$
WFD	–
FL	$2n'^2\kappa + 5\text{BC}(\kappa) + \text{BC}(1)$
total	$4n'^2\kappa + 5\text{BC}(\kappa) + \text{BC}(1)$

Table 5.4: Communication complexities of module EvalMult.

	Main
PC	$[\mathcal{O}(n^2) + \mathcal{O}(n^2)\text{BC}(k)]n_I + 18mn'^2\kappa + 17n'^3\kappa + \mathcal{O}(n'^2\kappa) + \mathcal{O}(n'^3\kappa)n_O$
WFD	–
FL	$2m\kappa + 2n'^2\kappa + \text{BC}(\log_2 m) + \mathcal{O}(n')\text{BC}(k)$
total	$[\mathcal{O}(n^2) + \mathcal{O}(n^2)\text{BC}(k)]n_I + 18mn'^2\kappa + 2m\kappa + 17n'^3\kappa + \mathcal{O}(n'^2\kappa) + \text{BC}(\log_2 m) + \mathcal{O}(n')\text{BC}(k) + \mathcal{O}(n'^3\kappa)n_O$

Table 5.5: Communication complexities of module Main.

ShareInput	$\mathcal{O}(n^2) + \mathcal{O}(n^2)\text{BC}(\kappa)$
RecOutput	$2n'^3\kappa + nn'\kappa$

Table 5.6: Communication complexities of protocols ShareInput and RecOutput.

Theorem 1. *There exists a MPC protocol which computes an arbitrary function and is secure with probability $2^{-\kappa + \mathcal{O}(\log n)}$ where n is the number of players and κ the security parameter. The function is computed over a Galois field $GF(q)$ with $q > \max\{2^\kappa, n\}$ and, if it has M multiplication gates, R random gates (both polynomial in n) and an arbitrary number of addition gates, n_I inputs and n_O outputs, then the communication complexity is bounded by $[\mathcal{O}(n^3\kappa) + \mathcal{O}(n^3)\text{BC}(\kappa)]n_I + 18(M + R)n^3\kappa + \mathcal{O}(n^4\kappa) + \mathcal{O}(n^2)\text{BC}(\kappa) + \mathcal{O}(n^4\kappa)n_O$.*

Proof. The error probability is composed by up to t repetitions of the segment S_{Main} where the security holds except with probability at most $3/(q-1)$ each time, so at most $3n/(q-1)$ in total, and the reconstruction, which works except with probability at most n^3/q . Both is in $2^{-\log_2 q + \mathcal{O}(\log n)} = 2^{-\kappa + \mathcal{O}(\log n)}$. The communication complexity is bound by $t < n/2$ times executing S_{Main} and n_O output reconstruction using RecOutput. \square

Broadcast is generally a very expensive operation, as it usually does not exist physically and must be emulated by some protocol. Therefore, we carefully tried to minimize the number and length of messages that need to be broadcast in our MPC protocol. For $t < n/2$, this emulation can be either done by a protocol based on computational assumptions [Nie03] or by the information theoretically secure broadcast protocols which presume a setup phase [PW96]. For both cases we consider the evolving total communication complexities when we use the most efficient protocols known:

Under the strong RSA assumption and assuming the RSA signatures are secure, l bits can be broadcast with complexity $\mathcal{O}(n^2l + n^3\kappa)$ [Nie03]. We have $\mathcal{O}(n^3)n_I + \mathcal{O}(n^2)$ broadcasts of κ -bit messages, resulting in $\mathcal{O}(n^6\kappa n_I + n^5\kappa)$. This gives us the bound $\mathcal{O}(n^6\kappa n_I + (M + R)n^3\kappa + n^5\kappa + n^3\kappa n_O)$ on the communication complexity of the overall protocol. This has to be compared with the bound $\mathcal{O}(n^6\kappa(n_I + n_O) + (M + R)n^7\kappa)$ of [CDD⁺99] in the same model.

For information-theoretical secure broadcast we consider [PW96]. Using this, for broadcasting one bit, we need to communicate $\mathcal{O}(n^6)$ bits if the number of

broadcasts in the protocol is independent of the function (and therefore already known during the setup phase) or $\mathcal{O}(n^{17})$ bits otherwise. This gives us the bound $\mathcal{O}(n^{21}\kappa n_I + (M + R)n^3\kappa + n^4\kappa + n^3\kappa n_O)$ on the communication complexity of the overall protocol. This has to be compared with the bound $\mathcal{O}(n^{21}\kappa(n_I + n_O) + (M + R)n^{22}\kappa)$ of [CDD⁺99] in the same model.

