Multiparty Computations Information-Theoretically Secure Against an Adaptive Adversary

Stefan Dziembowski

PhD thesis

BRICS
Department of Computer Science
University of Århus
Denmark

December 2000, Revised: January 2002 **Supervisor:** Ivan Damgård

Abstract

In this thesis we study a problem of doing Verifiable Secret Sharing (VSS) and Multiparty Computations (MPC) in a model where private channels between the players and a broadcast channel is available. The adversary is active, adaptive and has an unbounded computing power. The thesis is based on two papers [CDD00, CDD$^+$99].

In [CDD00] we assume that the adversary can corrupt any set from a given *adversary* structure. In this setting we study a problem of doing efficient VSS and MPC given an access to a secret sharing scheme (SS). For all adversary structures where VSS is possible at all, we show that, up to a polynomial time black-box reduction, the complexity of adaptively secure VSS is the same as that of ordinary secret sharing (SS), where security is only required against a passive, static adversary. Previously, such a connection was only known for linear secret sharing and VSS schemes.

We then show an impossibility result indicating that a similar equivalence does not hold for Multiparty Computation (MPC): we show that even if protocols are given black-box access for free to an idealized secret sharing scheme secure for the access structure in question, it is not possible to handle all relevant access structures efficiently, not even if the adversary is passive and static. In other words, general MPC can only be black-box reduced efficiently to secret sharing if extra properties of the secret sharing scheme used (such as linearity) are assumed.

The protocols of [CDD$^+$99] assume that we work against a *threshold* adversary structure. We propose new VSS and MPC protocols that are substantially more efficient than the ones previously known.

Another contribution of [CDD$^+$99] is an attack against a Weak Secret Sharing Protocol (WSS) of [RBO89]. The attack exploits the fact that the adversary is adaptive. We present this attack here and discuss other problems caused by the adaptiveness (one of the examples are taken from [CDD$^+$01]).

All protocols in the thesis are formally specified and the proofs of their security are sketched.

Contents

Chapter 1

Introduction

We consider protocols executed by a group of players. The protocols will be designed in a secure way. Very informally this means that, even if a certain players get corrupted, the coalition of them will not be able to learn the secrets of the honest players, or to influence the result of the protocol (this will hold up to some limits). We will first (Section 1.1) give an informal introduction to the types of protocols that we consider (the formal definitions appear in Chapter 1). The thesis is based on two papers [CDD00, CDD$^+$99]. In Section 1.2 we give an overview of [CDD00]. Then, in Section 1.3 we describe the contribution of [CDD$^+$99].

1.1 SS, VSS and MPC

We consider three related problems, namely *secret sharing* (SS), *verifiable secret sharing* (VSS) and *multiparty computation* (MPC).

SS was introduced by Shamir [Sha79] and generalized by Itoh et al.[ISN87]: a *Dealer* has a secret s and distributes a set of *shares* $s_1, ...s_n$ to n players, such that s can be reconstructed only by certain *qualified* subsets of players while unqualified subsets have no information about s. The collection of qualified sets is called the *access structure*. We stress that in [CDD00] we consider secret sharing for general access structures (while in [CDD$^+$99] we restrict ourselves to the threshold schemes where the access structure may only consist of all sets of size larger than some threshold). It is assumed that the dealer computes the shares correctly, and that players input correct shares for reconstruction.

When these assumptions are dropped, we get the (seemingly) harder problem of VSS (Chor *et al.* [CGMA85]): here, some of the players, including the dealer, may not follow the protocol. It may even be the case that some of them turn bad dynamically as the protocol proceeds, as long as the total set of bad players remains unqualified. Still the remaining honest players should be able to verify that they have shares of a well defined secret, while the cheating players should get no information about it, if the dealer is honest. Finally, the honest players should be able to reconstruct the secret, even against the actions of the cheating players.

Our final, still more general problem is that of MPC (A. Yao [Yao82], Goldreich, Micali and Wigderson [GMW87]): here, all players have a secret input, and the goal is to compute an agreed functions of these inputs, while maintaining privacy of the inputs and correctness of the result, again assuming that the set of bad players at any given time is unqualified.

1.2 The contribution of [CDD00]

1.2.1 Introduction

The classical results in unconditionally secure VSS/MPC by Ben-Or, Goldwasser and Wigderson[BOGW88], Chaum, Crépeau and Damgård [CCD88] and Rabin and Ben-Or [RBO89] can be seen as results that build efficient VSS and MPC protocols based on Shamir's threshold secret sharing scheme, in the model where secure channels are assumed to exist between every pair of players.

Gennaro [Gen95] was the first to consider VSS secure for general access structures. Then Hirt and Maurer [HM97] characterized exactly those general access structures for which VSS and MPC are possible.

Continuing the line of research from [BOGW88, CCD88, RBO89], Cramer, Damgård and Maurer [CDM00] have shown that VSS and MPC for general access structures can be built efficiently on top of any linear SS scheme (see also [CDD+99]). Thus a natural final step is to ask what happens if we start from an *arbitrary* SS scheme?

Informally, what we show in this paper is that VSS is as easy to achieve efficiently as ordinary SS, more precisely, there exists an efficient reduction that builds a secure VSS protocol from any SS scheme secure for the same access structure, provided VSS is possible at all for that structure. Since VSS trivially implies SS, this is an optimal result.

Similarly, showing that MPC in this sense is no harder than SS would be an optimal result. However, we show an impossibility result indicating that there is not much hope of proving this. A reduction showing how to do secure MPC for some access structure given *any* SS-scheme for that structure cannot make any assumptions on the way the SS-scheme works. So the natural approach is to treat the secret sharing as a black box, relying only on the functionality that follows from the definition of secure SS. We show that if we restrict ourselves to such reductions, there are access structures that cannot be handled efficiently, where by "efficiently", we mean that protocols run in time polynomial in the number of players, counting usage of the SS-scheme as only one step.

Thus, general (black-box) reductions building MPC from SS would have to either depend on the particular kind of SS-scheme being used (such as reductions depending on linearity of the SS scheme) or be inefficient on some access structures. This may be seen as an indication that, as far as applicability to unconditionally secure MPC is concerned, there is a fundamental difference between linear SS schemes and general ones. By contrast, it is shown in [CDM00] that general SS does suffice for *computationally* secure MPC.

1.2.2 A More Detailed View

To explain our results in more detail, we need to describe more precisely the model we use: we have a set of n players, connected pairwise by private channels, moreover, a broadcast channel is also available. We are given a *monotone access structure* Γ, that is, Γ consists of subsets of the player set, such that $A \in \Gamma$ and $A \subset B$ implies $B \in \Gamma$. For instance, Γ could consist of all subsets of size $n/2$ or more. For convenience in the following, we will instead talk about the family $\mathcal{F}$ of subsets not in Γ. Such a complement of an access structure is known as an *adversary structure*, a general notion introduced by Hirt and Maurer[HM97].

Finally we have an *adversary* who can corrupt any subset A of players, as long as $A \in \mathcal{F}$. This is called an $\mathcal{F}$-adversary. The adversary may be *passive*, meaning that he just gets access to all data of corrupted players, or *active*, meaning that he takes control over corrupted players and may make them deviate from the protocol they were supposed to follow. Orthogonal to this, we distinguish between *static* adversaries who must decide before the protocol execution who to corrupt, and *adaptive* adversaries [RBO89] who may decide dynamically throughout the protocol whom to corrupt, as long as the total corrupted set is in $\mathcal{F}$. For more on the role of adaptiveness see Section 1.4.

In this model, security of secret sharing (SS) can be rephrased to the requirement that a passive, static adversary gets no information about the secret when it is distributed. And that the secret can be reconstructed, even if the adversary can make the corrupted set of players fail to input shares for reconstruction.

In order to talk about the complexity and error probability of an SS scheme, we will think of it as two probabilistic algorithms $\mathrm{Distr}_{\mathrm{SS}}$ and $\mathrm{Recon}_{\mathrm{SS}}$, where $\mathrm{Distr}_{\mathrm{SS}}$ gets as input the secret s, the number of players n, it then computes a set of n shares as output. Sharing s means that one of the players, the dealer, runs $\mathrm{Distr}_{\mathrm{SS}}$ and sends the shares privately to the players. For reconstruction, some subset of the shares are broadcast, and each player can run $\mathrm{Recon}_{\mathrm{SS}}$, which gets as input the subset of the shares, the number n, and outputs a value s'.

Thus, underlying this, we have not just one, but a family of adversary structures, one for each n. We will say that $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$ is secure against the family $\{\mathcal{F}_n\}_{n=1}^{\infty}$ if the following hold for any n and any static, passive $\mathcal{F}_n$-adversary:

- Assuming the dealer is not corrupted, the adversary gets no (Shannon) information about the secret after it is shared.
- Sharing s and running $\mathrm{Recon}_{\mathrm{SS}}$ on input a set of shares not in $\mathcal{F}_n$ results in output s. Running $\mathrm{Recon}_{\mathrm{SS}}$ on input set of shares of players in $\mathcal{F}_n$ results in output a special symbol $\perp$, indicating that the input set was unqualified.

Note that it would not make any essential difference if the behavior of $\mathrm{Recon}_{\mathrm{SS}}$ on unqualified sets was left undefined: one can always test, with arbitrarily small error, if a set is qualified by sharing a random secrets and testing if they can be reconstructed from the subset in question.

Going to VSS, we will think of this as first a protocol $\mathrm{Distr}_{\mathrm{VSS}}$ for distribution, that takes a security parameter k as input, and where one of the players, the dealer, gets the secret as private input. Secondly, there is a protocol for reconstruction $\mathrm{Recon}_{\mathrm{VSS}}$, where each player starts from his view of the distribution (which we think of as his share), interacts with other players and reconstructs a value for the secret (for a formal specification see Section 4.3.1). Such a VSS is secure against the family $\{\mathcal{F}_n\}_{n=1}^{\infty}$ if the following hold for all n and all adaptive, active $\mathcal{F}_n$-adversaries:

- After distribution, a secret is uniquely defined from the views of the set of uncorrupted players, except with probability negligible in k.
- If the Dealer remains uncorrupted, the adversary has a negligible (in k) amount of information about the secret.
- Reconstruction of the secret results in all uncorrupted players reconstructing the secret defined at distribution time, except with negligible probability in k.

Here, negligible in k means that the quantity converges to 0 as a function of k faster than any polynomial fraction.

Hirt and Maurer [HM97] show that designing a VSS protocol is possible if and only if the adversary structure is $\mathcal{Q}_2$: for any two sets in the structure, their union is not the whole player set. The obvious question is therefore: when can it be done *efficiently*? – which we here will take to mean in polynomial time in the number of players. The number of $\mathcal{Q}_2$ structures is doubly exponential in n (see [HM97]) so it follows from a counting argument that we cannot hope to handle all structures efficiently.

One way we could hope to get upper and lower bounds for VSS is by relating it to the simpler problem of SS. Cramer, Damgård and Maurer [CDM00] show that any *linear* secret sharing scheme implies a VSS with polynomially related efficiency [1]. Here, a linear secret sharing scheme is one in which the shares and the secret are elements in a finite field, and the secret can be obtained as a linear function of the shares. Such schemes can be based on monotone span programs [KW93, Bei96].

Our first main contribution is a similar result, that holds for *any* secret sharing scheme. Since VSS trivially implies SS without significant loss of efficiency, this is the best result we can hope for.

[1] In fact, they worked in a model with no broadcast and zero error but a stronger condition on the adversary structure. However, the result translates to our model using the techniques of [CDD+99]

Theorem 1.1
Given any secure secret sharing scheme $\mathcal{S} = (\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$, secure against a family $\mathcal{F}$ of $\mathcal{Q}_2$ adversary structures, there exists a VSS protocol secure against $\mathcal{F}$ with complexity polynomial in n, k and the running time of $\mathcal{S}$.

We show this by combining the information checking idea of [RBO89] and [CDD$^+$99] with a new technique for upgrading from static to adaptive security.

Note that one way to prove such a result would be to provide a kind of "efficient compiler" that takes as input the algorithms $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$ and produces a VSS protocol for the same adversary structure, with polynomially related efficiency. We prove a slightly stronger result, in that we construct a single VSS protocol that works when given only black-box access to the algorithms of any SS scheme.

A seemingly even harder problem than VSS is that of MPC, as described above. For this problem a set of results very similar to those for VSS is known: In was proved in [HM97] that the $\mathcal{Q}_2$ condition on the adversary structure is necessary and sufficient for MPC to be possible. In [CDM00] it is shown how to perform secure distributed multiplication efficiently when given a linear SS scheme, and how efficient MPC follows from this given also a commitment scheme with extra homomorphic properties. Using a VSS construction from [CDD$^+$99] as commitment and exploiting its linearity, it then follows that a linear SS scheme for a $\mathcal{Q}_2$ adversary structure implies an MPC protocol for the same structure with polynomially related efficiency, secure against an active, adaptive adversary.

So it is natural to ask whether a result similar to Theorem 1.1 holds for MPC? As for VSS, this is the best result one can hope for in terms of poly-time efficiency. However, we show a result implying that a reduction of the type we provided to prove Theorem 1.1 does not exist for MPC, not even if we assume the adversary is passive and static. Informally, we show that MPC protocols which get black-box access for free to secret sharing, but do not use any special properties of the SS scheme in question, cannot handle all $\mathcal{Q}_2$ adversary structures efficiently.

Since, in both the passive and active models, distributed addition is easily handled (as we will argue later on), it follows that it is essentially distributed multiplication, or equivalently, Oblivious Transfer, that prohibits efficient construction of MPC protocols from black-box SS-schemes.

To make this more precise, we first fix (for concreteness) a function which will turn out to be hard to compute securely for all $\mathcal{Q}_2$ structures, namely the function f_{AND} which is the AND of n input bits, one from each player.

We will allow protocols to be constructed non-uniformly over n, k, this only makes the impossibility result stronger. Thus for a fixed value of n, k, we can write down the computing done by each player at each round of the protocol as a Boolean circuit. Sending of messages translates directly to wires connecting these circuits, and the view of a player becomes the collection of values that are handled by his part of the circuit. In the following, we will only be interested in what happens when both n and k go to infinity, so for simplicity, we only look at cases where $n = k$.

So we define a protocol for computing f_{AND} for an arbitrary number of players n as a family of Boolean circuits of this type $\{C_n \mid n = 1, 2, ..\}$, where C_n specifies the actions of the protocol for n players (and security parameter $k = n$). A protocol is polynomial time, if each C_n has size bounded by some polynomial.

We will say that a protocol computes f_{AND} securely against a family of adversary structures $\{\mathcal{F}_n\}_{n=1}^{\infty}$, if the following two conditions hold for all static, passive $\mathcal{F}_n$-adversaries [2]:

correctness For any set of inputs bits $b_1, ..., b_n$, the protocol computes as result for all players $b_1 \wedge ... \wedge b_n$, except with negligible probability (in $n = k$).

privacy For $A \in \mathcal{F}_n$, if the adversary corrupts A, he learns nothing about inputs bits outside A, beyond what is implied by input bits in A and $b_1 \wedge \cdots \wedge b_n$ (except for an amount negligible in $n = k$).

Note that requiring only passive, static security makes the impossibility result stronger.

Finally, we discuss how to model protocols that are allowed to use SS secure against the adversary structures in some family $\{\mathcal{F}_n\}_{n=1}^{\infty}$ but without relying on which particular SS-scheme is used.

Note that it would not work to give the protocol black-box access to the algorithms $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$ of some scheme. The shares thus produced depend of course on the algorithm, so as a result the protocol might still take actions depending on the particular scheme used. To avoid this, we give in stead the protocols access to an idealized secret sharing where the shares are replaced by a random number that is unrelated to any particular SS scheme.

More precisely, we allow protocols to make use of an extra incorruptible player T with unbounded computing power called an SS-$\mathcal{F}$*-oracle*. T will implement an "ideal" SS w.r.t. $\mathcal{F}_n$ whenever there are n players: any player can send a message "share s" to T containing a secret s. Then T will remember s and distribute to all players a randomly chosen but unique number ID(s). At any later time, the protocol can issue a request to T "Reconstruct A, ID(s)" meaning that the players in the subset A would like to reconstruct the secret identified by ID(s). Now, T computes whether A is in $\mathcal{F}_n$. If not, T will send s to the players in A. Otherwise T will only say that A is in $\mathcal{F}_n$.

When measuring running time, access to T counts only as one step - the protocol is not charged for the internal computing done by T [3]. A protocol with such an extra player is called an *SS-oracle protocol*. Such a protocol is said to compute a function securely against $\mathcal{F}$ if it can do so when given access to an SS-$\mathcal{F}$-oracle. Our result now is:

[2] Usually, when defining security of multiparty computation, one cannot separate correctness and privacy as we do here. However, in our particularly simple case of static and passive security with deterministic function, this is not a problem

[3] thus, in our circuit model, a call to T is modeled as a single oracle gate doing internally all T's computation.

Theorem 1.2
There exist families $\mathcal{F}$ of $\mathcal{Q}_2$ adversary structures, such that no polynomial-time SS-oracle protocol computes f_{AND} securely against $\mathcal{F}$.

Note that if we were talking about protocols with no oracle access, we would have a lemma already shown in [HM97] for which a simple counting argument suffices: Let a maximal $\mathcal{Q}_2$ adversary structure be one to which we cannot add a new subset without loosing the $\mathcal{Q}_2$ property. It is then easy to see that there are doubly exponentially many maximal $\mathcal{Q}_2$-adversary structures on n players. On the other hand, there are only exponentially many different protocols that can be specified by a number of bits polynomial in n. Thus, if the result was false, there would exist some protocol that could handle several different access structures. But the same protocol cannot be secure for two different maximal $\mathcal{Q}_2$ structures because it would then be secure for their union, which is not $\mathcal{Q}_2$.

However, in our case this argument breaks down: our protocols have access to an oracle giving answers that depend on the access structure in question, thus an oracle protocol may take different actions for different access structures. The main technical problem we solve is to show that even this is not sufficient to do MPC efficiently for all structures.

Note that Theorem 1.2 does not rule out that a result similar to Theorem 1 could hold for MPC: it may be the case that for every (class of) SS-scheme(s) there exist MPC protocols with polynomially related efficiency that depend on the particular scheme, or class of schemes considered. Only the existence of a general black-box reduction is ruled out.

1.3 The contribution of [CDD+99]

In [CDD+99] we study the problem of designing efficient VSS and MPC protocol secure against an active and adaptive adversary. As in Section 1.2.2 we assume that the players are connected with private authenticated channels and a broadcast channel is available. The only difference is that we consider only the threshold adversary structures. More precisely we assume that the adversary can corrupt any minority of the players. In other words the adversary structure consists of sets of size smaller than $n/2$ (where n is the number of players).

1.3.1 Previous work on the protocols secure for threshold structures

As the first general solution to this problem, [GMW87] presented a protocol that allows n players to securely compute an arbitrary function even if a central adversary actively corrupts any $t < n/2$ of the players and makes them misbehave maliciously. However, this protocol assumes that the adversary is computationally bounded. In a model with secure and authenticated channels between each pair of players (the secure-channels model), [BOGW88, CCD88] proved that unconditional security is possible if at most $t < n/3$ of the players are actively corrupted. This bound was improved in [RBO89, Bea91] to $t < n/2$ by assuming the existence of a broadcast channel.

1.3.2 Contributions

Rabin and Ben-Or [RBO89, Rab94] proposed VSS and MPC protocols, secure against an adversary that can actively corrupt any minority of the players. In [CDD$^+$99], we observe that a subprotocol of theirs, known as weak secret sharing (WSS, a type of unconditionally secure commitment scheme), is not secure against an adaptive adversary, contrary to what was believed earlier (and claimed in [RBO89, Rab94]). However, the VSS protocol of [RBO89, Rab94] is in fact adaptively secure. See Section 1.4 for more on adaptiveness.

We propose new and adaptively secure protocols for WSS, VSS and MPC that are substantially more efficient than the original ones of [RBO89] and later protocols by Beaver [Bea91]. To obtain error probability $2^{-k+O(\log n)}$ with n players, the VSS protocols of [RBO89, Bea91] communicate $\Omega(k^3n^4)$ bits. Our VSS protocol is constant round and uses communication $O(kn^3)$ bits, to achieve the same error probability $2^{-k+O(\log n)}$.

This improvement is based in part on a more efficient implementation of *Information Checking Protocol*, a concept introduced in [RBO89] which can be described very loosely speaking as a kind of unconditionally secure signature scheme. Our implementation is linear meaning that for two values that can be verified by the scheme, and linear combination of them can also be verified with no additional information. When using our VSS in MPC, this means that linear computations can be done non-interactively, contrary to what the implementation of [RBO89] (this property was also obtained in [Bea91], but with a less efficient Information Checking implementation).

An essential tool in MPC (provided in both [RBO89] and [Bea91]) is a protocol that allows a player who has committed to values a, b, c using WSS, to show that $ab = c$ without revealing extra information. We provide a protocol for this purpose giving error probability 2^{-k} which is extremely simple.

1.4 The role of adaptiveness

In this section we discuss the role of the adaptiveness of the adversary. We show a few examples of the protocols secure against a passive adversary and insecure against an adaptive one (for other examples the reader may consult [CFGN96]). In all the examples the adaptive insecurity is non-trivial, and the protocols may seem adaptively secure at the first sight. The section is mostly based on [CDD$^+$99] and [CDD$^+$01].

1.4.1 The WSS protocol of [RBO89]

This example is based on the WSS protocol of [RBO89]. It's adaptive insecurity was first observed in [CDD$^+$99]. The original protocol [RBO89, CDD$^+$99] was designed for the adversary structures of an threshold type. We generalize it here to general adversary structures.

Informal definition of WSS

Suppose we are working in a setting from Section 1.2.2, i.e. we are given a group of n players $\{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$ connected pairwise by secure channels and a broadcast channel. An adversary can actively corrupt players in some $\mathcal{Q}_2$ adversary structure $\mathcal{F}$. An intuitive explanation for a *weak secret-sharing (WSS) scheme* is that it is the non-computational analog of a computational commitment. It exhibits the same properties, i.e. it binds the committer to a single value after the committing phase $\mathrm{Commit}_{\mathsf{WSS}}$, yet the committer can choose not to expose this value in the opening phase $\mathrm{Open}_{\mathsf{WSS}}$. A WSS scheme for committing to a secret $s \in K$ consists of the two protocols $\mathrm{Commit}_{\mathsf{WSS}}$ and $\mathrm{Open}_{\mathsf{WSS}}$ that satisfy the following properties, with an allowed error probability 2^{-k}:

- *Secrecy*: If the dealer is honest and no honest player has yet started the protocol $\mathrm{Open}_{\mathsf{WSS}}$, then the adversary has no information about the shared secret s.
- *Correctness*: Once all currently uncorrupted players complete protocol $\mathrm{Commit}_{\mathsf{WSS}}$, there exists a *fixed* value, $r \in K$, such that the following requirements hold:
  1. If the dealer is uncorrupted throughout protocols $\mathrm{Commit}_{\mathsf{WSS}}$ and $\mathrm{Open}_{\mathsf{WSS}}$ then r is the shared secret, i.e. $r = s$, and each uncorrupted player will outputs r at the end of protocol $\mathrm{Open}_{\mathsf{WSS}}$.
  2. If the dealer is corrupted then each uncorrupted player outputs either r or `error` upon completing protocol $\mathrm{Open}_{\mathsf{WSS}}$.

We can now make the following informal definition (for a formal one see Section 4.1.1)

Definition 1.3
A $\mathcal{F}$-secure WSS scheme for committing to a secret $s \in K$ is a pair of protocols $(\mathrm{Commit}_{\mathsf{WSS}}, \mathrm{Open}_{\mathsf{WSS}})$ of two protocols that satisfy the above properties even in the presence of an active adversary who can corrupt any set of players in the adversary structure $\mathcal{F}$.

Implementation

In order to explain the attack we present a simplified version of the [RBO89] protocol which assumes digital signatures. It is in essence the same protocol but with many complicating (non relevant) details omitted. The same protocol is presented in Section 4.1.2 (it is called there an SWSS protocol) where we use it as a building block for *adaptively* secure WSS. We will assume the existence of a secret sharing scheme $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$ secure against $\mathcal{F}$.[4]

$\boxed{\mathrm{Commit}_{\mathsf{WSS}}}$

1. Let s be the secret the dealer D wants to commit to. He shares it using $\mathrm{Distr}_{\mathrm{SS}}$ to get shares $\mathrm{sh}_1, \ldots, \mathrm{sh}_n$.
2. For each i he sends sh_i to P_i with his signature on it.

$\boxed{\mathrm{Open}_{\mathsf{WSS}}}$

1. The dealer broadcasts the secret s and the random input r used by $\mathrm{Distr}_{\mathrm{SS}}$ in the previous phase.
2. Every player P_i runs $\mathrm{Distr}_{\mathrm{SS}}$ on s and r that he received from the dealer. If the obtained share of player P_i matches sh_i (that he received in the previous phase) then he broadcasts an acceptance. Otherwise he complaints by broadcasting sh_i together with the dealer's signature on it.
3. For each properly signed value sh_i broadcasted in Step 2. every player P_j checks, if the complaint was justified (by running $\mathrm{Distr}_{\mathrm{SS}}$ on s and r and comparing the share of player P_i with sh_i). If he finds a justified complaint then he rejects the opening (and outputs `error`) . Otherwise he accepts (and outputs s).

Non-adaptive security

Suppose the set A of corrupted players is fixed from the beginning. By the $\mathcal{Q}_2$ property the set $\overline{A}$ of players not in A is qualified. Let $s' \in K \cup \{\texttt{error}\}$ be the value that is reconstructed (by $\mathrm{Recon}_{\mathrm{SS}}$) from the shares of the players in $\overline{A}$. It is easy to see that the value opened in the $\mathrm{Open}_{\mathsf{WSS}}$ has to be equal either to s or to `error`. Thus, after the shares are sent (in Step 2 of $\mathrm{Commit}_{\mathsf{WSS}}$), the dealer is committed to s'.

Adaptive insecurity

We are now going to show an attack on the WSS protocol presented above. We will show that a corrupted dealer can distribute shares in such a way, that the correctness condition will be violated.

First, define an adversary structure to be a $\mathcal{Q}_3$ structure if for any *three* sets in the structure, their union is not the whole player set. The attack works for every $\mathcal{Q}_2$ adversary structure $\mathcal{F}$ that is not a $\mathcal{Q}_3$ structure. To make the picture

[4] In [RBO89, CDD+99] the secret sharing scheme was the one of Shamir [Sha79].

clearer we assume here that the dealer is a separate, always corrupted player (not belonging to the set $\{P_1, \dots, P_n\}$).

Let $\mathrm{Distr_{SS}}(s, r)$ denote the set of shares obtained by applying $\mathrm{Distr_{SS}}$ to s with random input r. Take three sets $A_0, A_1, A_3 \in \mathcal{F}$ such that their union is the whole player set. Take two secrets $s_0, s_1 \in K$. Let r_0 and r_1 be the random inputs for $\mathrm{Distr_{SS}}$ such that for every player $P_i \in A_3$ his share in $\mathrm{Distr_{SS}}(s_0, r_0)$ is equal to his share in $\mathrm{Distr_{SS}}(s_1, r_1)$. It is easy to see that such r_0 and r_1 always exist: otherwise an unqualified set A_3 of players would get information about the secret. The corrupted dealer can now send to the players in A_0 the shares from $\mathrm{Distr_{SS}}(s_0, r_0)$ and to the players in A_1 the shares from $\mathrm{Distr_{SS}}(s_1, r_1)$. Now, in the $\mathrm{Open_{WSS}}$ the adversary can make the players output s_i (for both $i = 0, 1$ by applying the following strategy.

1. corrupt the players in A_{1-i},
2. in Step 1. broadcast s_i and r_i.

Thus the correctness property is not satisfied. It is also easy to see that this attack does not work if the adversary structure satisfies the $\mathcal{Q}_3$ property.

1.4.2 The example of [CDD$^+$01]

This example comes from [CDD$^+$01]. The protocol involves three players: a dealer D and two receivers R_1 and R_2, where R_1, R_2 have no input, and D's input consists of two bits $s_1, s_2 \in \{0, 1\}$. They want to compute a function f_{send} that returns no output for D, the output for R_1 is s_1, and the output for R_2 is s_2. The adversary structure $\mathcal{F}$ contains $\{D, R_1\}$ (and its subsets), namely R_2 cannot be corrupted. The protocol proceeds as follows.

$\boxed{\pi_{\mathrm{send}}}$

1. D sends s_1 to R_1.
2. D sends s_2 to R_2.
3. Each R_i outputs the bit that was sent to it by D, and terminates. D outputs nothing and terminates.

It is easy to see that if the adversary is non-adaptive, then the protocol π_{send} securely evaluates f_{send} (the formal proof appears in [CDD$^+$01]).

We are now going to argue that the protocol is adaptively insecure (see for a formal proof see [CDD$^+$01]). Suppose the adversary is adaptive. We are going to show that if the players execute π_{send} the adversary has more power that if the players are given an access to an idealized oracle computing f_{send} (see Section 2.1 or [Can00] for the formal definitions of this notions). Suppose the adversary is adaptive. Assume the adversary has some extra knowledge about the pair (s_1, s_2), namely he knows that $s_1 = s_2$. Suppose his aim is to make R_2 always output 0. Moreover assume that he wants to avoid corrupting D, if it is possible. It is easy to see that in the model with an oracle computing f_{send} the only way to make R_1 output 0 is to corrupt D at the beginning of the protocol. On the other hand, when the protocol π_{send} is executed, the adversary can

1. corrupt R_1,
2. learn s_1 in Step 1., and
3. if $s_1 = 1$ then corrupt D and instruct him to send 0 to R_2 in Step 2., otherwise do nothing.

In this way R_2 always outputs 0, but D is corrupted only if necessary. Thus the protocol is insecure.

A similar problem appears in a practical implementation of an IC protocol (see Section 3.1.7 for more).

1.4.3 Another example

In this section we show another example of the same nature as the one in Section 1.4.2. We think it is important to present it, since the problems of this type will appear later (in Sections 4.1 and 4.2).

Suppose we are given a commitment scheme $\mathrm{CS}_1 = (\mathrm{Commit}_1, \mathrm{Open}_1)$ that allows the dealer D to commit to one bit $b \in \{0, 1\}$, and we want to construct a commitment scheme $\mathrm{CS}_n = (\mathrm{Commit}_n, \mathrm{Open}_n)$ that would allow the dealer to commit to $n > 1$ bits string $(b_1, \ldots, b_n) \in \{0, 1\}^n$. A natural way to do it is to define CS_n as a sequential composition of n executions of CS_1. More precisely implement it as follows.

$\boxed{\mathrm{Commit}_n(b_1, \ldots, b_n)}$

1. For $i = 1, \ldots, n$ execute $\mathrm{Commit}_1(b_i)$.

$\boxed{\mathrm{Open}_n}$

1. For $i = 1, \ldots, n$ execute the corresponding Open_1.
2. If all the openings were successful then output the resulting string, otherwise output `error`.

This protocol is secure if there are only two players: committer and verifier (and we are working with the standard computationally secure commitments). If we go to the distributed setting and assume some standard model (for example the threshold adversary from Section 1.3, with threshold $t > 2$), then the situation is more complicated. It is easy to see that the protocol is secure when the adversary is non-adaptive. However, if an adaptive adversary can execute the following attack.

- Corrupt one of the players, P_1, say,

- In Step 1 listen to the result of the first execution of Open_n. If the resulting bit b_1 is equal to 0, then immediately corrupt the dealer and make him halt. In this way the openings of all the remaining bits fail. Otherwise (if $b_1 \neq 0$), then do nothing.

It is easy to see that if the adversary applies this strategy, then

- the players never open any bit string with the first bit equal to 0, and
- the dealer does not get corrupted if his input bit b_1 was equal to 1.

On the other hand this aim impossible to achieve if we specify Commit_n as an idealized oracle which: (1) takes input s from the dealer in the Commit_n phase, and (2) sends s to the players in the Open_n phase (if the dealer remains honest). Thus the implementation is insecure.

1.5 Overview of the next chapters

First (Chapter 2) we introduce the definitions and the notation. The main part of the thesis is the formal specification and construction of the protocols from [CDD+99] (Chapter 4) and [CDD00] (Chapter 5). The protocols in both Chapter 4 and 5 will use the IC protocol implemented (and specified) in Chapter 3. The construction of the protocols is modular. The dependence is shown on Figure 1.1. We stress that the protocols in Chapters 3 and 4 work against a general $\mathcal{Q}_2$ adversary structure, whereas in Chapter 5 we assume that the adversary structure is of a threshold type. Finally in Chapter 6 we show the impossibility

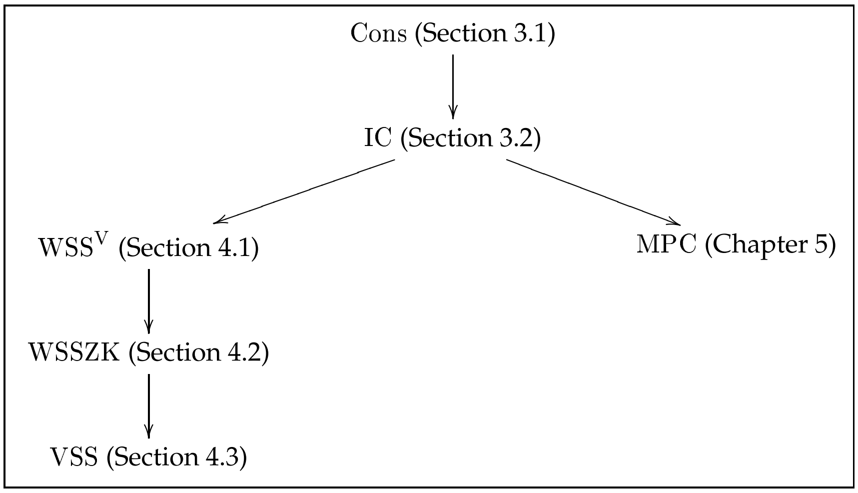

Figure 1.1: The scheme of the dependence between the protocols

result, Theorem 1.2.

1.6 Acknowledgments

I would like to express my deep gratitude to my supervisor Ivan Damgård for introducing me to the area of cryptography, for lots of discussions about the area of multiparty computations and for many valuable comments on my work.

I also thank all the other coauthors of the papers on which this thesis is based, namely Ronald Cramer, Martin Hirt, and Tal Rabin.

I am also grateful to Serge Fehr for his observation improving the efficiency of the MPC protocol from [CDD+99].

I would also like to thank Ran Canetti for lots valuable of comments on the earlier version of the thesis.

Chapter 2

Definitions and notation

2.1 The security of on-line protocols

In this section we are going to define the security of the on-line protocols. This is an extension of the definitions of the secure function evaluation of Canetti [Can00]. Let us sketch the main difference. In case of the secure function evaluation the scenario is as follows:

1. The parties take inputs.
2. The protocol is executed.
3. The parties give outputs.

In case of an (m-phase) on-line protocol the following steps are executed for $i = 1, \ldots, m$.

1. The parties take the inputs for the ith phase.
2. The protocol for ith phase is executed.
3. The parties give the outputs for the ith phase.

In case of a scenario of the first type the natural way to use a protocol σ as a sub-protocol of a protocol π is to: (1) suspend the execution of π at some point asking the parties provide some input values for σ; (2) execute σ on those values obtaining some output values; (3) resume π providing it with the values output in (2).

In case of the on-line protocols the situation is more complicated. Here, a sub-protocol σ is executed in phases that can be interleaved with some other code. More precisely (for every $i = 1, \ldots, m - 1$) immediately after the parties return the outputs of ith phase the execution of the sub-protocol is suspended and some other instructions are executed. Then, after some time the protocol σ is resumed: the parties produce the input values for the $(i + 1)$st phase and this phase starts. Note, that this input values may depend on the previous outputs.

Also, when the protocol σ was suspended the adversary can get some useful information.

In Section 2.1.3 we are going to define the security of a multiparty on-line protocol by comparing it's *real life* execution with an *ideal* one. In the next two sections we explain these notions. We will assume that we are given n players and some adversary structure $\mathcal{F}$.

Our definitions are based on the ones of [Can00].

2.1.1 A real life model

An m-phase on-line protocol (where $m \in \mathbb{N} \cup \{\infty\}$) is a collection of n parties $\mathrm{P}_1, \ldots, \mathrm{P}_n$, each being an interactive randomized Turing machine taking some random input r_i. The parties execute m consecutive phases in a synchronous way (i.e. each phase is divided into a number of rounds executed synchronously). More precisely, at the beginning of every phase each party expects to receive a message (later we will specify were this message comes from - see Step (3a)) containing its input for this phase. Then some code is executed (probably involving interaction with other parties) and at a specified round the party answers with a message containing the output of this phase.[1]

A *real life execution of an m-phase on-line protocol* $\mathrm{P}_1, \ldots, \mathrm{P}_m$ is defined as follows. We are given two interactive computationally-unbounded Turing machines: an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. The environment takes an auxiliary input z and a random input $r_{\mathcal{Z}}$, the adversary a random input $r_{\mathcal{A}}$. The execution proceeds according to the following scheme.

1. The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ are initialized. The inputs z and $r_{\mathcal{Z}}$ is passed to $\mathcal{Z}$ and the random input $r_{\mathcal{A}}$ is passed to $\mathcal{A}$.

2. Each player P_i is started, the random input r_i is passed to him.

3. The execution proceeds in m phases. The jth phase goes as follows.

   (a) The environment $\mathcal{Z}$ sends to every P_i a message containing a *jth phase input value* $x_i^j \in \{0,1\}^*$ and passes some auxiliary message to $\mathcal{A}$.

   (b) The computation is performed in a way identical to the case of the secure function evaluation (see [Can00] p. 174). Recall that the parties operate in synchronous rounds. At the beginning of each round the parties produce the messages that they want to send in this round. Then the adversary can execute a number of *mini-rounds*. Each mini-round goes as follows.

      i. The adversary may choose a party (say P') to corrupt (as long as the set of corrupted parties remains a member of the adversary structure $\mathcal{F}$). He learns the complete internal history (the

[1] Clearly a corrupted party may not answer at all, but since the model is synchronous we can safely assume that this counts as sending an empty output.

random input and all the messages ever received) of P′. Moreover the environment $\mathcal{Z}$ learns the identity of P′ and sends some auxiliary information to $\mathcal{A}$.

ii. $\mathcal{A}$ may activate a party (say P″). In this case P″ sends the all the messages it prepared for this round. $\mathcal{A}$ learns the messages sent to the corrupted parties.

The mini-rounds are iterated until there happened a round in which the adversary took no action (i.e. he neither corrupted nor activated any party).

(c) At the end of the computation every player P_i sends to the environment a message containing a *jth phase output value* $y_i^j \in \{0,1\}^*$. The adversary also produces some output and sends it to the environment.

(d) The post execution corruption for the ith phase begins. The environment $\mathcal{Z}$ interacts with the adversary in rounds. In each round $\mathcal{Z}$ sends a message `corrupt` P_i (for some $i \in \{1,\ldots,n\}$). The adversary answers with some message. $\mathcal{A}$ is allowed to corrupt some parties (as long as the set of the corrupted parties belongs to the adversary structure) and learn their internal data.

4. Finally $\mathcal{Z}$ outputs the *real-life output of the environment*. Wlog we can assume that it is just the complete history of the execution of $\mathcal{Z}$. Denote it by $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r_1,\ldots,r_n,r_{\mathcal{A}},r_{\mathcal{Z}})$.

Note, that if we assume that $r_1,\ldots,r_n,r_{\mathcal{A}}$ and $r_{\mathcal{Z}}$ are chosen randomly, we can view the real-life output of the environment as a random variable. Denote it $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$.

The reader may observe a following difference between our definition of environment and the original one [Can00]. In our setting the inputs are chosen *dynamically* by the environment whereas in case of a secure function evaluation [Can00] the inputs are chosen in advance (and thus the real-life output has one more parameter - namely the inputs to the players $\vec{x}$). Later (Section 2.1.4), we are going to comment more on this. We will also explain there the role of the post-execution corruption.

2.1.2 An ideal model

In the ideal scenario we are again given an environment $\mathcal{Z}$ defined as in Section 2.1.1. Recall that n is a number of players. An m-phase on-line protocol is specified by a sequence of n-party functions $\mathrm{F} = \{f^j\}_{j=1}^m$, where for every i

$$f^j : \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \times \{0,1\}^* \to (\{0,1\}^*)^n \times \{0,1\}^*. \qquad (2.1)$$

Let us comment a bit on the domain and range of each f^j. An input of f^j is a tuple $(k,(x_1^j,\ldots,x_n^j),X^j,r^j)$, where k is a security parameter, each x_i^j is an *input of* P_i, and r^j is a *random input*. The value of X^j is called a *trusted party input*. An output of f^j is a tuple of a form $((y_1^j, \ldots, y_n^j), Y^j)$. Each y_i^j is an *output of* P_i. The value of Y^j is called a *trusted party output*. The use of X^j and Y^j will become clear later.

We assume an existence of an interactive Turing machine $\mathcal{S}$ called a *simulator* with a random input $r_\mathcal{S}$ and a security parameter k. We are also given a *trusted party* T which is an interactive Turing machine. The execution goes as follows.

1. The environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ are initialized. The auxiliary input z and the random input $r_\mathcal{Z}$ are passed to $\mathcal{Z}$. The random input $r_\mathcal{S}$ is passed to $\mathcal{S}$.

2. The trusted party will have a sequence of internal variables $Q^0, \ldots, Q^m$, each $Q^j \in \{0,1\}^*$. He sets $Q^0 := \epsilon$. The values of the remaining Q^j will be set during the execution.

3. Then execution proceeds in m phases. Each jth phase is executed in stages similar to the ones in the function evaluation case ([Can00], p. 177).

   The input sending stage: The environment $\mathcal{Z}$ sends a message to each P_i containing the *jth phase input value* $x_i^j \in \{0,1\}^*$. The simulator learns the values received by the corrupted players.

   First corruption stage: The environment sends some auxiliary message to $\mathcal{S}$. Then $\mathcal{S}$ can corrupt the parties in the adaptive way. More precisely the corruptions are done in iterations. Once he corrupts a party he learns all her inputs and outputs from the previous phases plus her input to the jth phase. Moreover he informs the $\mathcal{Z}$ which party he corrupted and $\mathcal{Z}$ answers with some auxiliary message. Depending on the information received this way adversary may decide to start a new iteration.

   Computation stage: The following values are sent to T: every honest party P_i sends her input value x_i^j, every corrupted party sends a value chosen by the simulator $\mathcal{S}$. Let $\vec{x}'$ be a vector of values that T received. Now T chooses a random input r^j and computes the value of $f^j(k, \vec{x}', Q^{j-1}, r^j)$. Let $((y_1^j, \ldots, y_n^j), Y^j)$ be the result. The simulator sends to each P_i the value y_i^j and sets $Q^j := Y^j$.

   Second corruption stage: The simulator learns the values sent in the previous stage from T to the corrupted parties. He can then corrupt some more parties and learn the values they received. He does it in the similar way as the first corruption phase. The only difference is that after corrupting P_i he learns additionally the output value y_i^j.

   Output: Each uncorrupted party outputs y_i^j. Corrupted parties output $\perp$. The environment learns all the values output by the parties (note: he does not learn the value of Q^j). Moreover $\mathcal{S}$ produces some output and sends it to $\mathcal{Z}$.

Post-execution corruption: This phase looks similar to the post-execution corruption in the real-life model. The environment may send adaptively chosen requests to the simulator to corrupt a party. In order to reply to such a request the simulator can corrupt some parties and learn their input and output (as it is in the second corruption stage).

4. As in the real-life model, the environment $\mathcal{Z}$ produces an output (we can assume that it is just a complete history of the execution of $\mathcal{Z}$). Call it an *ideal output of the environment* $\text{EXEC}_{\pi,\mathcal{S},\mathcal{Z}}(k, z, r^1, \ldots, r^m, r_{\mathcal{S}}, r_{\mathcal{Z}})$.

As in the real-life execution, by choosing randomly the random inputs we get a random variable $\text{EXEC}_{\pi,\mathcal{S},\mathcal{Z}}(k, z)$. Figure 2.1. may be helpful in understanding how the Q_i's are used. Recall that $Q^0 = \epsilon$. Clearly we can also assume that $Q^m = \epsilon$. Therefore in all the practical applications we will usually forget about Q^0 and Q^m.

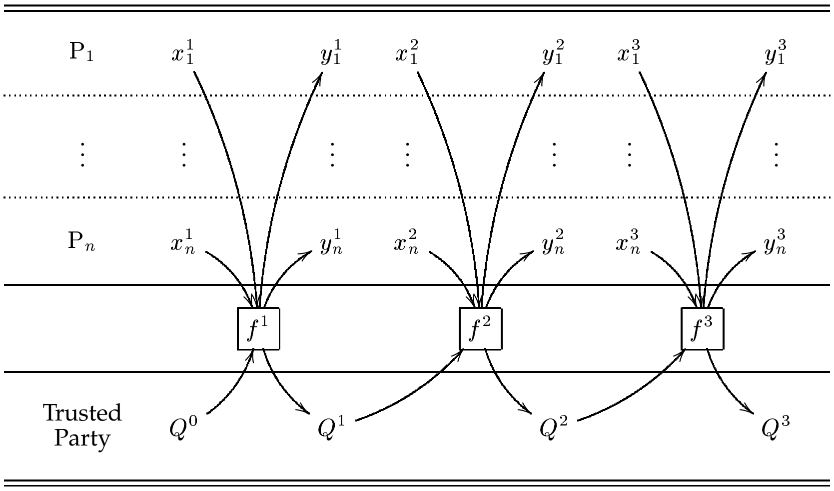

Figure 2.1: A scheme of an ideal execution of a 3-phase protocol. For sake of simplicity we have omitted the random inputs and the security parameter.

2.1.3 Comparing two models

Having introduced the real life model and the ideal model we can now define the notion of the security of an on-line protocol. Intuitively we will require that "from the point of view" of the environment the real life execution should "look the same" as the ideal execution. Formally the definition goes as follows.

Definition 2.1 (Secure evaluation - the zero error case)
Let $\mathrm{F} = \{f^j\}_{j=1}^m$ be a sequence of n-party functions and let π be an m-phase on-line protocol for n parties. We say that π *adaptively $\mathcal{F}$-securely evaluates* F if for any adaptive $\mathcal{F}$-limited adversary $\mathcal{A}$ and any environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for any auxiliary input z and any security parameter k we have

$$\mathrm{IDEAL}_{\mathrm{F},\mathcal{S},\mathcal{Z}}(k,z) \stackrel{\mathrm{d}}{=} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$$

(where $\stackrel{\mathrm{d}}{=}$ denotes equality of the distributions).

A slightly weaker version of security requires the protocol to be secure except some error probability.

Definition 2.2 (Secure evaluation - the non-zero error case)
Let $\mathrm{F} = \{f^j\}_{j=1}^m$ be a sequence of n-party functions and let π be an m-phase on-line protocol for n parties. Let ϵ be some function from $\mathbb{N}$ to $[0,1]$. We say that π *adaptively $\mathcal{F}$-securely evaluates* F *with an error* ϵ if for any adaptive $\mathcal{F}$-limited adversary $\mathcal{A}$ and any environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for any security parameter k there exist random events E_1 and E_2 such that for any auxiliary input z we have

$$P_{\mathrm{IDEAL}_{\mathrm{F},\mathcal{S},\mathcal{Z}}(k,z)|E_1} \stackrel{\mathrm{d}}{=} P_{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)|E_2} \tag{2.2}$$

(where $P_{X|E}$ denotes a distribution of a random variable X conditioned by the event E) and for $i = 1,2$ we have $Pr[E_i] \geq 1-\epsilon$.