Our MPC protocol is atypical for a player-elimination protocol as it has only a single computation segment, whereas the optimal segment count would be in the order of n^c for some $c \geq 1$ (see Chapter 2). A consequence of this property is, that the cost for repetitions, that are necessary when a corrupted player causes an incorrect segment output, dominates the communication complexity.

We tried to introduce a protocol segmentation as in the protocol presented in [HM01] which is for $t < n/3$. There, the players always have their intermediate values shared with a polynomial of degree t and always re-share the factors before each multiplication to degree t' and afterwards the product back to degree t . However, in our case, where up to $t < n/2$ player can be corrupted, the total elimination of an honest player is not admissible if we want to use the intermediate values in the on-going computation: As there could be up to $(n - 1)/2$ corrupted players, and therefore only $(n + 1)/2$ honest ones, we need the shares of *all* honest players together to reconstruct the secret. This is an inevitable property of the secret sharing, as the adversary can corrupt any smaller set of players. To overcome this problem, we considered re-sharing protocols where the set of eliminated players assists the computation. With this help, the re-sharing is principally feasible. But as this protocol can only use the technique of player-elimination in a very limited fashion – already eliminated players cannot be eliminated once again – the communication complexities rendered the application of such a re-sharing unfavorable.

Using a different view of player-elimination, where the pair of potentially corrupted players does not get excluded completely from further computation, we succeeded to introduce a segmentation of the computation protocol, as desired. Thereby, we use the fact that when we find a pair of players with one corrupted and one honest player, then the honest one is always certain that the other one is corrupted. So, the two players can take part in the on-going computation when we assert that they never have to communicate with each other again. However, this idea could not be integrated into this thesis any more and is described in [BHR].

Chapter 6

Conclusions

6.1 Summary

We give a summary of the results achieved in this thesis:

- We studied the technique of player elimination in multi-party protocols and presented the idea of modules, a concept to modularize protocols in the player-elimination framework. This allows to compose multi-party protocols which make use of player-elimination. We considered the general sequential composition of two modules, as well as how to do efficient parallel self-composition of modules.
- We studied the problems of signing values and verifiable secret sharing in player-elimination protocols. The resulting protocols have a relaxed correctness property. I.e., after the actual computation, the players perform some verification steps and the correctness only holds when no honest player complains about an inconsistency thereby. If there is a complaint, however, there exists a protocol which localizes a pair of players with at least one of them being corrupted.
- We presented a general MPC protocol for the secure-channels model with broadcast, which provides security against an active, adaptive threshold adversary that is allowed to corrupt up to $t < n/2$ out of the n players. The protocol broadcasts only $\mathcal{O}(n^2)$ field elements in total (without the input stage) and communicates $\mathcal{O}(n^3)$ field elements per multiplication over the pairwise channel. Therefore, it is much more efficient than the previously known MPC-protocols for this model. To specify this protocol, we heavily used the introduced modularization of player-elimination protocols.

6.2 Future Work

It might be interesting to further investigate a view of player-elimination, where the set of potentially corrupted players does not get excluded completely from further computation. Using this idea, one could also consider the player-elimination technique in a model with general adversary structures, where the potential total elimination of an honest player generally cannot be tolerated.

Appendix A

Module Index

name	parallel	δ	γ	section	page
ComputeSharing	yes	$1/q$	0	4.3	28
EvalMult	no	0	0	5.4	46
GenRandom	yes	$1/q$	0	5.2	38
GenTriple	yes	$3/(q-1)$	0	5.2	43
ICDistVal	no	$1/(q-1)$	0	3.2	16
ICSign	no	$1/(q-1)$	0	3.3	19
Main	no	$3/(q-1)$	$3/(q-1)$	5.5	47
MakeRobust	no	$1/(q-1)$	0	4.4	33
Multiply	yes	$3/(q-1)$	0	5.2	40
PreRec	no	0	0	4.2	25
PreShare	yes	$1/q$	0	4.2	22
Reconstruct	no	0	0	4.3	31

Appendix B

Task Description



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Theoretical Computer Science

Prof. Ueli Maurer
Zuzana Beerliova
Martin Hirt

Master's Project for Micha Riser

September 15, 2004 – March 14, 2005

Efficient Multi-Party Protocols

1 Introduction

Secure multi-party computation allows a set of n players to distributively compute a function of their inputs privately and correctly, even if some of the players are dishonest [GMW87, BGW88, CCD88]. The dishonesty of players is modeled by a *central adversary* corrupting players. The maximal number t of players that can be tolerated to be corrupted by the adversary depends on the strength of the adversary and on the available communication model. In this work, we focus on the secure-channels model, i.e., all communication channels are perfectly secure (and synchronous), and hence, the computational power of the adversary does not need to be bounded. In this model, one can tolerate a passive adversary that reads all internal information of up to $t < n/2$ players, or an active adversary that takes full control over up to $t < n/3$ players [BGW88, CCD88]. When broadcast channels are available, one can even tolerate an active adversary corrupting up to $t < n/2$ players [RB89, Bea91, CDD⁺99].