An alternative approach (see [Can00]) is to require the statistical distance between $\mathrm{IDEAL}_{\mathrm{F},\mathcal{S},\mathcal{Z}}(k,z)$ and $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ to be smaller than ϵ (instead of requiring (2.2). It is not difficult to see that this approach is more general than ours. It will not matter however in our applications. Observe also that in the above definitions we do not make any assumptions about the computational power of the simulator (see Section 2.2.3. for a short discussion of an alternative approach). The following definition will be useful.

Definition 2.3
A function $\epsilon : \mathbb{N} \to [0,1]$ is *negligible* if for every c there exists n such that

$$\text{for every } i > n \quad \epsilon(i) < \frac{1}{i^c}.$$

2.1.4 The role of the post-execution corruption

As remarked before a slightly alternative approach is to quantify also over all possible inputs $\vec{x}$ for the players in all possible phases (as it is in case of the secure function evaluation of [Can00]), instead of asking $\mathcal{Z}$ to produce it dynamically (possibly after getting some extra information during the post-execution corruption). It is easy to see that this in the zero error case this leads to a definition equivalent to this one (since the auxiliary input z for the environment can contain the inputs for the players).

In the non-zero error case the situation is different. Here the dynamic choice of inputs and the post-execution corruption are crucial for the security. Consider the following simple example. We have $2n + 1$ players: a dealer D and n players $\mathrm{P}_1, \ldots, \mathrm{P}_{2n}$. Player D is incorruptible and the adversary can corrupt at most n player out of $\mathrm{P}_1, \ldots, \mathrm{P}_{2n}$. The players execute a protocol π consisting of two phases specified by functions f^1 and f^2. Function f^1 is defined as follows.

- The input every player is empty.
- The output of D is a randomly chosen set $X \subset \{1, \ldots, 2n\}$ such that $|X| = n$. The output of every other player and the trusted party is empty.

Function f^2 is defined as follows.

- The input of D is a string $t \in \{0, 1\}^k$. The input of the trusted party and all the other players is empty.
- The output of D is a bit $b \in \{0, 1\}$ equal to 0 for every value of input t. The output of every other player is empty.

Consider now the following protocol.

Phase 1

1. Player D chooses randomly a set $X \subset \{1, \ldots, 2n\}$ such that $|X| = n$.
2. Player D chooses randomly a string $s = \{0, 1\}^n$. He shares s among the players in $\mathcal{X} = \{ \mathrm{P}_i : i \in X \}$. The sharing is very simple:
   (a) he chooses n random strings $\{s_i\}_{i \in X}$ such that a componentwise XOR of all the s_i's results in s, and
   (b) he sends each s_i to P_i.
3. Player D outputs X.

Phase 2

1. Player D inputs t. If $t \neq s$ then he outputs 0. Otherwise he outputs 1.

Now, if we do not allow the post-execution corruption, then it is easy to see that the protocol works securely unless one of the following happened:

1. the adversary guessed X at the end of the first phase — in this case (1) he corrupts the players in $\mathcal{X}$ and learns s (2) he sends s to the environment (3) the environment chooses $t = s$ to be the input of D in Phase 1 and D outputs 1,
2. the environment guessed s and sent it as an input to the dealer in Phase 2 — clearly then D outputs 1.

Clearly both case 1. and 2. happen with probability at most 2^{-n}. Thus the protocol works securely with an error probability negligible in n.

On the other hand, if we allow the post-execution corruption, then the following attack is possible.

1. The environment learns X, the output of D in Phase 1.

2. In the post-execution corruption the environment asks the adversary to corrupt the players in $\mathcal{X}$. In this way it learns s.

3. The environment chooses $t = s$ as an input of D. Thus D outputs 1.

Therefore the protocol is insecure.

2.1.5 The hybrid model

In this section we are going to define the *hybrid model*. Informally speaking it is a real life model in which at some points the protocol is allowed to call a trusted party to perform some computations. The idea is similar to the one of [Can00] and will allow us to formally define the use of subprotocols as subroutines. For simplicity we will focus on the scenario when only one (multi-phase) subprotocol is executed. The idea generalizes easily to the case of many subprotocols (see Section 2.1.8).

Suppose we are given a sequence of functions $\mathrm{F} = \{f^j\}_{j=1}^m$. An F-*hybrid* on-line protocol π is defined similarly to a real life on-line protocol with a following difference. At a certain point of the protocol π parties may call the trusted party T to compute the function f^1, later on they may ask T to compute f^2, and so on. Technically it looks as follows. The trusted party will have to remember the sequence of internal variables $Q^0, \ldots, Q^m$. Moreover the trusted party has a variable $\mathtt{j}$ determining the index of the next function f^j to be computed. At the beginning it sets $\mathtt{j} := 1; Q^0 := \epsilon$ and leaves remaining Q^j's undefined.

At a certain round (say round number $l_{\mathtt{j}}$) of π every uncorrupted party P_i writes his input value $x_i^{\mathtt{j}}$ in some special register. Then the adversary may corrupt the players in mini-rounds as described in Step (3b) in Section 2.1.1 (and learn the $x_i^{\mathtt{j}}$ values of the corrupted players). In the round $l_{\mathtt{j}} + 1$ every player P_i sends x_i^j to the trusted party (the corrupted players send some values determined by the adversary). The trusted party computes the value of $f^{\mathtt{j}}(k, (x_1^{\mathtt{j}}, \ldots, x_n^{\mathtt{j}}), Q^{\mathtt{j}}, r^{\mathtt{j}})$. Let $((y_1^{\mathtt{j}}, \ldots, y_n^{\mathtt{j}}), Y^{\mathtt{j}})$ be the result. For every $i = 1, \ldots, n$ the trusted party sends $x_i^{\mathtt{j}}$ to P_i (if P_i is corrupted then the value sent to him becomes known to the adversary). Moreover the trusted party sets $Q^{\mathtt{j}} := Y^{\mathtt{j}}$ and increases $\mathtt{j}$ by 1.

The output of the environment $\mathcal{Z}$ after an execution of an F-hybrid on-line protocol π (against an adversary $\mathcal{A}$, with an auxiliary input z and a security parameter k) is a random variable denoted by $\mathrm{EXEC}^{\mathrm{F}}_{\pi,\mathcal{A},\mathcal{Z}}(z, k)$.

2.1.6 Secure composition - zero error case

Having a F-hybrid on-line protocol π and a protocol σ that securely evaluates F we can construct a composed protocol π^σ, by replacing the call to the trusted party with the execution of σ. This is done in a straightforward way. See [Can00] (p. 163) for details. Later we will refer to π as a *sub-protocol* of a *super-protocol* σ. We would like to stress that we consider only *non-concurrent* compositions here, i.e. when the sub-protocol is executed the execution of the super-protocol is suspended.

Definition 2.4
Let F and G be sequences of n-party functions and let π be an on-line protocol in a F-hybrid model. We say that π *adaptively $\mathcal{F}$-securely evaluates* G *in the* F-*hybrid model* if for any adaptive $\mathcal{F}$-limited adversary $\mathcal{A}$ (in the F-hybrid model) and any environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for any z and k

$$\text{IDEAL}_{\mathrm{G},\mathcal{S},\mathcal{Z}}(k,z) \stackrel{\mathrm{d}}{=} \text{EXEC}^{\mathrm{F}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z). \tag{2.3}$$

The main theorem in this section is the following.

Theorem 2.5 (Secure Composition of On-Line Protocols)
Let $\mathcal{F}$ be an adversary structure over the set of players $\mathrm{P}_1,\ldots,\mathrm{P}_n$. Let π be an n-party on-line protocol adaptively $\mathcal{F}$-securely evaluating G *in an* F*-hybrid model. Let σ be an on-line protocol that adaptively $\mathcal{F}$-securely evaluates* F*. Then the protocol π^σ adaptively $\mathcal{F}$-securely evaluates* G*.*

Theorem 2.5 easily follows from the following claim.

Theorem 2.6 (Modular Composition of the On-Line Protocols)
Let $\mathcal{F}$ be an adversary structure over the set of players $\mathrm{P}_1,\ldots,\mathrm{P}_n$. Let π be an n-party on-line protocol in the F *hybrid model. Let σ be an n party on-line protocol that adaptively $\mathcal{F}$-securely evaluates* F*. Then, for any $\mathcal{F}$-limited real-life adversary $\mathcal{A}$ and for any environment machine $\mathcal{Z}$ there exists an adversary $\mathcal{S}$ in the* F*-hybrid model such that for every k and z we have*

$$\text{EXEC}^{\mathrm{F}}_{\pi,\mathcal{S},\mathcal{Z}}(k,z) \stackrel{\mathrm{d}}{=} \text{EXEC}_{\pi^\sigma,\mathcal{A},\mathcal{Z}}(k,z). \tag{2.4}$$

The proof of this theorem is a modification of a proof of Theorem 10 in [Can00] (page 184.). We will not present it here.

2.1.7 Secure composition - non-zero error case

In this section we discuss the secure composition of the protocol when the security holds with some error probability. Suppose σ is an n party on-line protocol that adaptively $\mathcal{F}$-securely evaluates some F with an error ϵ. Then Theorem 2.6 still holds, with a following difference. Instead of having (for every k and

z) the equation (2.4) we will have that for every k there exists an event E such that for every z the following holds

$$\mathrm{EXEC}^{\mathrm{F}}_{\pi,\mathcal{S},\mathcal{Z}}(k,z) \stackrel{\mathrm{d}}{=} [\mathrm{EXEC}_{\pi^{\sigma},\mathcal{A},\mathcal{Z}}(k,z)|E].$$

It is easy to see that this implies Theorem 2.5 with a following modification. If π is a protocol in F-hybrid model securely evaluating G with an error δ, then the protocol π^{σ} securely evaluates G with an error at most $\delta + \epsilon$.

2.1.8 Protocols with many subprotocols

In many cases we will construct protocols in a hybrid model, where trusted parties for more than one sequence of functions are given. An $(\mathrm{F}_1, \ldots, \mathrm{F}_t)$-hybrid model and a composition $\pi^{\sigma_1,\ldots,\sigma_t}$ are defined in a straightforward way. It is also clear that in the zero error case the Theorems 2.6 and 2.5 generalize easily. In the non-zero error case the error propagates in the following way. Suppose for each i a protocol σ_i securely evaluates F_i with an error ϵ_i. Let π be a protocol in $(\mathrm{F}_1, \ldots, \mathrm{F}_t)$-hybrid model securely evaluating G with an error δ. Then $\pi^{\sigma_1,\ldots,\sigma_t}$ securely evaluates G with an error $\delta + \sum_{i=1}^{t} \epsilon_i$.

2.1.9 Making the definition more specific

In this section we are going to present some conventions and modifications of the definitions form the previous sections. The aim is to make those definitions easier to use in the specific cases that we will encounter later.

The domains of the inputs

First of all recall that in the original definitions we set the inputs for the protocol (and for the trusted party) to be the elements of $\{0,1\}^*$. In practical cases we will usually assume that an input of each player P_i (in phase p, say) comes from some specified domain $\mathcal{D}_i^p$ (and then is encoded as a word in $\{0,1\}^*$). Note, that a corrupted player may submit as an input some word $w \in \{0,1\}^*$ that is not a valid encoding of any element in $\mathcal{D}_i^p$. He can also not send anything. In both cases we will assume that this counts like sending some default value. A similar convention applies to the messages exchanged between the players.

Lists

In case of the protocols that do not have a fixed number of rounds we will usually assume that the input and output of the trusted party carry some information about the history of the execution. Therefore the domain of this input and output will be a *list* of the elements from some fixed domain $\mathcal{D}$. We will use following terminology. A *list of length l (of elements of $\mathcal{D}$)* is a sequence $t_1 \cdots t_l$ such that each $l_i \in \mathcal{D}$. The set of all lists of elements of $\mathcal{D}$ will be denoted $\mathcal{D}^*$. If $L = t_1 \cdots t_l$ is a list then *L with t_{l+1} attached to it* (denoted $L \cdot t_{l+1}$) is a list

$t_1 \cdots t_l \cdot t_{l+1}$. We will also say that t_i *is the* i*th element of* L. An *empty list* ϵ is a list of length 0.

We will usually assume that the trusted party input for the pth phase is a list L of length $p-1$.[2]. The trusted party output for this phase will be the list L with some new element attached.

Assumptions about the super-protocol

Observe that in the previous sections, we required security for the environment sending arbitrary inputs to the players (in Step (3a) in the real life execution, and in the input sending stage of the ideal one). It turns out that in case of many protocols this requirement is too strong. More precisely it can be the case that a protocol π is not secure according to Definition 2.1, but it can be still safely used as a subprotocol of certain protocols. Suppose for example that π is a two-phase protocol. Let P_i be one of the players and let x^1 and x^2 be his inputs for phase 1 and 2 respectively. Suppose we know that each time we use π as a subprotocol (of some protocol σ, say) it will be the case that x^1 is equal to x^2 if P_i remains honest (until the round in which he sends x^2 to the trusted party). Let $\Sigma_{x^1=x^2}$ be the class of all such protocols. It is clear that in this case it will be enough to require a weaker version of security. Namely, we can modify the Definition 2.1 by restricting ourselves of the environments $\mathcal{Z}$ such that the input sent to P_i in phase 1 is equal to the one sent in phase 2. Let $\mathbf{Z}_{x^1=x^2}$ be the class of all such environments. Now, if we

1. specify π with some pair F of functions f^1, f^2,
2. prove the security of π restricting ourselves to the environments from $\mathcal{Z}_{x^1=x^2}$,
3. show that $\sigma \in \Sigma_{x^1=x^2}$ and prove its security in the F-hybrid model,

then we can compose σ with π as in Theorem 2.6.

This generalizes in a straightforward way to arbitrary conditions in which an input x^j (of a player P_i in phase j), depends on the inputs and outputs of player P_i in the previous phases (in some protocol π). Quite often this condition will not depend on the previous inputs and outputs at all. In this case we will simply have some two domains $\mathcal{D}$ and $\mathcal{D}'$, such that $\mathcal{D} \subset \mathcal{D}'$. If a player P_i remains honest then his input is guaranteed to come from $\mathcal{D}$. If P_i gets corrupted he will be allowed to choose from a bigger domain $\mathcal{D}'$. Intuitively this will mean that using the inputs from $\mathcal{D}' \setminus \mathcal{D}$ we model the extent of the damage that the corrupted P_i can do the the output of the function.

A dynamical choice of a function

We will also use a following convention. Observe that in the original definition of the ideal process (Section 2.1.2) the function f^j that the trusted party computes in phase j is fixed in advance. It turns out that many practical cases it is

[2]Thus we will assume that the input for the first phase is an empty list

much more convenient to allow the function to be chosen dynamically by the super-protocol.

Practically we implement is as follows. Suppose we want to be able to choose in phase p between functions $g_1, \ldots, g_a$. We will require each input x_i^j for a player P_i to be a pair from a set $\{1, \ldots, a\} \times \{0,1\}^*$. Suppose a super-protocol π wants the trusted party to compute g_l. Then π will instruct each player P_i to submit to the trusted party a pair (l, w_i^p). Therefore, when defining the security of σ we will assume that the inputs that the environment $\mathcal{Z}$ chooses for the players are pairs with an identical first component l. For inputs $\{(l_i, w_i^p)\}_{i=1}^n$ the value of the function f^p is now defined in the following way. Let l' be an index such that there exists a set $X \subseteq \mathcal{P}$ of players with the following properties:

- X is not in the adversary structure and
- for each $\mathrm{P}_i \in X$ we have $l_i = l'$.

(Clearly if the adversary structure is $\mathcal{Q}_2$ then there exists exactly one such l'.) Then the output of f^p is defined to be equal to the output of $g_{l'}$ on the values w_i^p.

In practical cases we will abstract from those technicalities and simply assume that the function g_l is chosen dynamically. We will do it both when we prove the security of the protocol and when we use σ as a subprotocol. Formally speaking in the second case we will have to make sure that all the honest players agree on which g_l they have chosen. This will usually follow in a straightforward way from a synchronous design of the protocol.

Finally, observe that if at least one player is corrupted that the simulator knows which function g^l is chosen.

2.2 The standard simulation

In the next sections we are going to specify and implement several protocols and prove their security. The main part of every such proof will be a construction (for every adversary $\mathcal{A}$) a simulator $\mathcal{S}$. The simulator that we are going to construct will have a special form which we will call a *standard form*. In this section we are going to sketch it. This will allow us to stay on a more informal level in the next sections. We believe that this makes the proofs more readable and more convincing, since we will not need to care about the technical details, and instead we will be able to concentrate on the real reasons why our protocols are secure.

We will assume that we are given some adversary structure $\mathcal{F}$. Every adversary that we consider here will be adaptive and $\mathcal{F}$-limited. Let us also assume that we are given an m phase protocol π specified by a sequence of functions $\mathrm{F} = \{f^j\}_{j=1}^m$ (see (2.1)). The functions that we will use in our specifications will always be deterministic. Therefore we will drop the random input and the security parameter and assume that each function f^j takes as an input a tuple $((x_1^j, \ldots, x_n^j), X^j) \in (\{0,1\}^*)^n \times (\{0,1\}^*)$ (compare it with (2.1)).

Take some environment $\mathcal{Z}$ and adversary $\mathcal{A}$ some random inputs $r_1, \ldots, r_n$, $r_{\mathcal{Z}}, r_{\mathcal{A}}$ and an input z. Let a *history of the (real life) execution* of a protocol π against $\mathcal{A}$ with an environment $\mathcal{Z}$ be the sequence containing the internal history of every player and the adversary (including all the messages sent and received) in some execution of π (against $\mathcal{Z}$ and $\mathcal{A}$). For a given mini-round r a *history of the execution until* r is defined to be a prefix of the history of the execution taken until r was completed. Let $\mathsf{HIST}(\pi, \mathcal{Z}, \mathcal{A}, z)$ denote the probability space containing the histories of the execution of π against $\mathcal{A}$ with environment $\mathcal{Z}$ and an auxiliary input z (when $r_1, \ldots, r_n, r_{\mathcal{Z}}, r_{\mathcal{A}}$ are chosen randomly). Similarly let $\mathsf{HIST}(\pi, \mathcal{Z}, \mathcal{A}, z, r)$ denote the probability space of the histories of the execution of π until mini-round r.

Let $\mathsf{hist}_{\mathcal{Z},\mathcal{A},r}$ be a function assigning to each history $H \in \mathsf{HIST}(\pi, \mathcal{Z}, \mathcal{A}, r)$ the set of messages exchanged during H between the adversary $\mathcal{A}$ (on one side) and $\mathcal{Z}$ and the corrupted players (on the other side). A value of $\mathsf{hist}_{\mathcal{Z},\mathcal{A},r}(H)$ will be called *the $\mathcal{A}$-view of history H (until mini-round r)*.

2.2.1 The case with zero error probability

We are going to construct a *standard simulator* $\mathcal{S}(\cdot)$ that simulates each adversary $\mathcal{A}$ using it as a black-box. Let $\mathcal{Z}$ be the environment. First, $\mathcal{S}(\mathcal{A})$ creates his own copies of the adversary and every party in the protocol. As long as a party remains honest $\mathcal{S}$ will simulate its steps. The main problem during the simulation is that the simulator does not know the values of the inputs that the environment sends to the honest players at the beginning of each phase. Therefore he will choose those inputs in the arbitrary way (up to some constraints described later). In the sequel we will call those values the *simulated inputs*. The input values sent by the environment to the players in the ideal model will be called the *real inputs*.

Observe that during the simulation the simulator (and the adversary) can get some information about the real inputs of the players, basing on the inputs and outputs of the corrupted players. Let us make it more precise. Suppose that the simulator is simulating a mini-round r (in some phase l) and until this moment the simulator corrupted some players $\mathrm{P}_{i_1}, \ldots, \mathrm{P}_{i_a}$. Therefore for every phase $j = 1, \ldots, l$ he knows the input values $z_{i_1}^j, \ldots, z_{i_l}^j$ handed to those players in the input sending stage. The same holds for the output values $w_{i_1}^j, \ldots, w_{i_l}^j$ (except of $j = l$, if the simulator has not yet entered the computation stage of phase j). A sequence $\{x_i^j\}_{i=1,\ldots,n,j=1,\ldots,l}$ (where each $x_i^j \in \{0,1\}^*$) will be called an *admissible sequence of inputs* if it could be the case that each x_i^j is an input of player P_i in phase j. More precisely the following has to be satisfied.

- For each corrupted player P_{i_b} and each phase $j = 1, \ldots, l$ it is the case that $x_{i_b}^j = z_{i_b}^j$.

- There exists a sequence of trusted party outputs $T^0, \ldots, T^l$ (where $T^0 = \epsilon$) and for each phase $j = 1, \ldots, l$ (except of phase $j = l$ if the simulator has not yet entered the computation stage) there exists a sequence $y_1^j, \ldots, y_n^j$ such that for each corrupted player P_{i_b} it is the case that $y_{i_b}^j = w_{i_b}^j$ and
$$f^j((x_1^j, \ldots, x_n^j), T^{j-1}) = ((y_1^j, \ldots, y_n^j), T^j).$$

The simulator will also simulate the execution of the trusted party computing the functions from F. In order to do it we will use internal variables $Q^0, \ldots, Q^{m-1}$ for storing the output of the trusted party. Call them the *simulated outputs of the trusted party*. In fact this will not be a part of a real simulation. It will be useful however for proving the correctness of the simulation.

As a side effect of each such simulation we will obtain some real life execution of the protocol π (with interaction with $\mathcal{Z}$ and $\mathcal{A}$). We will call it a *simulated execution* (of π with $\mathcal{Z}$ and $\mathcal{A}$). The simulation is constructed in such a way that $\mathcal{S}(\mathcal{A})$ will always have a complete information about the $\mathcal{A}$-view of the history of the simulated execution (until the current mini-round).

Let us now describe the simulation step by step. First $\mathcal{S}(\mathcal{A})$ needs to do all the necessary bureaucracy: it sends a random input to every party, and it starts the adversary. He also sets $Q^0 := \epsilon$. The simulation of each phase p is performed in the following way.

Simulation of phase p

1. The simulator sends to the simulated $\mathcal{A}$ the input values of the corrupted players (that he learns from $\mathcal{Z}$ in the input sending stage).

2. Acting as an environment the simulator guesses for each simulated player P_i (that remains honest) an arbitrary input value x_i^p from some domain $\mathcal{D}$. Usually $\mathcal{D}$ will just be the domain of the inputs of player P_i for phase p. In some cases however he will in advance know (basing on the inputs and outputs of the corrupted players) some constraints on x_i^p, and thus he will choose it from some subset $\mathcal{D}'$ of $\mathcal{D}$. This happens when we made some assumptions about the behavior of the honest players between the phases (see Section 2.1.9), or if we assume that a type of a phase is a part of the input (see Section 2.6). In both cases it will be clear how $\mathcal{D}'$ is defined.

3. The simulator goes to the first corruption stage. He forwards to $\mathcal{A}$ an auxiliary message that he gets from the environment.

4. The simulator simulates the execution of the protocol in a straightforward way. At the beginning he stays in the first corruption stage. Suppose the simulated protocol is in some mini-round r and the adversary requests to corrupt a player P_i. Then, the simulator corrupts it in the ideal model. In this way he learns all the real inputs and the outputs of

P_i from the previous phases and an input for this phase. This gives the simulator information about the admissible inputs of the players (for the phases $1, \ldots, p$). It usually turns out that the simulated inputs given by the simulator to the players so far are not admissible anymore (in particular the simulated inputs of the player P_i will not be equal to the real inputs). In this case the simulator has to change the simulated inputs to the admissible ones, and find such an execution of the protocol that the $\mathcal{A}$-view of the history is the same as in the original execution. He does it be executing a following procedure (let h be the $\mathcal{A}$-view of the history of the simulated execution until mini-round $r-1$)

$\boxed{\mathrm{ReRun}}$

(a) The simulator chooses an arbitrary admissible sequence of inputs $\mathbf{x} = \{x_i^j\}_{i=1,\ldots,n, j=1,\ldots,p}$.

(b) He restarts the parties and simulates the protocol from the beginning with the simulated inputs for each phase determined by $\mathbf{x}$ (and with some fresh random inputs). He also restarts the adversary (the random input can remain the same). The only differences are that (1) the simulator will not send any messages to the trusted party (2) instead of sending a message to $\mathcal{Z}$ he will simply compare it with a message sent in the original execution (which he reads from h). If it is not equal then he halts and starts Step (4b) again. Otherwise he continues the simulation. Each time he expects to get a message from $\mathcal{Z}$ or the trusted party the simulator will use the corresponding message extracted from h.

This goes as long as the mini-round r is reached. If this procedure does not terminate (clearly a computationally unbounded simulator can detect it in a finite time), then the simulator halts.

Then, the adversary sends all the internal history of P_i to the adversary, informs the environment $\mathcal{Z}$ about the fact that P_i was corrupted. The environment replies with some auxiliary data, which $\mathcal{S}(\mathcal{A})$ forwards to $\mathcal{A}$. Since now the adversary is given a full control over the behavior of P_i. The simulation continues (with the copies of the parties and the values of Y^j's from ReRun replacing the old ones)

5. At some point the simulator will go to the computational stage. Each time when we construct the simulator for a particular protocol we will specify the round in which he does it. We will also show how the simulator determines the values that he chooses to be the inputs of the corrupted players (that he sends to the trusted party). He will do it basing only on the $\mathcal{A}$-view of the current history of the simulated execution.

   Let $z_1^p, \ldots, z_n^p$ be defined as follows:

- for each P_i that got corrupted, set z_i^p to be the value that the simulator has sent to the trusted party as the input of P_i (for the current phase), and
- for each P_i that remained honest set z_i^p to be the simulated input of player P_i (for the current phase).

Now set $((w_1^p, \ldots, w_n^p), Y^{p+1}) := f((z_1^p, \ldots, z_n^p), Y^p)$.

6. As a result of the previous step the simulator learns the outputs of the corrupted players in the ideal model. Therefore the inputs that he sent to the players so far, may not be admissible anymore. Thus he has to make an update of the internal state of the parties, by the ReRun protocol as shown in Step 4.

7. The simulator goes to the second corruption stage and continues the simulation exactly as he did it in Step 4. He does it until the protocol for this phase halts. The adversary produces some output. The simulator forwards it to the environment.

   Now for each player P_i that remains honest we check whether w_i^p (defined in Step 5.) is equal to the value output by P_i in the simulated protocol. If it is not the case then we say that the simulated execution is *incorrect*.

8. Afterwards, the post-corruption stage is performed. This is done in a straightforward way. Each request from the environment to corrupt some player P_i is forwarded by $\mathcal{S}(\mathcal{A})$ to the adversary. If the adversary corrupts some player then the simulator behaves as in Step 4. The message output by the adversary is sent back to the environment. Since the functions in F are assumed to be deterministic every post-execution corruption will be trivial, and we will not focus on it.

We will often say that a player *got corrupted* if the adversary requested to corrupt him. Observe that each time an adversary makes such a request, the simulator will immediately corrupt the same party in the ideal model.

Note, that $\mathcal{S}(\mathcal{A})$ is not a well defined simulator, since we did not fix any particular way the simulated inputs are chosen in Step 2. of the simulation and in Step 6. of the ReRun. Therefore we will assume that $\mathcal{S}$ takes one more parameter, namely an auxiliary string α, that determines that choice, and we will write $\mathcal{S}(\mathcal{A}, \alpha)$. More precisely α is a description of a function that, basing on the entire history of the execution of the simulator, produces a sequence of inputs.

Definition 2.7
W will say that a standard simulator $\mathcal{S}(\cdot, \cdot)$ *works correctly* if for each adversary $\mathcal{A}$, each environment $\mathcal{Z}$, each input z for $\mathcal{Z}$, and each α every simulated execution of $\mathcal{S}(\mathcal{A}, \alpha)$ is correct (i.e. it was not detected to be incorrect in Step 7.).

Definition 2.8
We will say that a standard simulator $\mathcal{S}(\cdot,\cdot)$ *works secretly until mini-round* r if for each adversary $\mathcal{A}$, each environment $\mathcal{Z}$ and each input z, the $\mathcal{A}$-views of the histories (of the execution of $\mathcal{S}(\mathcal{A},\alpha)$) until mini-round r are distributed identically for every choice of α.

We will say that a standard simulator $\mathcal{S}(\cdot,\cdot)$ *works secretly* if it does so until the last mini-round of the protocol.

The following lemma is straightforward.

Lemma 2.9 For every standard simulator that works correctly and secretly, at the end of the simulation the simulated inputs are admissible. The same holds for the real inputs.

The main invariant that we will have to prove each time we analyse the standard simulation is that the simulator $\mathcal{S}(\cdot,\cdot)$ works secretly and correctly until each mini-round r. The following lemmas will be useful in proving it.

Lemma 2.10 Let $\mathcal{S}(\cdot,\cdot)$ be a standard simulator. Assume that we know that $\mathcal{S}(\cdot,\cdot)$ works secretly until mini-round r. Suppose that while simulating (for any $\mathcal{A},\mathcal{Z},z$ and α) the mini-round $r+1$ the simulator executed the ReRun procedure. Then with probability 1 he finishes this step in a finite time.

Proof
Let $\mathbf{z}$ be the simulated inputs at round r. Clearly the input values $\mathbf{x}$ that the simulator chooses in Step (4a) were admissible before P_i got corrupted. Therefore the distribution of the histories of adversary is identical for $\mathbf{z}$ and for $\mathbf{x}$. Therefore since the current $\mathcal{A}$-view of the history happened with simulated inputs $\mathbf{z}$ it also has to occur with (some probability p) with simulated inputs $\mathbf{x}$. Thus we are done. □

The idea behind the construction of the ReRun is the following. Suppose we know that the standard simulator $\mathcal{S}(\cdot,\cdot)$ works secretly until some mini-round r. Fix some environment $\mathcal{Z}$ and an input for it. Take some simulation done by $\mathcal{S}(\mathcal{A},\alpha)$. Let h be the $\mathcal{A}$-view of the history of the simulated execution until r. Suppose that in mini-round $r+1$ the simulator corrupted some player P_i, and as a result of this a ReRun procedure was executed. Let $\mathbf{x}$ be the simulated inputs chosen in Step (4a) of ReRun and let H be the resulting history of the execution. Now, take some other auxiliary input β such that the simulated inputs chosen by $\mathcal{S}(\mathcal{A},\beta)$ are taken from $\mathbf{x}$, as long as the $\mathcal{A}$-view of the history is a prefix of h.[3] Take some simulated execution of $\mathcal{S}(\mathcal{A},\beta)$. Let G be a history of it. By the construction of the ReRun procedure it is clear that

1. the distribution of the histories H under the condition that

$$\mathcal{A}\text{-view of } H \text{ until round } r \text{ is equal to } h \tag{2.5}$$

[3]Since the functions computed by the trusted party are deterministic, the inputs in $\mathbf{x}$ remain admissible until mini-round $r+1$, as long as the $\mathcal{A}$-view of the history is a prefix of h.

and

2. the distribution of the histories G under the condition that

$$\mathcal{A}\text{-view of } G \text{ until round } r \text{ is equal to } h \tag{2.6}$$

are equal. By the secrecy of the simulation we know that the probabilities of (2.5) and (2.6) are equal. Therefore we get that the histories H such that (2.5) holds and the histories G such that (2.6) holds, are distributed identically. Thus, if we know that the simulation works secretly until round r, and in some simulated execution in round $r + 1$ a corruption occurs, then we will be able to consider only the cases when the simulator is lucky and there is no need for executing ReRun. The same applies to the Step 6. (where the ReRun is called after the simulator learns the outputs of the trusted party).

Lemma 2.11 Let π be a protocol specified by a sequence of functions F. Let $\mathcal{S}(\cdot,\cdot)$ be a standard simulator for it. Suppose it works correctly and secretly. Then the protocol π securely evaluates F.

Proof
We need to show that for every environment $\mathcal{Z}$, every auxiliary input z to it, and every adversary $\mathcal{A}$ there exists α such that

$$\mathrm{IDEAL}_{\mathrm{F},\mathcal{S}(\mathcal{A},\alpha),\mathcal{Z}}(z) \stackrel{\mathrm{d}}{=} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(z).$$

We will show that for each mini-round r the messages received by environment are distributed in the same way in the simulated execution and in the real one. First, observe that since the simulation works secretly we can assume that the simulated inputs (guessed at the beginning of Step 2.) are always equal to the real ones (as they are both admissible by Lemma 2.9). More precisely we can assume that the input choosing function described by α will always produce the same inputs as the environment $\mathcal{Z}$. Clearly since $\mathcal{Z}$ is deterministic (for fixed z and $r_{\mathcal{Z}}$), and so is the the trusted party, such a function exists.

With this assumption it is easy to see the simulated executions are distributed identically with the real life ones. That is why it is enough to show that the messages received by $\mathcal{Z}$ and $\mathcal{A}$ in a real life execution are identical to the ones received by them in the simulated one. This follows easily from the construction of the simulator and the assumption that protocol works correctly. $\square$

2.2.2 The case with non-zero error probability

The protocols that we consider will usually involve some error probability. Informally speaking the protocol will remain secure unless some error occurs. In this section we will make this notion more precise.

Let $\mathcal{H}_\pi$ be the set of all possible $\mathcal{A}$-views of the history of the execution of π against any $\mathcal{A}$ with any $\mathcal{Z}$ until any mini-round. Let ERROR be some subset of $\mathcal{H}_\pi$. Call it *an error event*. Suppose the protocol is in some mini-round r and the current $\mathcal{A}$-history is h. Then we will say that *an error occurs in mini-round r* if $h \in \mathsf{ERROR}$.

For a given error event ERROR the *error probability* will be a maximal probability (over all $\mathcal{Z}$ and $\mathcal{A}$) that an error occurs in some mini-round of a randomly generated history of the execution.

We modify the standard simulation in the following way. As long as the history of the simulated execution is not in ERROR the simulation goes as in Section 2.2.1. If in some mini-round an error occurs then the simulator sends to the environment some special message "`error`" and the simulation halts. In this case we will say that *the simulation caused an* ERROR. Let $\mathcal{S}(\cdot,\cdot,\mathsf{ERROR})$ denote such a simulator. Observe that since the $\mathcal{A}$-view of the history contains all the messages sent to $\mathcal{Z}$, it will also include the "`error`", message, if the simulation causes an ERROR. Thus if the error occurs during the ReRun procedure, then the procedure is restarted. Also, the definition of the secrecy of the simulation can remain unchanged.

Definition 2.12
We will say that a standard simulator $\mathcal{S}(\cdot,\cdot,\mathsf{ERROR})$ *works secretly until mini-round r unless* ERROR *occurs* if for each adversary $\mathcal{A}$, each environment $\mathcal{Z}$ and each input z, the $\mathcal{A}$-views of the histories (of the execution of $\mathcal{S}(\mathcal{A},\alpha,\mathsf{ERROR})$) until mini-round r are distributed identically for every choice of α.

We will say that a standard simulator $\mathcal{S}(\cdot,\cdot,\mathsf{ERROR})$ *works secretly unless* ERROR *occurs* if it does so until the last mini-round of the protocol.

Note, that this definition is rather restrictive. It covers only the cases when the probability of an error occurring in a given round r is equal for every choice of simulated inputs. This will be the case in all our protocols, although in general there exist secure protocols that do not satisfy this condition.

Definition 2.13
For a given error event ERROR we will say that a simulator $\mathcal{S}(\cdot,\cdot,\mathsf{ERROR})$ *works correctly unless* ERROR *occurs* if for each adversary $\mathcal{A}$, each environment $\mathcal{Z}$, each input z for $\mathcal{Z}$, and each α, every simulated execution of $\mathcal{S}(\mathcal{A},\alpha,\mathsf{ERROR})$ is correct, unless it caused an ERROR.

Clearly Lemmas 2.9, 2.10 also hold in this case (Lemma 2.10) requires the error probability to be smaller than 1). Lemma 2.11 has now the following form.

Lemma 2.14 Let π be a protocol specified by a sequence of functions F. Let ERROR be an error event and let ϵ be the error probability. Let $\mathcal{S}(\cdot,\cdot,\mathsf{ERROR})$ be the standard simulator for π. Suppose it works correctly and secretly unless ERROR occurs. Then the protocol π securely evaluates F with an error ϵ.

Proof
For given $\mathcal{A}$ and $\mathcal{Z}$ let $\mathcal{E}_0$ be the event that during the simulated execution of

$\mathcal{S}(\mathcal{A}, \cdot, \mathsf{ERROR})$ the environment received an `error` message, and let $\mathcal{E}_1$ be the event that the ERROR occurs in the real life execution of π.

Clearly, by the assumption that the simulator works secretly, the probability of $\mathcal{E}_0$ is identical for every choice of the inputs done by $\mathcal{S}(\mathcal{A}, \cdot, \mathsf{ERROR})$. As in the proof of Lemma 2.11 we can assume that the simulated inputs (guessed at the beginning of Step 2) are always equal to the real ones. Therefore (by the correctness condition) for each mini-round r all the $\mathcal{A}$-views of the histories of the simulated executions until r, that do not belong to ERROR are distributed identically with real-life ones that do not belong to ERROR. Also the probability of $\mathcal{E}_0$ is equal to the probability that an error occurs in the real life execution of π (let $\mathcal{E}_1$ denote this event). Thus we get that

$$P_{\mathrm{IDEAL}_{\mathrm{F},\mathcal{S}(\mathcal{A},\cdot,\mathsf{ERROR}),\mathcal{Z}}|\overline{\mathcal{E}_0}} \stackrel{d}{=} P_{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}|\overline{\mathcal{E}_1}}.$$

□

Here $\overline{\mathcal{E}_i}$ denotes the event that $\mathcal{E}_1$ did *not* happen.

In practical applications we will specify ERROR in an informal language. It should always be clear how to translate it into a formally defined subset of $\mathcal{H}_\pi$

Finally, observe that the standard simulation can be extended in a straightforward way to protocols working in a hybrid-model.

2.2.3 Universal security

Another variant of the security definition is to require the simulator to operate in time polynomial in the complexity of the adversary (see for example [Can00]). In [CDD$^+$01] it is called a *universal security*. It is easy to see that the standard simulation does not work if we choose this definition. This is because of the ReRun procedure, that can be iterated a big number of times. Nevertheless it will not be difficult to see that the protocols in Chapters 3 and 5 are universally secure, since the ReRun procedure can be substituted by a more efficient one.

Chapter 3

The Information Checking Protocol

In this chapter we are going to present an information checking protocol (IC), which is a modification of the one introduced by [RBO89]. The protocol works in a model from Section 1.2.2, i.e. we are given a set of n players $\{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$, connected pairwise by private channels, and a broadcast channel. The protocol is secure against an active and adaptive adversary that can corrupt at any set of players from a $\mathcal{Q}_2$ adversary structure $\mathcal{F}$. Two of the players in $\mathcal{P}$ play a special role: a dealer D and an intermediary INT. The protocol will be executed over some large field K. The idea is that the IC protocol should provide us with a similar functionality as the digital signature schemes. Intuitively we want a dealer to be able to send a signed value $s \in K$ to the intermediary in such a way that (1) the intermediary can check if the signature is correct (2) he can later broadcast s together with a signature, demonstrating to the other players that s was indeed the value that he received from the dealer. Moreover the signature scheme will have a following *linearity* property: if player INT knows the signatures of D on some values $s_1, s_2 \in K$, then he can produce (non-interactively) a signature on $v = c_0 s_0 + c_1 s_1$ (for some publicly known constants c_0 and c_1). He can later present the signed value v to the other players (without disclosing s_0 and s_1) in such a way that the players will be convinced that v was indeed calculated according to the formula $v = c_0 s_0 + c_1 s_1$.

3.1 The Cons protocol

In this section we introduce a Cons protocol, which is an auxiliary protocol, that we will later use to build the IC protocol. The Cons protocol is an online protocol consisting of two rounds: a sending round $\mathrm{Send}_{\mathrm{Cons}}$ specified by a function $\mathsf{Send}_{\mathsf{Cons}}$ and a verifying round $\mathrm{Auth}_{\mathrm{Cons}}$ specified by a function $\mathsf{Auth}_{\mathsf{Cons}}$. The protocol involves three players: a dealer D, an intermediary INT and a receiver R. The other players in $\mathcal{P}$ do not take any action. We will assume that the adversary structure is such that the adversary can corrupt as many out of D, INT and R as he wants. Also, since he may corrupt some of the remaining players in $\mathcal{P}$, he can listen to all the messages that are broadcast. In the next sections we will write $\mathrm{Cons}(\mathrm{P}_1, \mathrm{P}_2, \mathrm{P}_3)$ to denote the Cons protocol, where players $\mathrm{P}_1, \mathrm{P}_2$ and P_3 play the roles of the dealer, the intermediary and the receiver, respectively. A similar convention will be applied to $\mathrm{Send}_{\mathrm{Cons}}(\mathrm{P}_1, \mathrm{P}_2, \mathrm{P}_3)$ and $\mathrm{Auth}_{\mathrm{Cons}}(\mathrm{P}_1, \mathrm{P}_2, \mathrm{P}_3)$.

3.1.1 Lin-consistent vectors

In this section we introduce a notion a *lin-consistent vector* (called 1_x-consistent in [CDD+99]), that we will use in the specification of the Cons protocol. We start we the following definition.

Definition 3.1
A vector $(s, t, x, y) \in K \times K \times (K \setminus \{0\}) \times K$ is *lin-consistent* if $y = s + tx$.

We will need the following notation. Take two vectors $\mathbf{w} = (s, t, x, y)$ and $\mathbf{w}' = (s', t', x, y')$ (both from K^4) and $a, a' \in K$. Then $a\mathbf{w} +_x a'\mathbf{w}'$ denotes a vector $(as + a's', at + a't', x, ay + a'y')$. The basic properties of lin-consistent vectors are given in the following lemmas.

Lemma 3.2 For every two lin-consistent vectors $\mathbf{v} = (s, t, x, y)$ and $\mathbf{v}' = (s', t', x, y')$ and every two numbers $b, b' \in K$ a vector $b\mathbf{v} +_x b'\mathbf{v}'$ is lin-consistent too.

Proof
Straightforward calculation. □

Lemma 3.3 Suppose $\mathbf{w}$ and $\mathbf{w}'$ are vectors from K^4 and $\mathbf{w}$ is not lin-consistent. Then there exists at most 1 element $a \in K$ such that $\mathbf{w} +_x a\mathbf{w}'$ is lin-consistent.

Proof
Suppose there exist two distinct a_0 and a_1 such that $\mathbf{u}_0 = \mathbf{w} +_x a_0\mathbf{w}'$ and $\mathbf{u}_1 = \mathbf{w} +_x a_1\mathbf{w}'$ are lin-consistent. We are now going to get a contradiction with the assumption that $\mathbf{w}$ is not lin-consistent. Clearly if a_0 or a_1 is equal to 0 then we are done. Otherwise take the vector $(1 - a_0 a_1^{-1})^{-1}\mathbf{u}_0 +_x (1 - a_1 a_0^{-1})\mathbf{u}_1$. It is easy to verify that this vector is equal to $\mathbf{w}$, however by Lemma 3.2, it has to be lin-consistent. Contradiction. □

3.1.2 Specification

The function $\mathsf{Send}_{\mathsf{Cons}}$ takes as an input a vector $(s, t, x, y) \in K^4$ from the dealer. It outputs:

- a pair (s, t) — to the intermediary, and
- a pair (x, y) — to the receiver.

The function $\mathsf{Auth}_{\mathsf{Cons}}$ takes as an input a *lin-consistent* vector (s_1, t_1, x_1, y_1) from the dealer. It outputs:

- a pair (s_1, t_1) — to the intermediary, and
- a pair (x_1, y_1) — to the receiver.

We will assume that if the dealer remained honest then (s, t, x, y) is equal to (s_1, t_1, x_1, y_1) (and thus if the dealer remained honest then (s, t, x, y) has to be lin-consistent).

The reader may ask what is the reason for introducing the $\mathrm{Send}_{\mathrm{Cons}}$ phase. In Section 3.1.7 we will give an answer for this.

3.1.3 Implementation

The $\mathrm{Send}_{\mathrm{Cons}}$ protocol is implemented in a straightforward way.

$\boxed{\mathrm{Send}_{\mathrm{Cons}}}$

1. The dealer sends (s, t) to INT and (x, y) to R (he does it in a single round).
2. Players INT and R output what they have received in the previous step.

The $\mathrm{Auth}_{\mathrm{Cons}}$ protocol is implemented in the following way:

$\boxed{\mathrm{Auth}_{\mathrm{Cons}}}$

1. The player D generates randomly s', t' and y' such that (s', t', x, y') is a lin-consistent vector. He sends (s', t') to INT and y' to R.
2. The player INT chooses randomly a value $a \in K$ (such that $a \neq 0$). He broadcasts a and a pair $(s + as', t + at')$.
3. Player D checks if the values broadcasted in Step 2. agree with what he sent to INT previously. If yes then he broadcasts an acceptance. Otherwise he broadcasts (s, t, x, y).[1] In this case the protocol ends here and the broadcasted values will be used in the following: player INT outputs (s, t) and R outputs (x, y).
4. The player R checks whether a vector $(s + as', t + at', x, y + ay')$ is lin-consistent.[2] If yes then he broadcasts an acceptance and outputs (x, y), player INT outputs (s, t) and this phase is finished.

   Otherwise R broadcasts a complaint. In this case D broadcasts x and y. The broadcasted values will be used in the following: player R outputs (x, y). Player INT outputs (s, t'), where t' is such that (s, t', x, y) is lin-consistent.

If players INT and R remained honest and they output (s_1, t_1) and (x_1, y_1), resp., such that (s_1, t_1, x_1, y_1) is not lin-consistent, then we say that *the dealer succeeded in cheating*.

[1] Clearly (s, t, x, y) broadcasted at this point needs to be a lin-consistent vector. If it is not then it is assumed that the D broadcasted some default vector (say $(0, 0, 1, 0)$).

[2] Here $(s + as', t + at')$ and a are the values that INT broadcasted in Step 2.

3.1.4 Construction of the simulator

The proof goes by the standard simulation (see Section 2.2). The $\mathrm{Send}_{\mathrm{Cons}}$ is easy: the simulator stays in the first corruption stage until the round in which the dealer sends his messages to INT and R. At this point he moves to the computation stage. If the dealer got corrupted then let (s, t, x, y) be the values that the simulated dealer chosen to send in this round. The simulator chooses (s, t, x, y) to be the input of the dealer that the simulator sends to the trusted party.

The $\mathrm{Auth}_{\mathrm{Cons}}$ is more involved. The simulator will stay in the first corruption stage until the simulated $\mathrm{Auth}_{\mathrm{Cons}}$ protocol halts. This works since we know that if the dealer is honest then $(s, t, x, y) = (s_1, t_1, x_1, y_1)$, and thus the values sent by him to INT in Step 1. are obtained by the simulator by corrupting INT (the same holds for R).

After the simulated protocol terminates the simulator does the following. The only non-trivial situation is when the dealer got corrupted. The main point is to determine the input of the dealer that the simulator will submit to the trusted party in the computation stage. We consider the following cases.

INT **got corrupted and** R **remained honest** In this case the simulator looks at the pair (x_1, y_1) output by the simulated R. He starts the computation stage choosing $(y_1, 0, x_1, y_1)$[3] to be the input of the dealer.

R **got corrupted and** INT **remained honest** This case is similar to the previous one. Let (s_1, t_1) be the output of the simulated INT. The simulator starts the computation stage choosing $(s_1, t_1, 1, t_1)$ to be the input of the dealer.

Both INT **and** R **got corrupted** In this case the simulator can choose an arbitrary lin-consistent vector to be the input of D.