2 Description

The goal of this work is to find *efficient* multi-party protocols for the secure-channels model with broadcast, for a reasonable complexity measure. Up to date, the most efficient protocol for this model is the one of [CDD⁺99] with an improvement of [Feh00], which requires $\Omega(n^4\kappa)$ bits of communication over the broadcast channels per multiplication, where n denotes the number of players and κ the security parameter. When instantiating the broadcast primitive with the most efficient cryptographically secure broadcast protocol, this sums up to $\Omega(n^7\kappa)$ bits. This strongly contrasts with the protocols for the secure-channels model (without broadcast) and with those for the cryptographic model, where only $\mathcal{O}(n^2\kappa)$ bits are required [HM01, HN04].

The core technique used for constructing efficient protocols in the secure-channels model is “player elimination” [HMP00]. However, this technique is basically limited to adversaries corrupting at most $t < n/3$ players and hence cannot immediately be applied to the protocol of [CDD⁺99], where $t < n/2$ is admissible. In [HN04], the communication complexity of cryptographic protocols (also with $t < n/2$) was substantially reduced by using techniques similar to player elimination. Similar ideas might be applicable in the secure-channels model with broadcast. Another approach might be to bundle the broadcast invocations in [CDD⁺99]: This would then allow to use amortization techniques in the broadcast subprotocol [FH04].

Another different approach towards efficient protocols might be to consider non-optimal models. Candidates for non-optimality might include lower thresholds t or restricted classes of circuits (e.g. NC_1).

3 Tasks

The following is an (incomplete) list of tasks:

- Study of the literature (e.g., [CDD⁺99, HMP00, HM01, HN04, FH04]),
- try to combine [FH04] with [CDD⁺99] and NC_1 -circuits,
- try to combine [HMP00] with [CDD⁺99],
- develop new protocols.

The results have to be presented in a talk by the end of the project. Some instructions about the documentation can be found in the enclosed leaflet.

References

- [Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.
- [BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, 1988.
- [CDD⁺99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology — EUROCRYPT '99*, Lecture Notes in Computer Science, 1999.
- [Feh00] Serge Fehr. Personal communication, 2000.
- [FH04] Matthias Fitzi and Martin Hirt. Efficient broadcast of long messages. Manuscript, 2004.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.
- [HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer-Verlag, August 2001.
- [HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer-Verlag, December 2000.
- [HN04] Martin Hirt and Jesper Buus Nielsen. Upper bounds on the communication complexity of cryptographic multiparty computation. Cryptology ePrint Archive, Report 2004/318, November 2004. <http://eprint.iacr.org/>.
- [RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pages 73–85, 1989.

Bibliography

- [Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.
- [BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.
- [BHR] Zuzana Beerliova, Martin Hirt, and Micha Riser. Manuscript.
- [Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136. IEEE Computer Society, 2001.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, 1988.
- [CDD⁺99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology — EURO-CRYPT '99*, Lecture Notes in Computer Science, 1999.
- [CDF01] Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In *Advances in Cryptology - CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 503–523. Springer-Verlag, 2001.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.
- [GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM Press, 1998.

- [HM97] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 25–34, August 1997.
- [HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer-Verlag, August 2001.
- [HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer-Verlag, December 2000.
- [HN04] Martin Hirt and Jesper Buus Nielsen. Upper bounds on the communication complexity of cryptographic multiparty computation. Cryptology ePrint Archive, Report 2004/318, November 2004. <http://eprint.iacr.org/>.
- [Nie03] Jesper Buus Nielsen. On protocol security in the cryptographic model. Dissertation Series DS-03-8, BRICS, Department of Computer Science, University of Aarhus, August 2003.
- [PW96] Birgit Pfitzmann and Michael Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. Research report, November 1996. Submitted for Publication.
- [RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multi-party protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pages 73–85, 1989.
- [Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
- [Yao82] Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164. IEEE, 1982.