Both INT **and** R **remained honest** The simulator he looks at the output (s_1, t_1) of the simulated INT and (x_1, y_1) of the simulated R. If (s_1, t_1, x_1, y_1) is not lin-consistent (i.e. the dealer succeeded in cheating) then the simulator finishes the simulation in an arbitrary way. Otherwise the simulator chooses (s_1, t_1, x_1, y_1) to be the input of the dealer and starts the computation stage. All the remaining stages are straightforward.

3.1.5 Analysis of the simulator

In this section we prove the correctness of the simulation defined in Section 3.1.4. We will say that an ERROR occurs if the dealer succeeds in cheating. We now have the following.

Lemma 3.4 For any execution of Cons the probability that the dealer succeeds in cheating is at most $1/|K|$.

[3] In this place he can choose arbitrary 1-consistent vector whose two last components are (x_1, y_1)

Proof
Let (s_c, t_c, x_c, y_c) be the values sent by the dealer to INT and R in the $\mathrm{Send}_{\mathrm{Cons}}$ phase. Clearly since he is corrupted (s_c, t_c, x_c, y_c) may not be lin-consistent. We will show that whatever are the actions of the corrupted dealer, with a probability at least $1 - 1/|K|$ the values s_o, t_o, x_o, y_o output by INT and R at the end of $\mathrm{Auth}_{\mathrm{Cons}}$ are lin-consistent. The argument goes as follows. Clearly if the corrupted dealer decided to broadcast some vector in Step 3. then we are done. Suppose he did not do it. Clearly if (s_c, t_c, x_c, y_c) is lin-consistent then we are done to. Assume that it is not. Let s'_c, t'_c, y'_c be the values sent by corrupted D in Step 1. By Lemma 3.3 there is at most $1/|K|$ probability that for a randomly chosen $a \in K$ vector $(s_c + as'_c, t_c + at'_c, x_c, y_c + ay'_c)$ is lin-consistent. Therefore with a chance at least $1 - 1/|K|$ player R will complain in Step 4. It is easy to see that in this case we are done. □

Lemma 3.5 The simulator constructed in Section 3.1.4 works secretly and correctly, unless ERROR occurred.

Proof
Assume that ERROR did not occur (i.e. the dealer did not succeed in cheating). First, observe that the correctness follows easily from the construction of the simulator. What remains is to consider the secrecy.

It is easy to see that the simulator works secretly for $\mathrm{Send}_{\mathrm{Cons}}$. Consider now the $\mathrm{Auth}_{\mathrm{Cons}}$. Clearly at the moment when D gets corrupted, or both R and INT get corrupted we are done (since at this moment we know all the inputs to the protocol). Therefore we need to consider the following cases:

Nobody is corrupted As long as none of D, INT and R is corrupted the only messages that the adversary receives are the ones broadcasted in Step 2. i.e. a value a and a pair $(s + as', t + at')$. Clearly their joint distribution does not depend on the input values s, t, x and y. Thus as long as nobody is corrupted the simulation works secretly.

R **gets corrupted first** In this case the only nontrivial thing that we need to prove is, that as long as D and INT remain honest, the information that comes to a corrupted receiver during $\mathrm{Auth}_{\mathrm{Cons}}$ is independent on s.

More precisely, observe that by corrupting R the adversary learns x and y. In that way he gets some information about the dealer's input values s and t, namely he now knows that $t = x^{-1}(y - s)$. Therefore what we need to show is that for every value s the distribution of the messages received by a corrupted R is the same. These messages contain the following values: y' (in Step 1.), and $a, s + as', t + at'$ (in Step 2.). Set $z_s := s + as'$ and $z_t := t + at'$. It is clear that since the dealer remained honest then $z_t = x^{-1}(ay' + y - z_s)$ (this is actually exactly the property that R checks in Step 4.). Therefore z_t brings no information and we can forget about it. Thus we remain with variables $y', a, s + as'$. It is clear that since y', a and s' are chosen independently and uniformly (and $a \neq 0$) the tuple

$(y', a, s + as')$ is also distributed uniformly (in $K \times (K \setminus \{0\}) \times K$) and thus it is independent on the value of s.

INT **gets corrupted first** This case is similar to the previous one. It is simpler however. This comes from the fact that the only variables that INT gets in the messages that come to him are s' and t' (Step 1). Clearly their distribution does not depend on (x, y).

$\square$

3.1.6 Security of the protocol

From Lemmas 3.5, 3.4 (and Lemma 2.14 from Section 2.2) we get the following.

Theorem 3.6
Protocol Cons *securely evaluates* Cons *with an error* $1/|K|$.

3.1.7 Why two phases are needed

The protocols $\mathrm{Send}_{\mathrm{Cons}}$ and $\mathrm{Auth}_{\mathrm{Cons}}$ can be combined in a straightforward way in a single-round protocol (simply $\mathrm{Send}_{\mathrm{Cons}}$ can become the first round of $\mathrm{Auth}_{\mathrm{Cons}}$). One may think that such a protocol will satisfy the specification given by the $\mathsf{Auth}_{\mathsf{Cons}}$ function. However, this is not true as when we require the *adaptive security*. This phenomenon is similar to the one described in Section 1.4.2. Look at the following example. Suppose player INT is corrupted and D is honest. The adversary does not want to corrupt R, but he wants him to output some value ($(1, 1)$, say) in case the input value s of the dealer is equal to 0. He wants to corrupt D only if it is necessary (i.e. when in his input $s = 0$). Now observe that in the real life the adversary is able to see s at the beginning of the protocol. If it is equal to 0 then he can corrupt the dealer and in Step 3. instruct D to broadcast $(1, 0, 1, 1)$, making the receiver output $(1, 1)$. In an ideal scenario he does not have this option: he has to decide whether to corrupt D *before* he can see s.

3.2 The IC protocol

Now, we are ready to introduce the IC protocol.

3.2.1 Specification

First, we are going to present the specification of the IC protocol. Fix two players: a dealer D and an intermediary INT. The protocol can have an unbounded number of phases. In each phase $p \in \mathbb{N}$ one of the following protocols may be executed:

1. a $\mathrm{Sign}^p_{\mathrm{IC}}$ phase — specified by a function $\mathsf{Sign}^p_{\mathsf{IC}}$ (every such phase will be called a *signing phase*),

2. a $\mathrm{Sum}^p_{\mathrm{IC}}(a_0, a_1, c_0, c_1)$ (where $a_0, a_1 \in \{1, \ldots, p-1\}$ and $c_0, c_1 \in K$) — specified by a function $\mathsf{Sum}^p_{\mathsf{IC}}(a_0, a_1, c_0, c_1)$ (a *summing phase*), and

3. a $\mathrm{Verify}^p_{\mathrm{IC}}(a)$ (where $a \in \{1, \ldots, p-1\}$) — specified by $\mathsf{Verify}^p_{\mathsf{IC}}(a)$ (a *verifying phase*).

We will assume that in the first phase the protocol $\mathrm{Sign}^1_{\mathrm{IC}}$ is called. The input and output of the trusted party in each phase will be a list $L \in (K \cup \{\texttt{nothing}\})^*$ (see Section 2.1.9 for more on lists). The idea is that each signing and summing phase will attach a new element $s \in K$ to the list. The verifying phase will attach a special symbol `nothing`.

The $\mathsf{Sign}^p_{\mathsf{IC}}$ function

The $\mathsf{Sign}^p_{\mathsf{IC}}$ function takes the following input:

- a list $L \in (K \cup \{\texttt{nothing}\})^*$ (of length $p-1$) — from the trusted party, and

- a value $s \in K$ — from the dealer D.

The output of INT is s, the output of other players is empty. The output of the trusted party is $L \cdot s$.

The $\mathsf{Sum}^p_{\mathsf{IC}}(a_0, a_1, c_0, c_1)$ function

The $\mathsf{Sum}^p_{\mathsf{IC}}(a_0, a_1, c_0, c_1)$ can be executed only if none of the phases a_0 and a_1 was a verifying phase. The function takes as an input a list $L \in (K \cup \texttt{nothing})^*$ (of length $p-1$) from the trusted party. For $i = 0, 1$ let s_i be the a_ith element of L. Set $s = a_0 s_0 + a_1 s_1$. The output of the trusted party is the list $L \cdot s$. The input of all the players is empty. The output of INT is s. The output of each other player is empty.

The $\mathsf{Verify}^p_{\mathsf{IC}}(a)$ function

The $\mathsf{Verify}^p_{\mathsf{IC}}(a)$ function can be executed only if a was not a verifying phase. It takes the following input:

- a list $L \in (K \cup \{\texttt{nothing}\})^*$ (of length $p-1$) — from the trusted party,

- a value $\mathrm{ifc}_{\mathrm{D}} \in K \cup \{\texttt{error}\}$ — from the dealer INT and

- a flag $\mathrm{corrupt}_{\mathrm{INT}} \in \{\texttt{true}, \texttt{false}\}$ — from the intermediary INT.

We will assume that if player INT remained honest then $\mathrm{corrupt}_{\mathrm{INT}} = \mathtt{false}$ and if player D remained honest then $\mathrm{ifc}_{\mathrm{D}} = \mathtt{error}$. Let s be the ath element of L. The output of the trusted party is $L \cdot \mathtt{nothing}$. The output of every player is a value x defined as follows.

$$x := \begin{cases} s & \text{if } \mathrm{corrupt}_{\mathrm{INT}} = \mathtt{false} \\ \mathrm{ifc}_{\mathrm{D}} & \text{otherwise.} \end{cases} \tag{3.1}$$

Intuitively this means that if INT remains honest (or behaves honestly) then the players will output s. Otherwise they will output a value chosen by D (if he is corrupted), or they will output `error` (if D remains honest). Note, that this implies that the players output some value that is neither equal to `error` nor to s, only if both INT and D are corrupted.

3.2.2 Implementation

In this section we are going to implement the IC protocol satisfying the specification from Section 3.2.1. The protocol is going to work in a Cons-hybrid model (see Section 3.1). As a result of each phase p the players will set their internal variables to some values. To avoid being too technical we introduce a notion of a *local output*. If we say that some player P *outputs locally* a value x (at the end of phase p), we will mean that P has a special register ($\mathtt{x}_p$, say) in which he puts the value of x. If we later refer to the local output of P from phase p, we will mean the value stored in $\mathtt{x}_p$.

The local output of each verifying phase is empty. The local output of every other phase is as follows.

- player D outputs locally a value $s_{\mathrm{D}} \in K$,
- player INT outputs locally a sequence $\{t_i\}_{i=1}^{n} \in K^n$,
- each player P_i outputs a pair $(x_i, y_i) \in K^2$.

The $\mathrm{Sign}^p_{\mathrm{IC}}$ protocol

The $\mathrm{Sign}^p_{\mathrm{IC}}$ is implemented as follows.

$\boxed{\mathrm{Sign}^p_{\mathrm{IC}}}$

1. For $i = 1, \ldots, n$ execute the following steps:
   (a) If this is the first phase of the protocol (i.e. $p = 1$) then the dealer D choses a random value $x_i \in K$. If $p > 1$ then he will use the same x_i which he chosen in the first phase.
   (b) The dealer D chooses randomly a value $t_i \in K$ and computes y_i such that (s, t_i, x_i, y_i) is lin-consistent.

(c) The players call the trusted party to compute $\mathsf{Send}_{\mathsf{Cons}}(\mathrm{D}, \mathrm{INT}, \mathrm{P}_i)$ and then $\mathsf{Auth}_{\mathsf{Cons}}(\mathrm{D}, \mathrm{INT}, \mathrm{P}_i)$. In both cases the dealer sends (s, t_i, x_i, y_i) as his input.

2. Note that if the dealer is corrupted then it may be the case that for two different i's (i_0 and i_1, say), he chosen two different secrets (s_0 and s_1, say). In this case INT broadcasts a complaint and the dealer has to broadcast all the vectors (s, t_i, x_i, y_i) (for $i = 1, \ldots, n$). Clearly the values that he sends need to be consistent (otherwise the players use some default values instead).

3. Every player P_i locally outputs (x_1, y_1), the player INT outputs s and locally outputs $\{t_i\}_{i=1}^n$, and the player D outputs locally s.

It is easy to see that if the dealer remains honest then each P_i outputs the same x_i in every phase. Therefore if a player P_i observes that in two phases he has locally output to different values of x_i, then he knows that D is corrupted. In this situation we will say that P_i *decides that the dealer is corrupted*. For simplicity we assume that this will not influence the actions of P_i until a verifying phase.

The $\mathrm{Sum}_{\mathrm{IC}}^p(a_0, a_1, c_0, c_1)$ **protocol**

The $\mathrm{Sum}_{\mathrm{IC}}^p(a_0, a_1, c_0, c_1)$ phase is implemented as follows. For $j = 0, 1$

- let s_{D}^j be the local output of D from phase a_j
- let s^j be the output of INT from phase a_j,
- let $\{t_i^j\}_{i=1}^n$ be the local output of INT from phase a_j, and
- let (x_i^j, y_i^j) be the local output of each P_i from phase a_j.

Then

- player D locally outputs $c_0 s_{\mathrm{D}}^0 + c_1 s_{\mathrm{D}}^1$,
- player INT outputs $c_0 s^0 + c_1 s^1$,
- player INT locally outputs $\{c_0 t_i^0 + c_1 t_i^1\}_{i=1}^n$, and
- each player P_i locally outputs $(x_i^0, c_0 y_i^0 + c_1 y_i^1)$.

The $\mathrm{Verify}_{\mathrm{IC}}^p(a)$ **protocol**

The $\mathrm{Verify}_{\mathrm{IC}}^p(a)$ protocol is implemented as follows.

- let s be the output of INT from phase a,
- let $\{t_i\}_{i=1}^n$ be the local output of INT from phase a, and
- let (x_i, y_i) be the local output of each P_i from phase a.

$\boxed{\mathrm{Verify}^p_{\mathrm{IC}}(a)}$

1. First, for technical reasons we assume that if it happened that a player P_i outputted `error` in some previous verifying phase he outputs `error` in this phase as well and halts. Since the outputs of all the (honest) players in a verifying phase are equal, this means that if one honest player halts at this point, then all the other ones do so.

   Otherwise we go to the next step.

2. Player INT broadcasts s and for every $i = 1, \ldots, n$ sends each t_i to P_i. It is important that he does it in a single round (see Section 3.2.7).

3. Every player P_i that decided (in the signing phase) that a dealer is corrupted broadcasts a complaint against the dealer. Otherwise he checks if (s, t_i, x_i, y_i) is lin-consistent. If no, then he broadcast a compliant

4. If the set of players that broadcasted the compliant is qualified then every player outputs `error`. Otherwise he outputs the value s broadcasted by INT in Step 2.

If it happened that D remained honest, INT got corrupted and the players have outputted some value $s' \neq s_{\mathrm{D}}$ (where s_{D} is the local output of D in phase a), then we say that *the adversary managed to forge a signature*.

3.2.3 Construction of the simulator

In this section we construct a standard simulator (see Section 2.2) for the IC protocol. Clearly as long as no player gets corrupted the simulation goes in a straightforward way. That is why we will assume that at least one player got corrupted. We start with the signing phase. At the beginning the simulator stays in the first corruption stage. Clearly as long as none of D and INT gets corrupted the simulation is straightforward. Consider the following cases.

Player INT **gets corrupted first** In this case the simulator goes to the computation stage. From the output of INT he learns the value of the dealer's input s (and updates the simulated protocol in a standard way). He finishes the simulation staying in the second corruption stage.

Player D **gets corrupted first** In this case the simulator stays in the first corruption stage until the protocol finishes this phase. Then, he has to determine the value of the dealer's input s that he will send to the trusted party in the computation stage. If INT gets corrupted then he sets s' to be equal to some arbitrary value (0, say). Otherwise he sets s' to be the value of the local output s of the simulated player INT. He then goes to the computation stage. The second corruption stage is empty.

The simulation of a summing phase is trivial since it involves no interaction. What remains is the verifying phase. First, the simulator stays in the first corruption stage. Let r_1 be the round in which player INT sends the messages in Step 2. If at the moment when the simulated protocol reaches r_1 player INT remains honest then the simulator goes to the computation stage. In this way he learns the value of s. Therefore he can update the simulated protocol in a standard way and finish the simulation in the second corruption stage. Otherwise (if INT was corrupted when the simulator entered r_1) the simulator stays in the first corruption stage. Then he looks at the output x of the simulated players (that remained honest). Now:

1. if D got corrupted then he sets $\mathrm{ifc}_{\mathrm{D}} = x$ and $\mathrm{corrupt}_{\mathrm{INT}} = \mathtt{true}$,
2. if D remained honest then he sets

$$\mathrm{corrupt}_{\mathrm{INT}} := \begin{cases} \mathtt{true} & \text{if } x = \mathtt{error} \\ \mathtt{false} & \text{otherwise.} \end{cases}$$

3.2.4 Analysis of the simulation

Let ERROR be the event that the adversary managed to forge a signature. In this section we prove the following.

Lemma 3.7 The simulator constructed in Section 3.2.3 works secretly and correctly unless ERROR occurs.

Proof
Assume that the ERROR did not occur. The proof goes by induction over the number of phases p. Suppose that we know that the simulator works secretly and correctly (unless ERROR occurs) until some phase $p-1$.

We start with the secrecy. Clearly the only interesting situation is when both D and INT remain honest. In this case we need to show that the messages received by the corrupted players are distributed independently on the choice of simulated inputs. It is easy to see that we can consider each player P_i separately. Suppose $\vec{s} = (s_1, \ldots, s_l)$ be a vector containing the simulated dealer's inputs for a signing phase so far. Let $\vec{y} = (y_1, \ldots, y_l)$ be a vector containing the local outputs of the simulated P_i and let $\vec{t} = (t_1, \ldots, t_l)$ be a vector containing the local outputs of the simulated INT. From the assumption that the protocol works correctly until phase $p-1$ we know that by executing the verifying phases player P_i learns some (publicly known) linear combinations of secrets in $\vec{s}$. Let $\vec{c_1}, \ldots, \vec{c_m}$ be those linear combinations (represented as vectors in K^l). Let C be a matrix whose jth row (for $j = 1, \ldots, m$) is $\vec{c_j}$. What player P has seen so far are the values sent to the simulator by the environment:

- matrix C,
- vector $\vec{S} = C \cdot \vec{s}$,

and the values produced by the simulator:

- value $x = x_i$ from the first phase,
- $\vec{y}$, and
- $\vec{T} = C \cdot \vec{t}$.

What we need to show is that for fixed C and $\vec{S}$ the distribution of $(x, \vec{y}, \vec{T})$ is the same for each $\vec{s'} \in K^l$ such that $C \cdot \vec{s'} = \vec{S}$. The crucial observation here is the following.

Lemma 3.8

$$\vec{T} = (C\vec{y} - \vec{S})x^{-1} \tag{3.2}$$

Proof
From the lin-consistency of each vector (s_j, t_j, x, y_j) we get

$$s_j + t_j x = y_j.$$

And thus

$$\vec{s} + \vec{t}x = \vec{y}.$$

And therefore

$$C \cdot \vec{s} + C \cdot \vec{t}x = C \cdot \vec{y}.$$

What implies (3.2) □

Thus by Lemma 3.8 (for fixed C and $\vec{S}$) the value of vector $\vec{T}$ is determined by x and $\vec{y}$. Therefore we are done since from the construction of the protocol it is clear that $(x, \vec{y})$ are chosen independently from $\vec{s}$. Thus the secrecy holds.

Consider now the correctness. Observe that the output of the players is empty in each signing or summing phase. Therefore we can focus on the verifying phase $\mathrm{Verify}^p_{\mathrm{IC}}(a)$. Recall that r_1 is the round in which player INT sends the messages in Step 2. Clearly if both INT and D remain honest we are done. Therefore we need to consider the following cases (let L be the input of the trusted party for phase p in the ideal model)

Both D **and** INT **got corrupted before** r_1 In this case the correctness holds easily since the simulator has a full control over the output of the players in the ideal model, by manipulating the inputs $\mathrm{ifc}_{\mathrm{D}}$ and $\mathrm{corrupt}_{\mathrm{INT}}$ (as in Point 1. on Page 53).

Only D **got corrupted before** r_1 Here the correctness holds by a straightforward analysis of the protocol. Clearly, the output by the (honest) simulated players is equal to the output of INT from phase a. It is easy to see that this is equal to the ath element of list L.

Only INT **got corrupted before** r_1 By the assumption that the adversary did not managed to forge a signature, the simulated players output either `error` or the real input of D from phase a. Again, it is easy to see that this value is equal to the ath element of L. Thus we are done.

□

What remains is to prove the following.

Lemma 3.9 For any execution of IC the probability that the dealer manages to forge a signature in any verifying phase is at most $n/(|K|-1)$.

Proof
Let $\mathrm{Verify}_{\mathrm{IC}}^{p}(a)$ be the first phase in which the adversary tries to forge a signature. Let s_a be the ath element of list L (the trusted party input for phase p in the ideal model). Let $s' \neq s$ be the value that the corrupted intermediary broadcasts in Step 2. It is easy to see that the adversary succeeds only if there exists an honest player P_i such that the adversary manages to produce t_i' that makes vector (s', t_i', x_i, y_i) lin-consistent. Clearly the only information that the adversary has on (x_i, y_i) is that he knows the pair (s_a, t) that makes (s_a, t, x_i, y_i) lin-consistent. In other words, he knows:

$$s_a + tx_i = y_i \tag{3.3}$$

and he wants to find (s', t') such that $s' \neq s_a$ and

$$s' + t'x_i = y_i. \tag{3.4}$$

By simple algebraic transformations we get that for any s' the matching t' is equal to $t_i + x_i^{-1}(s_a - s')$. Since x_i was chosen randomly the only information that the adversary gets about t' is that it cannot be equal to t_i. Therefore for any s_a his chance of guessing t' is at most $1/(|K|-1)$.

Since the number of players is n, by the union-bound we get that the total probability that the dealer manages to cheat in this phase is at most $n/(|K|-1)$. Observe that if the intermediary is caught on cheating (i.e. the players output `error`), then he has no more chances of cheating in the next phases (by the construction of Step 1 of the verifying protocol). □

3.2.5 Security of the protocol

Putting Lemmas 3.7 and 3.9 together we get the following.

Lemma 3.10 Protocol IC securely evaluates IC in a Cons-hybrid model with an error probability $n/(|K|-1)$.

Let ICReal denote the composed protocol $\mathrm{IC}^{\mathrm{Cons}}$. We get.

Theorem 3.11
Protocol ICReal *securely evaluates* IC *with an error at most* $n(m+1)/(|K|-1)$*, where* m *is the number of phases.*

Proof
It is easy to see that the Cons oracle is called at most n times in each phase. Therefore by Theorem 2.5 and remarks in Sections 2.1.7 and 2.1.8 we get that ICReal securely evaluates IC with an error at most

$$\frac{n}{|K|-1} + \frac{nm}{|K|}$$

which is clearly smaller than $n(m+1)/(|K|-1)$. Thus we are done. □

3.2.6 Complexity of ICReal

In this section we calculate the complexity of the ICReal protocol.

Lemma 3.12 Let n be the number of players, and let k be equal to $\log(|K|)$. Then the message complexity of each signing or verifying phase ICReal is $O(nk)$.[4] The message complexity of the summing phase is 0.

Proof
It is easy to see that in the signing phase the number of bits transmitted between INT and D (on one side) and every player P_i (on the other) is linear in k and the number of bits transmitted between D and INT is linear in nk. Therefore in total the number of bits transmitted in each signing phase is $O(nk)$. It is easy to see that the same bound holds for a verifying phase.

The case of summing phase is trivial, since the phase is non-interactive. □

3.2.7 More on Step 2. of a verifying phase

Finally, observe that the proof would not work (and the protocol would be insecure) if in Step 2. of the verifying phase the values were sent by INT in separate rounds. The type of problem is similar to the one described in Section 1.4.3 and comes from the adaptiveness of the adversary. For example suppose that s is broadcasted in some round r and the rest of the messages are sent in round $r+1$. Recall that in the simulation of the verifying phase we had a round r_1 in which the simulator had to decide whether to stay in the first corruption stage (he did it if INT was corrupted), or to go to the computation stage (he did it otherwise). Suppose INT remains honest during r. If he now chooses to go to the computation stage when he simulates r then the simulation does not work correctly (since the adversary can change the outputs of the honest players, if he corrupts INT). On the other hand if we choose to stay in the first corruption stage, then the simulation does not work secretly (since the broadcasted value s clearly depends on the simulated input of INT). It is also easy to see how to design an appropriate attack.

[4] We assume here that a message complexity of broadcasting a message of m bits is m.

3.2.8 Some informal terminology

In this section we introduce an informal terminology similar to the one of the digital signatures that will be useful in presenting the protocols working in a hybrid model with an access to IC oracles (Section 4.3 and Chapter 5). Usually we will use many copies of IC at the same time. More precisely we will assume that for every pair of players $(\mathrm{P}_0, \mathrm{P}_1)$ we have an oracle for IC where P_0 acts as a dealer and P_1 acts as an intermediary. When say that *a player* P_0 *sends a signature* $\sigma_s(\mathrm{P}_i, \mathrm{P}_j)$ *to* P_1 we mean that a signing phase is executed with a dealer P_0 and an intermediary P_1. By later saying that P_0 *broadcasts* s *together with* $\sigma_s(\mathrm{P}_i, \mathrm{P}_j)$ we will mean that a corresponding verifying phase is executed. Suppose that P_1 knows the signatures $\sigma_s(\mathrm{P}_0, \mathrm{P}_1)$ and $\sigma_u(\mathrm{P}_0, \mathrm{P}_1)$. By the linearity property we can assume that he is able to produce a signature $\sigma_{c_0 s + c_1 u}(\mathrm{P}_0, \mathrm{P}_1)$ for a value $c_0 s + c_1 u$ (where c_0 and c_1 are publicly known).

Chapter 4

Constructing VSS from SS

In this chapter we show the construction of a Verifiable Secret Sharing protocol (VSS) based on a Secret Sharing scheme (SS), i.e. we prove Theorem 1.1 (see Section 1.2.2).

The model is as in the previous chapter, namely we have a set of n players $\mathcal{P}$ connected pairwise by private channels, and a broadcast channel is available. We work over some $\mathcal{Q}_2$ adversary structure $\mathcal{F}$. The adversary is active and adaptive. We assume that we are given a Secret Sharing scheme $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$ secure against $\mathcal{F}$ (see Section 1.2.2 for the definition). We are going to construct a VSS protocol. The construction is divided in three steps (Sections 4.1, 4.2 and 4.3). The protocol VSS that we construct, will allow sharing of elements in a field Z_2. Clearly it can be extended to an arbitrary domain in a standard way.

A very general overview of the construction is as follows. We start (Section 4.1) with constructing a WSS protocol (which can be seen as a distributed commitment scheme). Then (Section 4.2) , we use a construction of Rudich (see [CvdGT95]) to transform WSS into a commitment scheme which allows a committer to prove in zero-knowledge that the sum of committed bits is equal to some bit. Given such a tool, it is clear that a dealer in a VSS protocol can use the commitment scheme we just developed to commit to all inputs and outputs of a run of the $\mathrm{Distr}_{\mathrm{SS}}$ algorithm in our SS scheme, and prove in zero-knowledge to the rest of the players that indeed the committed inputs result in the committed outputs (the shares of some secret). This almost immediately leads to a VSS protocol. However, apart from the fact that this may result in a huge loss of efficiency compared to the underlying SS scheme, it would also give a protocol whose actions depends heavily on which particular secret sharing scheme is used. In Section 4.3, we give a protocol achieving something slightly stronger, namely a VSS protocol that works given only black-box access to a secret sharing scheme and furthermore does not rely on any particular properties of this scheme. As we shall see, this matches the impossibility result we prove about MPC later (Chapter 6).

In the security proofs in this chapter we will use some combinatorial lemmas. Their proofs are moved to Section 4.4 at the end of the chapter.

4.1 The $\mathrm{WSS}^{\mathrm{V}}$ protocol

We start with specifying and implementing a single-verifier weak secret sharing protocol ($\mathrm{WSS}^{\mathrm{V}}$). It is a weaker version of a WSS protocol [RBO89] (see Section 1.4.1 for an informal definition of WSS). For a better understanding, we first define WSS and then modify it to get the definition of $\mathrm{WSS}^{\mathrm{V}}$.

4.1.1 Specification

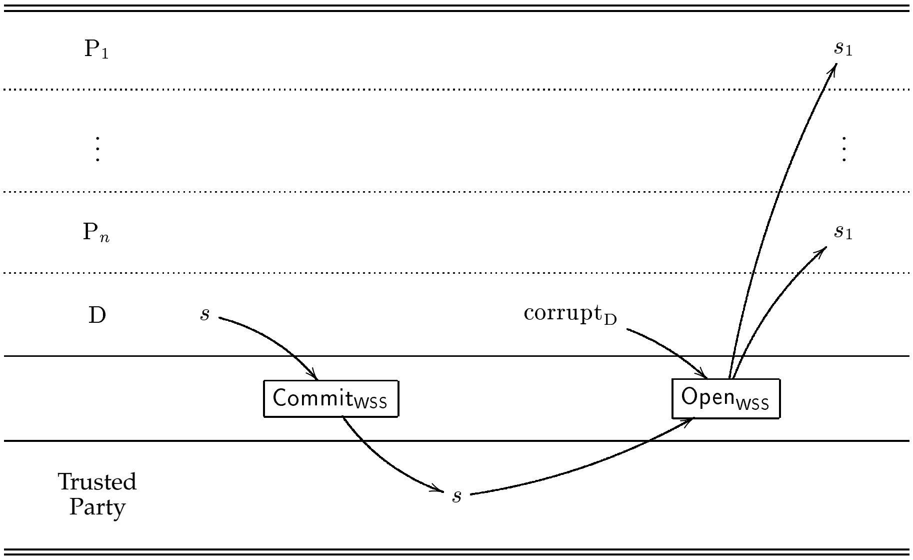

Figure 4.1: A scheme of a WSS protocol.

The WSS protocol

The WSS protocol consists of two phases: the commitment phase $\mathrm{Commit}_{\mathsf{WSS}}$ and the opening phase $\mathrm{Open}_{\mathsf{WSS}}$, which are specified by functions $\mathsf{Commit}_{\mathsf{WSS}}$ and $\mathsf{Open}_{\mathsf{WSS}}$ respectively. The protocol involves all players in $\mathcal{P}$, one of them being a dealer D. The only input that function $\mathsf{Commit}_{\mathsf{WSS}}$ takes is a value $s \in \mathbb{Z}_2$ from player D. The output of every player is empty. Only the trusted party outputs s. The $\mathsf{Open}_{\mathsf{WSS}}$ function takes input s from the trusted party and a flag $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ from D. Other players have no input. We will assume that if D remains honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$. The output of $\mathsf{Open}_{\mathsf{WSS}}$ given to every player is a value $s_1 \in \mathrm{Z}_2 \cup \{\mathtt{error}\}$ defined as follows:

$$s_1 = \begin{cases} s & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ \mathtt{error} & \text{otherwise.} \end{cases} \tag{4.1}$$

In what follows we will use a slightly modified version of WSS protocol. Let a *verifier* V be one of the players. The correctness of the protocol will depend on the honesty of V. More precisely a V*-verifier* WSS *(* $\mathsf{WSS}^{\mathrm{V}}$*)* consists of three rounds $\mathrm{Commit}^{\mathrm{V}}_{\mathrm{WSS}}$, $\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}$ and $\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}$, specified by functions $\mathsf{Commit}^{\mathrm{V}}_{\mathsf{WSS}}$, $\mathsf{PreOpen}^{\mathrm{V}}_{\mathsf{WSS}}$ and $\mathsf{Open}^{\mathrm{V}}_{\mathsf{WSS}}$. Function $\mathsf{Commit}^{\mathrm{V}}_{\mathsf{WSS}}$ is identical to $\mathsf{Commit}_{\mathsf{WSS}}$. Function $\mathsf{PreOpen}^{\mathrm{V}}_{\mathsf{WSS}}$ takes as an input a value $s' \in \mathrm{Z}_2$ from the dealer and outputs s' to every player. It also takes s from the trusted party and outputs it back to the trusted party. Function $\mathsf{Open}^{\mathrm{V}}_{\mathsf{WSS}}$ takes the following inputs:

- a flag $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ — from the dealer D,
- a value $\mathrm{ifc}_{\mathrm{V}} \in \mathtt{error} \cup \mathrm{Z}_2$ – from the verifier V, and
- a value s — from the trusted party.

We will assume that if the dealer is honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$ and if the verifier is honest then $\mathrm{ifc}_{\mathrm{V}} = \mathtt{error}$. Also, if the dealer remained honest then his input s (to $\mathsf{Commit}^{\mathrm{V}}_{\mathsf{WSS}}$) is equal to input s' (to $\mathsf{PreOpen}^{\mathrm{V}}_{\mathsf{WSS}}$). The output s_1 of every player is defined as

$$s_1 = \begin{cases} s & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ \mathrm{ifc}_{\mathrm{V}} & \text{otherwise.} \end{cases} \tag{4.2}$$

The role of $\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}$ is similar to $\mathrm{Send}_{\mathrm{Cons}}$ (Section 3.1). It is introduced to avoid an attack similar to the one in Section 1.4.3.

4.1.2 Implementation

In this section we present the implementation of the $\mathrm{WSS}^{\mathrm{V}}$ protocol. We will do it in a modular way. First, we introduce a SWSS protocol. This protocol is another weaker version of WSS, namely it works only when the set of the corrupted players is fixed after the committing phase is finished (in particular, this means that it is statically secure). We will not specify the protocol formally. Instead, we will prove some lemmas concerning it. Later we show how to use SWSS to construct $\mathrm{WSS}^{\mathrm{V}}$ and we will use those lemmas in the formal proof of $\mathrm{WSS}^{\mathrm{V}}$ security.

The protocols in this section work in an IC-hybrid model. We assume that the IC-functions are defined over some field K (we will say more about it later). In order to make the protocols better readable we use the informal terminology from Section 3.2.8. In this section we do exploit the linearity of the signatures. The only values that we are going to sign are 0 and 1.

SWSS

The SWSS consists of two phases: $\mathrm{SCommit}_{\mathsf{WSS}}$ and $\mathrm{SOpen}_{\mathsf{WSS}}$ implemented the same way as the WSS protocol in Section 1.4.1.

$\boxed{\mathrm{SCommit}_{\mathsf{WSS}}}$

1. Let $s \in \mathrm{Z}_2$ be the secret the dealer D wants to commit to. He shares it using $\mathrm{Distr}_{\mathrm{SS}}$ to get shares $\mathrm{sh}_1, \ldots, \mathrm{sh}_n$.
2. For each i he sends sh_i to P_i together with a signature $\sigma_{\mathrm{sh}_i}(\mathrm{D}, \mathrm{P}_i)$.

$\boxed{\mathrm{SOpen}_{\mathsf{WSS}}}$

1. The dealer broadcasts the secret s and the random input r used by $\mathrm{Distr}_{\mathrm{SS}}$ in the previous phase.
2. Every player P_i runs $\mathrm{Distr}_{\mathrm{SS}}$ on s and r that he received from the dealer. If the obtained share of player P_i matches sh_i (that he received in the previous phase) then he broadcasts an acceptance. Otherwise he complaints by broadcasting sh_i together with $\sigma_{\mathrm{sh}_i}(\mathrm{D}, \mathrm{P}_i)$.
3. For each properly signed value sh_i broadcasted in Step 2. every player P_j checks, if the complaint was justified (by running $\mathrm{Distr}_{\mathrm{SS}}$ on s and r and comparing the share of player P_i with sh_i). If he finds a justified complaint then he rejects the opening (and outputs `error`). Otherwise he accepts (and outputs s).

As observed in [RBO89] (see Section 1.4.1) this protocol is adaptively insecure.

Now, suppose that the set of corrupted players S is fixed when the phase $\mathrm{SCommit}_{\mathsf{WSS}}$ is finished. We will argue that fixing S commits the dealer to some value. More precisely take an arbitrary real life execution of $\mathrm{SCommit}_{\mathsf{WSS}}$. Let $\mathrm{sh}'_1, \ldots, \mathrm{sh}'_n$ be the values sent by the dealer (in Step 2.) to the players. For every set of players $S \in \mathcal{A}$ define a value $\sigma(S) \in \mathrm{Z}_2 \cup \{\mathtt{error}\}$ in the following way. By an exhaustive search try to find an input s' and a random input r for $\mathrm{Distr}_{\mathrm{SS}}$ with a following property: for all players $P_i \notin A$ the share of P_i is equal to sh'_i. If such s' exists then set $\sigma(S) = s'$, otherwise set $\sigma(S) = \mathtt{error}$. In this way, for every sequence of values $\mathrm{sh}'_1, \ldots, \mathrm{sh}'_n$ we defined a function $\sigma_{\mathrm{sh}'_1,\ldots,\mathrm{sh}'_n} : \mathcal{A} \to \mathrm{Z}_2 \cup \{\mathtt{error}\}$. Now we claim the following.

Lemma 4.1 Take some real life execution of SWSS. Suppose $\mathrm{sh}'_1, \ldots, \mathrm{sh}'_n$ are the values sent by the dealer (in Step 2. of $\mathrm{SCommit}_{\mathsf{WSS}}$) to the players. Let Q be the set of players corrupted when the $\mathrm{SOpen}_{\mathsf{WSS}}$ finished. Then the output of the players is either $\sigma_{\mathrm{sh}'_1,\ldots,\mathrm{sh}'_n}(Q)$, or `error`.

Proof
Follows easily from the construction of $\mathrm{SOpen}_{\mathsf{WSS}}$ and the definition of the function $\sigma_{\mathrm{sh}'_1,\ldots,\mathrm{sh}'_n}$. □

Clearly we also have the following.

Lemma 4.2 If the dealer remains honest then executing $\mathrm{SCommit}_{\mathsf{WSS}}$ and then (some time later) $\mathrm{SOpen}_{\mathsf{WSS}}$ results in each (honest) player outputting s.

Lemma 4.3 Let A be a set from the adversary structure, such that $\mathrm{D} \not\in A$. Let X_A be a random variable all the messages received by the players in A from other players during the execution of $\mathrm{SCommit}_{\mathsf{WSS}}$. Then the distribution of X_A does not depend on the value of s.

Proof
Follows directly from the definition of secret sharing. □

WSS$^{\mathrm{V}}$

In this section we are going to present a WSS$^{\mathrm{V}}$ protocol that satisfies the specification from Section 4.1.1. The $\mathrm{Commit}^{\mathrm{V}}_{\mathrm{WSS}}$ phase goes as follows.

$\boxed{\mathrm{Commit}^{\mathrm{V}}_{\mathrm{WSS}}}$

1. The dealer D chooses randomly a string of bits $\mathbf{a} = a_1, \ldots, a_{2n+k+1}$. He commits to every a_i using the $\mathrm{SCommit}_{\mathsf{WSS}}$ protocol.
2. The verifier V randomly chooses two distinct $2n + k + 1$ bit strings $\mathbf{v}_0, \mathbf{v}_1$ and broadcasts them.
3. D computes the string $\mathbf{z} = \mathbf{v}_s \oplus \mathbf{a}$ and broadcasts it.

The $\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}$ phase is implemented in the following way.

$\boxed{\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}}$

1. D broadcasts s.
2. Every player outputs the value broadcasted by D.

The $\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}$ phase goes as follows:

$\boxed{\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}}$

1. D opens (using $\mathrm{SOpen}_{\mathsf{WSS}}$) all the commitments to a_i's that he made in Step 1. of the $\mathrm{Commit}^{\mathrm{V}}_{\mathrm{WSS}}$ phase.
2. If any of the openings in the previous step was unsuccessful then the $\mathrm{Open}_{\mathsf{WSS}}$ is also unsuccessful and the players outputs `error`. Otherwise each player checks if it indeed holds that $\mathbf{z} = \mathbf{v}_s \oplus \mathbf{a}$. If yes then he decides that the opening was successful and outputs s. Otherwise he outputs `error`.

The intuition behind this protocol is as follows. First, assume the V remains honest. Then the intuitive idea is that although the dealer may open the commitment to $\mathbf{a}$ in many different ways (since it is only statically secure), the maximum number is at most 2^n (except with negligible probability). This is a negligible fraction of the possible 2^{2n+k+1} strings. This means that right after having made $\mathbf{a}$, the dealer is effectively committed (in the *adaptive* sense) to a negligible size subset of the possible strings[1].

4.1.3 The security proof for $\mathrm{WSS}^{\mathrm{V}}$

Construction of the simulator

The simulation goes in a standard way (see Section 2.2). For simulating the phase $\mathrm{Commit}^{\mathrm{V}}_{\mathrm{WSS}}$ the simulator stays in the first corruption stage until the protocol for this phase halts. The only interesting situation is when the dealer got corrupted. In this case for $i = 1, \ldots, 2n + k + 1$ let $\mathrm{sh}^i_1, \ldots, \mathrm{sh}^i_n$ be the values that the dealer sent to the players while committing (using $\mathrm{SCommit}_{\mathsf{WSS}}$) to a_i (where a_i is the ith element of vector $\mathbf{a}$ from Step 1.). Now, for every set $S \in \mathcal{A}$ let $\mathbf{x}_S$ be a sequence $\{\sigma_{\mathrm{sh}^i_1,\ldots,\mathrm{sh}^i_n}(S)\}_{i=1}^{2n+k+1}$. Let $\mathcal{W}$ be the set of sequences $\{\mathbf{x}_S \oplus \mathbf{z}\}_{S\in\mathcal{A}}$.[2] Define

$$s' := \begin{cases} 0 & \text{if } \mathbf{v}_0 \in \mathcal{W} \\ 1 & \text{otherwise} \end{cases} \tag{4.3}$$

Finally, the simulator starts the computation stage choosing s' to be the input of the dealer. The second corruption stage is empty.

The simulation of $\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}$ is is easy: it goes along the same lines as the simulation of $\mathrm{Send}_{\mathrm{Cons}}$ (see Section 3.1.4).

The simulation of the $\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}$ phase goes in the following way. If the dealer remains honest, then the simulator goes to the computation stage in the round in which the broadcast is done (in Step 1.). In this way he learns the value of s (since we can assume that at least one player got corrupted), and the simulation can continue in a standard way. Otherwise (if the dealer got corrupted before the broadcast round), the simulator will stay in the first corruption stage until the protocol terminates. Let x be the output of the (honest) players. If the verifier V got corrupted then the simulator sets $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{true}$ and $\mathrm{ifc}_{\mathrm{V}} = x$. Otherwise he sets:

$$\mathrm{corrupt}_{\mathrm{D}} = \begin{cases} \mathtt{true} & \text{if } x = \mathtt{error} \\ \mathtt{false} & \text{otherwise.} \end{cases}$$

He starts the computations stage with the chosen input values. The second corruption stage is empty.

[1] Elements of our proof are reminiscent of a method introduced by M. Naor [Nao91] in the context of ordinary, computationally secure commitments from pseudo-randomness.

[2] Here $\oplus$ denotes the componentwise xor, with a convention that $\mathtt{error} \oplus b = \mathtt{error}$, for every b

Analysis of the simulator

Let us start with some lemmas about the real life execution of WSS$^{\mathrm{V}}$ (let $\mathcal{W}$ be defined as in the simulation)

Lemma 4.4 For $t \in \{0,1\}$ if $\mathbf{v}_t \notin \mathcal{W}$ then it can never happen that at the end of $\mathrm{Open}_{\mathsf{WSS}}$ the honest players output t.

Proof
Suppose the contrary. This means that D broadcasts $s = t$ in Step 1 of $\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}$. Moreover from the construction of $\mathrm{Open}_{\mathsf{WSS}}$ (and from Lemma 4.1) there needs to exist a set $A \in \mathcal{A}$ such that $\mathbf{x}_A \oplus \mathbf{v}_s = \mathbf{z}$. This implies $\mathbf{v}_s = \mathbf{x}_A \oplus \mathbf{z}$ and yields contradiction. □

Lemma 4.5 Suppose the verifier remains honest. Then the chance that both $\mathbf{v}_0$ and $\mathbf{v}_1$ belong to $\mathcal{W}$ is at most 2^{-k}.

Proof
We have to calculate the chance that there exist $\mathbf{w}_0, \mathbf{w}_1 \in \mathcal{W}$ such that $\mathbf{v}_0 \oplus \mathbf{v}_1 = \mathbf{w}_0 \oplus \mathbf{w}_1$. Now we argue as follows. The cardinality of $\mathcal{W}$ is at most 2^n. Therefore the number of strings $\mathbf{w} = \mathbf{w}_0 \oplus \mathbf{w}_1$ (such that both $\mathbf{w}_i \in \mathcal{W}$) is at most 2^{2n-1}. By the assumption that V is honest $\mathbf{v}_0 \oplus \mathbf{v}_1$ is a random (non all-zero) bit string of a length $2n+k+1$. Since there are $2^{2n+k+1}-1$ such strings, the chance that $\mathbf{v}_0, \mathbf{v}_1 \in \mathcal{W}$ is at most $2^{2n}/(2^{2n+k+1}-1) \leq 2^{-k}$. □

Let ERROR be the event that $\mathbf{v}_0$ and $\mathbf{v}_1$ belong to $\mathcal{W}$. We are now ready to prove the following.

Lemma 4.6 The simulator constructed in Section 4.1.3 works secretly and correctly unless ERROR occurs.

Proof
Clearly, the secrecy is a concern only in the dealer did not get corrupted and the simulation did not enter the round in which broadcast is done in Step 1. In this case the secrecy comes from Lemma 4.3 and the fact that in Step 1. the a_i's are chosen randomly.

For the correctness it is easy to see that the only non-trivial situation is when the dealer is corrupted and the verifier remains honest. By Lemma 4.5 if the value output by the players at the end of $\mathrm{Open}^{\mathrm{V}}_{\mathrm{WSS}}$ is not equal to `error`, then it is equal to s' (defined in (4.3)), unless $\mathbf{v}_0$ and $\mathbf{v}_1$ both belong to $\mathcal{W}$. This, however does not happen, unless ERROR occurs. Thus we are done. □

Putting Lemmas 4.5 and Lemma 4.6 together we get the following.

Lemma 4.7 Protocol WSS$^{\mathrm{V}}$ securely evaluates $\mathsf{WSS}^{\mathrm{V}}$ in an IC-hybrid model with an error probability at most 2^{-k}.

Take an ICReal protocol (from Section 3.2.5) working over a field $K = \mathrm{GF}(q)$, where $q > \max(n, 2^k)$. Let $\mathrm{WSSReal}^{\mathrm{V}}$ be the real-life protocol $\mathrm{WSS}^{\mathrm{V}^{\mathrm{ICReal}}}$. Combining Lemmas 4.7 with Theorem 3.11 we get the following.

Theorem 4.8
Protocol $\mathrm{WSSReal}^{\mathrm{V}}$ *securely evaluates* $\mathsf{WSS}^{\mathrm{V}}$ *with an error negligible in* k.

4.2 The WSSZK protocol

In the previous section we implemented a protocol for distributed commitments. What we need for constructing a VSS protocol, is a commitment protocol that satisfies some additional properties. Namely, we want the following.

1. Suppose a dealer has committed to two secrets s_0 and s_1. We want the dealer to be able to open to the other players the sum $s_0 \oplus s_1$ in a zero-knowledge way, i.e. the players should not get any extra information about the values of s_0 and s_1.
2. Suppose a dealer D_0 has committed to some secret s. We want the dealer to be able to transfer it to some other player D_1 in such a way that:
   - if both D_0 and D_1 remained honest, then the adversary gets no information about s, and
   - even if they are both corrupt, it must still be guaranteed that the two commitments contain the same value.

In the protocol that we are going to construct the security of each commitment will rely on some verifier V remaining honest. Therefore what we also want is the following:

3. Suppose a dealer D committed to some secret s, what was verified by a verifier V_0. We want him to be able to make a new commitment to s, this time verified by some other verifier V_1, in such a way that if both V_0 and V_1 remain honest, then it is guaranteed that both commitments were made to the same value.

For achieving first goal we will use the method of Rudich (see [CvdGT95]). For the second and third one we will use a Commitment Transfer Protocol (CTP), a generalization of an idea from [CDM00]).

In the security proofs we will use the combinatorial lemmas proven in Section 4.4.

4.2.1 Specification

The WSSZK protocol consists of several phases. The number of them may not be fixed. However, we will assume that there exists an upper bound m on it. In each pth phase (where $p = 1, \ldots, m$) one of the following protocols may be executed:

- the *commitment protocol* $\mathrm{Commit}^p_{\mathrm{WSSZK}}(\mathrm{V},\mathrm{D})$ (where $\mathrm{V},\mathrm{D} \in \mathcal{P}$) — specified by a function $\mathsf{Commit}^p_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{D})$,
- the *pre-open protocol* $\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D},a)$ (where $a \in \mathbb{N}$ and $\mathrm{D} \in \mathcal{P}$) — specified by a function $\mathsf{PreOpen}^p_{\mathsf{WSSZK}}(\mathrm{D})$,
- the *open protocol* $\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V},\mathrm{D},a)$ (where $a \in \mathbb{N}$ and $\mathrm{V},\mathrm{D} \in \mathcal{P}$) — specified by a function $\mathsf{Open}^p_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{D},a)$,
- the *pre-sum-open protocol* $\mathrm{PreSum}^p_{\mathrm{WSSZK}}(\mathrm{D},a,b)$ (where $a,b \in \mathbb{N}$ and $\mathrm{D} \in \mathcal{P}$) — specified by a function $\mathsf{PreSum}^p_{\mathsf{WSSZK}}(\mathrm{D},a,b)$
- the *sum-open protocol* $\mathrm{SumOpen}^p_{\mathrm{WSSZK}}(\mathrm{V},\mathrm{D},a,b)$ (where $a,b \in \mathbb{N}$ and V, $\mathrm{D} \in \mathcal{P}$) — specified by a function $\mathsf{SumOpen}^p_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{D},a,b)$,
- the *pre-commitment-transfer protocol* $\mathrm{PreCTP}^p_{\mathrm{WSSZK}}(\mathrm{D}_0,\mathrm{D}_1,a)$ (where $a \in \mathbb{N}$ and $\mathrm{D}_0,\mathrm{D}_1 \in \mathcal{P}$) — specified by a function $\mathsf{PreCTP}^p_{\mathsf{WSSZK}}(\mathrm{D}_0,\mathrm{D}_1,a)$, and
- the *commitment-transfer protocol* $\mathrm{CTP}^p_{\mathrm{WSSZK}}(\mathrm{V}_0,\mathrm{V}_1,\mathrm{D}_0,\mathrm{D}_1,a)$ (where a, $b \in \mathbb{N}$ and $\mathrm{D}_0,\mathrm{D}_1,\mathrm{V}_0,\mathrm{V}_1 \in \mathcal{P}$) — specified by a function $\mathsf{CTP}^p_{\mathsf{WSSZK}}(\mathrm{V}_0, \mathrm{V}_1,\mathrm{D}_0,\mathrm{D}_1,a)$.

The honest party input and output will contain the history of the execution of the $\mathrm{Commit}_{\mathrm{WSSZK}}$ protocol. Therefore it will be a list of triples: (a dealer, a bit and that he committed to, a verifier), or some special value `nothing`. Formally we will write $\mathcal{D}_{\mathrm{T}} = (\mathcal{P} \times \mathrm{Z}_2 \times \mathcal{P} \cup \mathtt{nothing})^*$. We will start with an empty list and attach a new element to it in each phase. See Section 2.1.9 for more information on lists.

The reason for introducing phases $\mathrm{PreOpen}_{\mathrm{WSSZK}}$, $\mathrm{PreSum}_{\mathrm{WSSZK}}$ and phase $\mathrm{PreCTP}_{\mathrm{WSSZK}}$ is the same as for introducing $\mathrm{PreOpen}^{\mathrm{V}}_{\mathrm{WSS}}$ in Section 4.1 and $\mathrm{Send}_{\mathrm{Cons}}$ in Section 3.1: it is a technical way to avoid an insecurity of the same type as the example in Section 1.4.3.

If (for some players $\mathrm{D}_0,\mathrm{D}_1,\mathrm{V}_0$ and V_1 and some $a \in \mathbb{N}$) a phase p is a $\mathrm{Commit}^p_{\mathrm{WSSZK}}(\mathrm{V}_1,\mathrm{D}_1)$ phase, or a $\mathrm{CTP}^p_{\mathrm{WSSZK}}(\mathrm{V}_0,\mathrm{V}_1,\mathrm{D}_0,\mathrm{D}_1,a)$ phase, then we will say that *player* D_1 *made a commitment number p verified by* V_1

If the ath element of L is $(\mathrm{D},s,\mathrm{V})$ (for some players D and V and a value $s \in \mathrm{Z}_2$) then we will say that *the secret of the commitment number a in the list L is* s.

The commitment protocol

The only function that can be called in the first phase is $\mathsf{Commit}^p_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{D})$ (for some V,D). It takes the following input:

- an input $s \in \mathrm{Z}_2$ — from player D,
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party (if $p = 1$ then we assume that L is an empty list).

The output of every player is empty. The output of the trusted party is the list $L \cdot (\mathrm{D}, s, \mathrm{V})$.

The pre-open protocol and the open protocol

The $\mathsf{PreOpen}^p_{\mathsf{WSSZK}}(\mathrm{D}, a)$ function can be called only if player D made the commitment number a. It takes as an input:

- value $s \in K$ — from player D,
- a flag $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ — from D, and
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We will assume that if D remains honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$. Let s_0 be the secret of the commitment number a in the list L. The output x of every player is a value defined as follows.

$$x := \begin{cases} s_0 & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ s & \text{otherwise.} \end{cases}$$

The output of the trusted party is $L \cdot \mathtt{nothing}$. As the reader may guess the implementation of this function will be just an instruction for D to broadcast s_0.

The $\mathsf{Open}^p_{\mathsf{WSSZK}}(\mathrm{V}, \mathrm{D}, a)$ can be called only if (1) player D has made the commitment number a verified by V and (2) in the previous phase a function $\mathsf{PreOpen}^{p-1}_{\mathsf{WSSZK}}(\mathrm{D}, a)$ was called. The function takes the following inputs:

- a bit $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ — from D,
- a value $\mathrm{ifc}_{\mathrm{V}} \in \{\mathtt{error}\} \cup \mathrm{Z}_2$ — from V, and
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We make an assumption that if D remained honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$. If V remained honest then $\mathrm{ifc}_{\mathrm{V}} = \mathtt{error}$. The output of the trusted party is $L \cdot \mathtt{nothing}$. Let s_0 be the secret of the commitment number a in the list L. The output of every player is a value x defined as follows:

$$x := \begin{cases} s_0 & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ \mathrm{ifc}_{\mathrm{V}} & \text{otherwise.} \end{cases} \tag{4.4}$$

The pre-sum-open protocol and the sum-open protocol

The $\mathsf{PreSum}^p_{\mathsf{WSSZK}}(\mathrm{D}, a, b)$ plays a role similar to $\mathsf{PreOpen}_{\mathsf{WSSZK}}$. It can be called only if player D has made the commitments number a and b. It takes as an input:

- a value $s \in K$ — from player D,
- a flag $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ — from D, and

- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We will assume that if D remains honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$. Let s_0 and s_1 be the secrets of the commitments number a and b in the list L, respectively. The output x of every player is a pair defined as follows

$$x := \begin{cases} s_0 \oplus s_1 & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ s & \text{otherwise.} \end{cases}$$

The output of the trusted party is $L \cdot \mathtt{nothing}$.

The $\mathsf{SumOpen}^{p}_{\mathsf{WSSZK}}(\mathrm{V}, \mathrm{D}, a, b)$ function can be called only if (1) player D has made the commitments number a and b, both verified by V and (2) in the previous phase a function $\mathsf{PreSum}^{p-1}_{\mathsf{WSSZK}}(\mathrm{D}, a, b)$ was called. The function takes the following inputs:

- a flag $\mathrm{corrupt}_{\mathrm{D}} \in \{\mathtt{true}, \mathtt{false}\}$ — from D,
- a value $\mathrm{ifc}_{\mathrm{V}} \in \{\mathtt{error}\} \cup \mathrm{Z}_2$ — from V
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We also assume that if D remained honest then $\mathrm{corrupt}_{\mathrm{D}} = \mathtt{false}$ and if V remained honest then $\mathrm{ifc}_{\mathrm{V}} = \mathtt{error}$. The output of the trusted party is $L \cdot \mathtt{nothing}$. Let s_0 and s_1 be the secrets of the commitments number a and b in the list L, respectively. The output of every party is a value $x \in \mathrm{Z}_2 \cup \mathtt{error}$ defined as follows.

$$x := \begin{cases} s_0 \oplus s_1 & \text{if } \mathrm{corrupt}_{\mathrm{D}} = \mathtt{false} \\ \mathrm{ifc}_{\mathrm{V}} & \text{otherwise.} \end{cases}$$

The pre-commitment-transfer phase and the commitment-transfer protocol

The role of $\mathsf{PreCTP}^{p}_{\mathsf{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)$ is similar to the role of $\mathsf{PreOpen}_{\mathsf{WSSZK}}$ and $\mathsf{PreSum}_{\mathsf{WSSZK}}$. It can be called only if player D_0 has made the commitment number a. It takes as an input:

- a value $s \in \mathrm{Z}_2$ — from player D_0,
- a flag $\mathrm{corrupt}_{\mathrm{D}_0} \in \{\mathtt{true}, \mathtt{false}\}$ — from player D_0, and
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We will assume that if D_0 remained honest then $\mathrm{corrupt}_{\mathrm{D}_0} = \mathtt{false}$. Let s_0 be the secret of the commitment number a in the list L. The output x of player D_1 is a value defined as follows:

$$x := \begin{cases} s_0 & \text{if } \mathrm{corrupt}_{\mathrm{D}_0} = \mathtt{false} \\ s & \text{otherwise.} \end{cases}$$

The output of the other players is empty.

The $\mathsf{CTP}^{p}_{\mathsf{WSSZK}}(\mathrm{V}_0, \mathrm{V}_1, \mathrm{D}_0, \mathrm{D}_1, a)$ function can be called only if (1) player D_0 has made the commitment number a (2) in the previous phase a function $\mathsf{PreCTP}^{p-1}_{\mathsf{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)$ was called. The function takes the following inputs:

- a bit $\mathrm{corrupt}_{\mathrm{D}_0} \in \{\mathtt{true}, \mathtt{false}\}$ — from D_0,
- a value $\mathrm{ifc}_{\mathrm{V}_0} \in \mathrm{Z}_2 \cup \{\mathtt{error}\}$ — from V_0,
- a list $L \in \mathcal{D}_{\mathrm{T}}$ — from the trusted party.

We assume here that if D_0 remained honest then $\mathrm{corrupt}_{\mathrm{D}_0} = \mathtt{false}$ and if V_0 remained honest then $\mathrm{ifc}_{\mathrm{V}_0} = \mathtt{error}$. Let s_0 be the secret of the commitment number a in the list L. Define x as follows:

$$x := \begin{cases} s_0 & \text{if } \mathrm{corrupt}_{\mathrm{D}_0} = \mathtt{false} \\ \mathrm{ifc}_{\mathrm{V}_0} & \text{otherwise.} \end{cases} \tag{4.5}$$

If $x \neq \mathtt{error}$ then the output of the trusted party is a list $L \cdot (\mathrm{D}_1, s_0, \mathrm{V}_1)$ and the output of every player is empty. Otherwise the output of the trusted party is $L \cdot 0$ (here 0 is an arbitrary default value) and every player outputs $\mathtt{error}$

4.2.2 Implementation

In this section we are going to present the implementations of the protocols specified in Section 4.2.1. Formally the protocols that we will describe are implemented in a hybrid model with an access to oracles for computing $\mathsf{WSS}^{\mathrm{V}}$. In order to make the description better readable we will use a more informal terminology. Instead of saying that players call an oracle $\mathsf{Commit}^{\mathrm{V}}_{\mathsf{WSS}}$, with player V being a verifier and player D acting as a dealer with some input s, we will say that D $\mathrm{WSS}^{\mathrm{V}}$*-commits to the bit s, with verifier* V. Later, when we say that *the player* D *opens the* $\mathrm{WSS}^{\mathrm{V}}$*-commitment* we will mean that the players first call $\mathsf{PreOpen}^{\mathrm{V}}_{\mathsf{WSS}}$ and then $\mathsf{Open}^{\mathrm{V}}_{\mathsf{WSS}}$. By adding the "$\mathrm{WSS}^{\mathrm{V}}$-" prefix we hope to avoid the confusion between the commitment protocol that we construct (namely WSSZK) and the one that we start from ($\mathrm{WSS}^{\mathrm{V}}$).

In the protocols we will sometimes say that *the value of a commitment number p is publicly known to be equal to some value s*. This will happen in the situations in which the value that the dealer committed to is known to all the players (e.g. it was set to a default value because the dealer was publicly detected to be corrupted). Note, that we cannot just ask every player to set some default values of his shares. This is because we work in a hybrid model and the meaning of *share* depends on the implementation of the protocols substituting the calls to the trusted party.

The basic idea is a modification of a method of [CvdGT95]. A commitment to a bit s will be represented as a sequence of pairs $\{(r_{i,L}, r_{i,R})\}_i$ such that (1) for each i we have $r_{i,L} \oplus r_{i,R} = s$ and (2) the dealer is $\mathrm{WSS}^{\mathrm{V}}$-committed to each $r_{i,j}$. To open the commitment the dealer simply broadcasts the pairs and opens the $\mathrm{WSS}^{\mathrm{V}}$-commitments. The players accept the opening if the condition (1) is satisfied. The idea is that this representation of the commitment will allow the players to compute in zero-knowledge the sum of two $\mathrm{WSS}^{\mathrm{V}}$-committed bits (see Section 4.2.2). For the technical reasons it will be useful to prepare the a separate sequence of $(r_{i,L}, r_{i,R})$ pairs for each phase p (such a sequence will be called a *segment*). The reader may observe that we are very generous here and some optimizations are possible.

The commitment protocol

The $\mathrm{Commit}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D})$ protocol is implemented as follows (recall that m is the upper bound on the total number of phases in the protocol).

$\boxed{\mathrm{Commit}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D})}$

1. Player D inputs s and generates km pairs of random bits: $\{(r_{i,L}, r_{i,R})\}_{i=1}^{km}$ such that for every i it is the case that $r_{i,L} \oplus r_{i,R} = s$.
2. Player D $\mathrm{WSS}^{\mathrm{V}}$-commits to the bits $\{r_{i,j}\}_{i \in \{1,\ldots,km\}, j \in \{L,R\}}$ with verifier V.
3. Player V chooses a random permutation $\pi : \{1, \ldots, km\} \to \{1, \ldots, km\}$ and broadcasts it.
4. The players divide the $\mathrm{WSS}^{\mathrm{V}}$-commitments made in Step 2. into m sequences $\{\mathrm{segment}^p_t\}_{t=1}^{m}$ determined by the permutation announced in the previous step, setting:

$$\mathrm{segment}^p_t = \{(r_{\pi(i),L}, r_{\pi(i),R})\}_{i=(t-1)k+1}^{tk+1}. \tag{4.6}$$

The intuition is as follows. Suppose that the dealer got corrupted and the verifier remained honest. Some of the $(r_{i,L}, r_{i,R})$ pairs xor to 0 (call them 0-*pairs*) and some other ones xor to 1 (call them 1-*pairs*). The idea is that if k is sufficiently large, then the number of 0-pairs (and thus 1-pairs) in every segment should be more or less the same. Therefore with large probability there has to exists at least one $b \in \{0, 1\}$ such that the number of b-pairs is at least $k/3$ in each segment. Intuitively if there exists one b satisfying this property then the (corrupted) dealer has committed to b. Otherwise (when both $b = 0, 1$ satisfy it), the dealer has made an invalid commitment (and he will be caught in the opening or in the sum-opening phase).

Lemma 4.9 For each $j = 1, \ldots, m$ and $b = 0, 1$ let z^b_j denote the number of b-pairs in $\mathrm{segment}^p_j$. If the verifier remains honest then, except with probability $m(m-1)\exp(-k/18)$ there exists a bit b such that for each i we have $z^b_i \geq k/3$.

Proof
Set $b_i := r_{i,L} \oplus r_{i,R}$ and use Corollary 4.22 (from Section 4.4). □

If it happens otherwise (i.e. such a bit does not exists) we say that *the adversary managed to cheat in the commitment protocol*.

The pre-open protocol and the open protocol

The $\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D}, a)$ protocol is implemented in a straightforward way.

$\boxed{\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D}, a)}$

1. Player D broadcasts the value that he committed to in phase a.
2. Each player P_i outputs the value broadcasted in Step 1.

The $\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a)$ protocol is implemented as follows (clearly if the value of the commitment number a is publicly known, then the protocol becomes trivial). Let s be the value broadcasted by D in Step 1. of the phase $\mathrm{PreOpen}^{p-1}_{\mathrm{WSSZK}}(\mathrm{D}, a)$.

$\boxed{\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a)}$

1. Player D opens all the $\mathrm{WSS}^{\mathrm{V}}$-commitments to the bits in $\mathrm{segment}^p_a$. Let $\{(u_{i,L}, u_{i,R})\}_{i=1}^k$ be the resulting bits.
2. If the openings was successful and for every $i = 1, \ldots, k$ it is the case that $u_{i,L} \oplus u_{i,R} = s$ then every player outputs s. Otherwise he outputs `error`.

The pre-sum-open protocol and the sum-open protocol

The $\mathrm{PreSum}^p_{\mathrm{WSSZK}}(\mathrm{D}, a, b)$ protocol is implemented as follows.

$\boxed{\mathrm{PreSum}^p_{\mathrm{WSSZK}}(\mathrm{D}, a, b)}$

1. Player D broadcasts the sum of values that he committed to in phases a and b.
2. Each player P_i outputs the value broadcasted in Step 1.

The $\mathrm{SumOpen}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a, b)$ protocol is implemented in the following way (clearly if any of the values of the commitments number a or b is publicly known then we can use the $\mathrm{Open}^p_{\mathrm{WSSZK}}$ instead) . First, to simplify the notation write $\{(u_{i,L}, u_{i,R})\}_{i=1}^k = \mathrm{segment}^p_a$ and $\{(v_{i,L}, v_{i,R})\}_{i=1}^k = \mathrm{segment}^p_b$. Let s be the value broadcasted by D in Step 1. of $\mathrm{PreSum}^p_{\mathrm{WSSZK}}(\mathrm{D}, a, b)$.

$\boxed{\mathrm{SumOpen}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a, b)}$

1. Verifier V broadcasts a random permutation $\rho : \{1, \ldots, k\} \to \{1, \ldots, k\}$.[3]
2. For every $i \in \{1, \ldots, k\}$ and $j \in \{L, R\}$ player D broadcasts $w_{i,j} := u_{i,j} \oplus v_{\rho(i),j}$. If there exists $i \in \{1, \ldots, k\}$ such that $s \neq w_{i,L} \oplus w_{i,R}$ then every player outputs `error` and halts.

[3] In fact this step is not necessary for the correctness of the protocol (i.e. the verifier could choose for example a trivial permutation $\rho(i) = i$.). We introduce it for the simplicity of the proof.

3. Otherwise for every $i = 1, \ldots, k$ the following is executed:
   (a) player V chooses random $h \in \{L, R\}$ and broadcasts it,
   (b) players opens the $\mathrm{WSS}^{\mathrm{V}}$-commitments to $u_{i,h}$ and to $v_{\rho(i),h}$
   (c) if the openings failed or if $w_{i,h} \neq u_{i,h} \oplus v_{\rho(i),h}$ then every player outputs `error` and halts.
4. Every player outputs s.

The following lemma should be helpful in understanding the idea of the protocol (recall the intuition presented after the implementation of $\mathrm{Commit}_{\mathrm{WSSZK}}$).

Lemma 4.10 Suppose the verifier remained honest and s_a and s_b are such that there are at least $k/3$ s_a-pairs in $\mathrm{segment}_a^p$ and at least $k/3$ s_b-pairs in $\mathrm{segment}_b^p$, then the probability that the players output $x \neq s_a \oplus s_b$ (and $x \neq$ `error`) is at most $\exp(-k/108) + 2^{-k/18}$.

Proof
Let $X_a = \{\ i : u_{i,L} \oplus u_{i,R} = s_a\ \}$ and let $X_b = \{\ \rho(i) : v_{i,L} \oplus v_{i,R} = s_b\ \}$. Let $X = X_a \cap X_b$. From the way s_a and s_b are chosen we have that $|X_a| \geq k/3$ and $|X_b| \geq k/3$. Thus by Lemma 4.23 with probability at least $1 - \exp(-k/108)$ we have $|X| \geq k/18$. Suppose that the players output $x \neq s_a \oplus s_b$ (and $x \neq$ `error`). It is easy to see that this means that for each $i \in X$ there exists $j \in \{L, R\}$ such that a value $w_{i,j}$ (broadcasted by the dealer in Step 2.) is not equal to $u_{i,j} \oplus v_{\rho(i),j}$. Therefore the chances of a dealer of not being caught (in Step 3c.) are $2^{-|X|} \leq 2^{-k/18}$. Thus the total probability of the players outputting an incorrect value is $\exp(-k/108) + 2^{-k/18}$. □

If it happened that the players have output a non-error value $x \neq s_a \oplus s_b$, then we will say that *the adversary managed to cheat in the sum-open phase*.

The pre-commitment-transfer protocol and the commitment-transfer protocol

The $\mathrm{PreCTP}^p_{\mathrm{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)$ protocol goes as follows.

$\boxed{\mathrm{PreCTP}^p_{\mathrm{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)}$

1. Player D_0 sends to D_1 the value that he committed to in phase a.
2. Player D_1 outputs the value that he received in Step 1.

The $\mathrm{CTP}^p_{\mathrm{WSSZK}}(\mathrm{V}_0, \mathrm{V}_1, \mathrm{D}_0, \mathrm{D}_1, a)$ protocol is implemented in the following way (we will use a $\mathrm{HandleError}$ procedure defined later). Let s_0 be the value that D_0 committed to in phase a. Let s_1 be the value received by D_1 in Step 1. of $\mathrm{PreCTP}^p_{\mathrm{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)$ (note, that s_0 is equal to s_1 if both D_i's are honest).

$\boxed{\mathrm{CTP}^p_{\mathrm{WSSZK}}(\mathrm{V}_0, \mathrm{V}_1, \mathrm{D}_0, \mathrm{D}_1, a)}$

1. Player D_0 sends to D_1 all the data that he used when creating the commitment number a (i.e. all the pairs $\{(r_{i,L}, r_{i,R})\}_{i=1}^{km}$).

2. Player D_1 commits to s_1 by applying the procedure $\mathrm{Commit}^p_{\mathrm{WSSZK}}(\mathrm{V}_1, \mathrm{D}_1)$. If the result of the execution is `error` then players execute procedure $\mathrm{HandleError}$ (introduced below).

3. Player D_1 proves that the value he committed to is equal to s_0. He does it by executing $\mathrm{SumOpen}^p_{\mathrm{WSSZK}}(\mathrm{V}_1, \mathrm{D}_1, a, p)$. Recall that this involves opening the $\mathrm{WSS}^{\mathrm{V}}$-commitments in Step (3b). A technical point to note here is that the $\mathrm{WSS}^{\mathrm{V}}$-commitments to $u_{i,j}$'s are open by player D_0 (because he made them) and $\mathrm{WSS}^{\mathrm{V}}$-commitments to $v_{i,j}$'s are open by D_1. At the end every player looks at the output. If it is `error` or 1 then the players execute $\mathrm{HandleError}$ procedure, otherwise they output nothing and halt.

The $\mathrm{HandleError}$ procedure goes as follows. Player D_0 opens the commitment number a: he broadcasts s_0 and the players execute $\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V}_0, \mathrm{D}_0, a)$. If the opening is unsuccessful then each player outputs `error`, otherwise the commitment number p becomes known to be equal to s.

4.2.3 Construction of the simulator

We are doing it by the standard simulation method (see Section 2.2.). Wlog we can assume that at least one player got corrupted, and thus the simulator always knows the function chosen by the environment in a given phase.

The commitment protocol

Suppose the pth phase is a $\mathsf{Commit}^p_{\mathsf{WSSZK}}(\mathrm{V}, \mathrm{D})$ phase. The simulator will stay in the first corruption stage until the simulated protocol for phase p halts. The only interesting situation is when the dealer D got corrupted. In this case the simulator analyzes the situation in order to determine the value of s' that he will send to the trusted party as the input of D. If the verifier V got corrupted then he sets s' to be equal to some arbitrary value 0, say. Let us assume then that V remains honest. If during the simulation the players decided that D is corrupted than the simulators sets $s' = 0$. Otherwise he looks at the pairs $(r_{i,L}, r_{i,R})$ that the dealer has $\mathrm{WSS}^{\mathrm{V}}$-committed to in Step 1. Let z_i^b (for $j = 1, \ldots, m$ and $b = 0, 1$) be as in Lemma 4.9 If the adversary did not manage to cheat then there exist a bit b such that for each i we have $z_i^b \geq k/3$. If this is the case then the simulator sets $s' = b$. Note, that it can be the case that both $b = 0, 1$ satisfy this condition — in this case the simulator sets s' to an arbitrary value 0, say. Otherwise (if the adversary managed to cheat) the simulator sets s' to a default value 0.

The pre-open protocol and the open protocol

The simulation of $\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D}, a)$ is easy. If the dealer D remains honest then the simulator will go to the computation stage in a round in which the broadcast in Step 1. is done. In this way he will learn the value of the secret and the simulation can go on. If the dealer gets corrupted (before the broadcast in Step 1.), then the simulator stays in the first corruption stage until the protocol for this phase halts. Then he looks on the value s' output by the players. He sets $s = s'$ and $\mathrm{corrupt}_{\mathrm{D}} = \texttt{true}$ and sends this values to the trusted party as the input of D.

In the simulation of a $\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a)$ phase the simulator will stay in the first corruption stage until the protocol halts. Note, that he can do it because he already knows the secrets that the dealer will open, if he remained honest (since we have assumed that in the phase $p-1$ the broadcast protocol was executed). Then, he looks at the result of the simulation. If V got corrupted then the simulator is done since by setting $\mathrm{corrupt}_{\mathrm{D}} = \texttt{true}$ and setting $\mathrm{ifc}_{\mathrm{V}}$ to an arbitrary value s' we can always make the function $\mathsf{Open}_{\mathsf{WSSZK}}(\mathrm{V}, \mathrm{D}, a)$ output s'. Therefore the only interesting situation is when D is corrupted and V remains honest (and the players have not detected in phase a that the dealer D is corrupted). If in the simulated protocol the players output `error` then the simulator sets $\mathrm{corrupt}_{\mathrm{D}} := \texttt{true}$, otherwise $\mathrm{corrupt}_{\mathrm{D}} := \texttt{false}$. He then sends $\mathrm{corrupt}_{\mathrm{D}}$ to the trusted party as the input of D.

The pre-sum-open protocol and the sum-open protocol

The simulation of $\mathrm{PreSum}^p_{\mathrm{WSSZK}}(\mathrm{D}, a, b)$ and $\mathrm{SumOpen}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a, b)$ go in a very similar way to the simulation of $\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D}, a)$ and $\mathrm{Open}^p_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a)$. We skip the details.

The pre-commitment-transfer protocol and the commitment-transfer protocol

The simulation of $\mathrm{PreCTP}^p_{\mathrm{WSSZK}}(\mathrm{D}_0, \mathrm{D}_1, a)$ protocol is similar to the simulation of the $\mathrm{PreOpen}^p_{\mathrm{WSSZK}}(\mathrm{D}, a)$.

The simulation of $\mathrm{CTP}^p_{\mathrm{WSSZK}}(\mathrm{V}_0, \mathrm{V}_1, \mathrm{D}_0, \mathrm{D}_1, a)$ goes in the following way. The simulator stays in the first corruption stage until the protocol finished this phase. Clearly the only interesting situation is when at least one of D_0 and D_1 got corrupted. Let y be equal to `error` if the players output `error` at the end of the simulation. Otherwise let y be equal to the value that player D_1 committed to in Step 2. (defined in the same way as s' in the simulation of $\mathsf{Commit}^p_{\mathsf{WSSZK}}(\mathrm{V}, \mathrm{D})$). Suppose D_0 got corrupted. If V_0 also got corrupted then the simulator sets $\mathrm{corrupt}_{\mathrm{D}_0} = \texttt{true}$ and $\mathrm{ifc}_{\mathrm{V}_0} = y$. If V_0 remained honest then the simulator sets

$$\mathrm{corrupt}_{\mathrm{D}_0} := \begin{cases} \texttt{true} & \text{if } y = \texttt{error} \\ \texttt{false} & \text{otherwise.} \end{cases} \tag{4.7}$$

4.2.4 Analysis of the simulation

Define the error event ERROR to be the sum of all the following events

1. the adversary managed to cheat in any of the commitment protocols (executed as a separate protocol or as a sub-protocol of the commitment transfer protocol),
2. the adversary managed to cheat in any of the sum-open protocols (executed as a separate protocol or as a sub-protocol of the commitment transfer protocol)

Lemma 4.11 The probability that ERROR occurs is negligible in k (for a fixed m).

Proof
Follows Lemmas 4.9 and 4.10. □

Lemma 4.12 The simulator constructed in Section 4.2.3 works correctly and secretly unless ERROR occurs.

Proof
The proof goes by induction over the number of phases p. Suppose we know that the simulator works secretly and correctly until $p-1$. Consider the phase p.

Secrecy is easy. The only non-trivial cases are: the sum open protocol (when the dealer D remains honest) and commitment transfer protocols (when both dealers D_0 and D_1 remain honest. Let us focus on the sum open protocol (the case of commitment transfer protocol is analogous). The secrecy comes from the fact that for each pair (u_L, u_R) in $\mathrm{segment}_a^p$ or in $\mathrm{segment}_b^p$ the honest dealer never opens both components u_l and u_R. Since the pairs in $\mathrm{segment}_a^p$ and $\mathrm{segment}_b^p$ are used only in the phase number p we are done.

The correctness is more complicated. First, let us assume that during the simulation the adversary did not cheat in any of the commitment protocols. Assume also, that he did not cheat in any of the commitment protocols is used as sub-routines in the commitment-transfer protocols.

Informally speaking we need to show that in the $\mathrm{Open}_{\mathrm{WSSZK}}^p(\mathrm{V}, \mathrm{D}, a)$ and $\mathrm{SumOpen}_{\mathrm{WSSZK}}^p(\mathrm{V}, \mathrm{D}, a, b)$ phases a correct value is opened, and that in the $\mathrm{CTP}_{\mathrm{WSSZK}}^p(\mathrm{V}_0, \mathrm{V}_1, \mathrm{D}_0, \mathrm{D}_1, a)$ phase a correct value is transferred. It is easy to see that we can assume that the verifiers (V , V_0 and V_1) remain honest.

Let us first consider $\mathrm{Open}_{\mathrm{WSSZK}}^p(\mathrm{V}, \mathrm{D}, a)$. Let s_a denote the secret of the commitment number a in the list L (that is the input of the trusted party for $\mathsf{Open}_{\mathsf{WSSZK}}^p(\mathrm{V}, \mathrm{D}, a)$). Suppose the output x of the simulated players is not equal to `error`. We need to show that it is equal to s_a. From the way s was chosen we know that there exists a pair (u_L, u_R) in $\mathrm{segment}_a^p$ such that $u_l \oplus u_R = s_a$ (in fact there are even at least $k/3$ such pairs). Thus it is easy to see that the only non-error value that the simulated players can output in Step 2 is equal to s_a.

Now, let us consider $\mathrm{SumOpen}^{p}_{\mathrm{WSSZK}}(\mathrm{V}, \mathrm{D}, a, b)$. Let s_a and s_b be the secrets of the commitments number a and b (respectively) in the list L. From the assumption that the adversary did not manage to cheat it is easy to see that if the output x the simulated players is not equal to `error`, then it has to be equal to $s_a \oplus s_b$.

The correctness of the simulation of the commitment-transfer protocol follows easily from the correctness of the sum-opening protocol. □

Combining Lemmas 4.12 and 4.11 we get the following.

Lemma 4.13 Protocol WSSZK securely evaluates WSSZK in an $\mathsf{WSS}^{\mathrm{V}}$-hybrid model with an error negligible in k.

Let WSSZKReal denote protocol WSSZK composed with protocol $\mathrm{WSS}^{\mathrm{V}}$ (from Section 4.1). We get the following.

Theorem 4.14
Protocol WSSZKReal *securely evaluates* WSSZK *with an error negligible in* k.

4.3 The VSS protocol

4.3.1 Specification

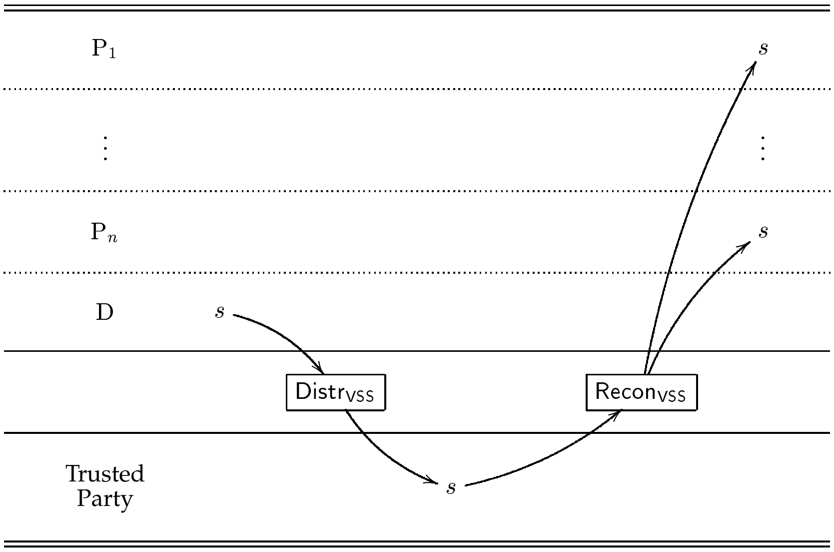

Figure 4.2: A scheme of a VSS protocol.

The VSS is a stronger primitive than WSS. For an informal definition see Section 1.2.2. Formally it is defined in the following way. It has two phases: $\mathrm{Distr}_{\mathrm{VSS}}$ and $\mathrm{Recon}_{\mathrm{VSS}}$ specified by functions $\mathsf{Distr}_{\mathsf{VSS}}$ and $\mathsf{Recon}_{\mathsf{VSS}}$ respectively. We choose one of the players to be a dealer D. The $\mathsf{Distr}_{\mathsf{VSS}}$ takes the input value $s \in \mathrm{Z}_2$ from the dealer and outputs it to the Trusted Party. The $\mathsf{Recon}_{\mathsf{VSS}}$ takes s from the Trusted Party outputs it to every player (see Figure 4.2).

4.3.2 Some terminology

We will implement the VSS protocol in a hybrid model, with a given access to a trusted party computing WSSZK (see Section 4.2). To make the description of the protocol clearer we adopt the following convention. First, assume that the players keep a counter p to count the number of WSSZK phases executed so far. If we say that *a player* P *has committed to* s*, what was verified by* V we mean that the players asked the trusted party to compute $\mathsf{Commit}^{a}_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{P})$, and s was the value submitted by D. Here a is the current value of p, which will refer to as the *number of the commitment*. The value of s will be later called the *secret of the commitment number* a. If we later on say that *players open the commitment number* a, then we mean that they execute $\mathsf{PreOpen}^{p}_{\mathsf{WSSZK}}(\mathrm{D},a)$ and then $\mathsf{Open}^{p+1}_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{P},a)$. The *result of the opening* is the value output by every honest player.

Suppose a player P has committed to s_0 and s_1, what was verified by V. Let a and b be the respective indices of these commitments. Then, by saying that *the players compute in zero-knowledge the sum of the values of the commitments* a *and* b we will mean that they execute $\mathsf{PreSum}^{p}_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{D},a,b)$ and then $\mathsf{SumOpen}^{p+1}_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{P},a,b)$.

Suppose a player P_0 has committed to s_0 what was verified by V. Let a be the number of this commitment. Let P_1 be some player. We say that P_0 *transfers the commitment* a *to* P_1 if $\mathsf{PreCTP}^{p}_{\mathsf{WSSZK}}(\mathrm{P}_0,\mathrm{P}_1,a)$ and then $\mathsf{CTP}^{p+1}_{\mathsf{WSSZK}}(\mathrm{V},\mathrm{V},\mathrm{P}_0,\mathrm{P}_1,a)$ was executed. The value of $p+1$ will be the number of the resulting commitment.

Transferring a commitment from a verifier V_0 to another verifier V_1 is defined analogously.

Finally, let us recall that we are given $(\mathrm{Distr}_{\mathrm{SS}},\mathrm{Recon}_{\mathrm{SS}})$, a secret sharing scheme secure against the adversary structure $\mathcal{F}$ (see the remarks at the beginning of this chapter).

4.3.3 The $\mathrm{VSS}^{\mathrm{V}}$ protocol

We start by constructing a $\mathrm{VSS}^{\mathrm{V}}$ protocol, a single-verifier version of VSS. It will consist of two phases $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$ and $\mathrm{Recon}^{\mathrm{V}}_{\mathrm{VSS}}$. We will not specify this protocol formally. Informally speaking $\mathrm{VSS}^{\mathrm{V}}$ is a VSS protocol, where the correctness holds only if the verifier V remains honest. The idea is similar to the one of $\mathrm{WSS}^{\mathrm{V}}$ protocol (Section 4.1). Later, we will use $\mathrm{VSS}^{\mathrm{V}}$ as a building block for VSS.

In the implementation of $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$ we are going to use the method known as "a cut & choose technique". The $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$ protocol is implemented in the following way.

$\boxed{\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}}$

1. Player D takes an input bit s and commits to it. Let $\mathrm{commit}^{s}_{\mathrm{D,V}}$ be the number of the commitment.

2. D chooses random bits $r_1, \ldots, r_{2k}$ and, for each i uses $\mathrm{Distr}_{\mathrm{SS}}$ to generate shares $s_{i1}, \ldots, s_{in}$ where r_i is the secret. Next, D makes commitments to all these values, resulting in commitments number $\mathrm{commit}^{r_i}_{\mathrm{D,V}}$ and $\mathrm{commit}^{s_{ij}}_{\mathrm{D,V}}$ (for $i = 1, \ldots, 2k$ and $j = 1, \ldots, n$).

3. V chooses at random a subset E consisting of half the indices $1, \ldots, 2k$, and broadcasts it. Now, for each $i \in E$, all commitments $\mathrm{commit}^{r_i}_{\mathrm{D,V}}$ and each $\mathrm{commit}^{s_{ij}}_{\mathrm{D}}$ (for $j = 1, \ldots, n$) are open, and D broadcasts all random inputs used to generate those shares s_{ij}.

4. Every player checks for each $i \in E$, that each set of shares s_{ij} consistently determines r_i (he does it by running the $\mathrm{Distr}_{\mathrm{SS}}$ protocol with the random input broadcasted in the previous step). If the verification fails then the dealer is disqualified, every player outputs `error` and halts. Otherwise the protocol goes to the next step (all information with $i \in E$ can now be discarded).

5. Write $\overline{E}$ for $\{1, \ldots, 2k\} \setminus E$. For each $i \in \overline{E}$, the players compute in zero-knowledge the sum of the commitments number $\mathrm{commit}^{r_i}_{\mathrm{D,V}}$ and $\mathrm{commit}^{s}_{\mathrm{D,V}}$. For every $i \in \overline{E}$ let c_i be the result.

6. Finally, for each $i \in \overline{E}$ and each $j = 1, \ldots, n$, the dealer transfers every commitment number $\mathrm{commit}^{s_{ij}}_{\mathrm{D,V}}$ to P_j what results in a commitment $\mathrm{commit}^{s_{ij}}_{\mathrm{P}_j}$.

The following lemma should be helpful in understanding the intuition behind the implementation.

Lemma 4.15 Suppose the verifier V remained honest. For every $i \in \{1, \ldots, 2k\}$ and every $j \in \{1, \ldots, n\}$ let s''_{ij} be the secret of the commitment number $\mathrm{commit}^{s_{ij}}_{\mathrm{D,V}}$ and let r''_i be the secret of the commitment number $\mathrm{commit}^{r_i}_{\mathrm{D,V}}$. Let Bad be the set of all i's such that the shares $\{s''_{ij}\}^{n}_{j=1}$ are not valid shares of r''_i. If $|\mathrm{Bad}| > k/3$ then with probability at least $1 - \exp(k/18)$ it in Step 4 the players output `error`.

Proof
By Lemma 4.20 (Section 4.4) if $|\mathrm{Bad}| > k/3$ then with probability at least $1 - \exp(k/18)$ it happened that $\mathrm{Bad} \cap E \neq \emptyset$. Thus in Step 4. they output `error`. □

If $|\mathrm{Bad}| > k/3$ and the players did not output `error` then we will say that *the dealer managed to cheat the verifier* V.

The $\mathrm{Recon}^{\mathrm{V}}_{\mathrm{VSS}}$ goes as follows.

$\boxed{\mathrm{Recon}^{\mathrm{V}}_{\mathrm{VSS}}}$

1. For each $i \in \overline{E}$ and $j = 1, \dots, n$ all commitments $\mathrm{commit}^{s_{ij}}_{\mathrm{P}_j}$ are opened by P_j.
2. Every player verifies all openings and discards those s_{ij}'s for which the commitment was not correctly opened. For those i's where a qualified set of shares (supposedly of r_i) remains, he reconstructs a value r'_i (using $\mathrm{Recon}_{\mathrm{SS}}$).
3. Every player P_i XOR's r'_i with the value for c_i (from Step 5. of $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$). This gives a set of bits (which will all be equal to s if D has been honest). Every P_i decides by majority among these bits the final value to output.

4.3.4 Implementation of the VSS protocol.

Using the $\mathrm{VSS}^{\mathrm{V}}$ introduced in the previous section we will now implement the VSS protocol.

The $\mathsf{Distr}_{\mathsf{VSS}}$ protocol

Let $s \in \mathrm{Z}_2$ is the input for the dealer D. The protocol goes as follows

$\boxed{\mathsf{Distr}_{\mathsf{VSS}}}$

1. For every player $\mathrm{P}_i \in \mathcal{P}$ the player execute $\mathrm{Distr}^{\mathrm{P}_i}_{\mathrm{VSS}}$, with the dealer D submitting s as his input.
2. Observe that in Step 1 of each $\mathrm{Distr}^{\mathrm{P}_i}_{\mathrm{VSS}}$ (executed in the previous step) the dealer committed to s by commitment number $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_i}$. For every pair of players $(\mathrm{P}_i, \mathrm{P}_j)$ the dealer has now to prove that the secrets of the commitments number $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_i}$ and $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_j}$ are equal if P_i and P_j remain honest. Therefore for every pair $(\mathrm{P}_i, \mathrm{P}_j)$ the following is executed

   $\boxed{\mathrm{Compare}(\mathrm{P}_i, \mathrm{P}_j)}$

   (a) the commitment number $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_i}$ is transferred from the verifier P_i to the verifier P_j (let ${\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_j}}'$ be the number of the resulting commitment)

(b) the players compute in zero-knowledge the sum of the secrets of the commitments number $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_j}{}'$ and $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{P}_j}$.

If during any of $\mathrm{Compare}(\mathrm{P}_i, \mathrm{P}_j)$ it happens that the players output `error` (as a result of some call to the trusted party computing WSSZK), or that the sum in Step (2b) is equal to 1 then the players decide that the dealer is corrupted and assume that he shared a default value 0.

The $\mathrm{Recon}_{\mathrm{VSS}}$ **protocol**

If in the $\mathrm{Distr}_{\mathrm{VSS}}$ protocol the players decided that the dealer is corrupted then the $\mathrm{Recon}_{\mathrm{VSS}}$ becomes trivial. Otherwise it is implemented as follows:

$\boxed{\mathrm{Recon}_{\mathrm{VSS}}}$

1. For every player $\mathrm{P}_i \in \mathcal{P}$ the players execute $\mathrm{Recon}^{\mathrm{P}_i}_{\mathrm{VSS}}$. The output of P_i is his output from $\mathrm{Recon}^{\mathrm{P}_i}_{\mathrm{VSS}}$.

4.3.5 Construction of the simulator

Again, we apply the standard simulation (see Section 2.2). Let us first focus on the simulation of $\mathrm{Distr}_{\mathrm{VSS}}$. The simulator will stay in the first corruption stage until the protocol finishes this phase. The only non-trivial case is when the dealer gets corrupted and is not caught (at the end of in Step 2). The simulator needs to find a value s' that he will send to the trusted party as the input of the dealer. He takes some player P_i that remains honest and looks at the execution of $\mathrm{Distr}^{\mathrm{P}_i}_{\mathrm{VSS}}$ (in Step 1. of $\mathrm{Distr}_{\mathrm{VSS}}$). He sets s' to be the secret of the commitment number $\mathrm{commit}^{s}_{\mathrm{D},\mathrm{V}}$. He starts the computation stage. The second corruption stage is empty.

The simulation of $\mathrm{Distr}_{\mathrm{VSS}}$ goes as follows. The simulator goes immediately to the computation stage. In this way he learns the real value of the dealer's secret s and the simulation can continue the simulation in the second corruption stage.

4.3.6 Analysis of the simulator

Let ERROR be the event that the dealer managed to cheat any verifier (in the $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$ subprotocol). In this section we are going to show the following lemma.

Lemma 4.16 The simulator constructed in Section 4.3.5 works correctly and securely, unless ERROR occurs.

Proof
It is easy to see that the secrecy requirement holds trivially for the $\mathrm{Recon}_{\mathrm{VSS}}$ phase. Therefore what remains is the $\mathrm{Distr}_{\mathrm{VSS}}$ phase. We can also assume that

the dealer remains honest. Therefore the only nontrivial part is each $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$. It is easy to see that all the values received by any coalition of the players (not including D) in Steps 1.–5. are independent from the value of s. Also, by the properties of the secret sharing, the values received in Step 6. by any set of players $A \in \mathcal{A}$ are random. Thus we are done.

Let us now assume that the dealer did not manage to cheat any verifier (i.e. the ERROR did not occur) and argue for the correctness. If the dealer remained honest until the end of the $\mathrm{Distr}_{\mathrm{VSS}}$ phase, then the correctness is straightforward. Therefore let us assume that he got corrupted during $\mathrm{Distr}_{\mathrm{VSS}}$ phase. Clearly we can also assume that D was not detected to be corrupted (i.e. the players did not decide that he shared a default value). Let V be an arbitrary player that remained honest until the end of $\mathrm{Distr}_{\mathrm{VSS}}$. First, we take a look at the $\mathrm{Distr}^{\mathrm{V}}_{\mathrm{VSS}}$ protocol (executed in Step 1.). What we will now show is that the value that V outputs at the end of $\mathrm{Recon}_{\mathrm{VSS}}$ is equal to s_{V} the secret of the commitment number $\mathrm{commit}^{s}_{\mathrm{D,V}}$. By the assumption that the dealer managed to cheat the verifier V. for at least $2/3$ of i's the value of $c_i \oplus r'_i$ is equal to s_{V}. Therefore s_{V} is in a clear majority (in Step 3. of $\mathrm{Recon}_{\mathrm{VSS}}$) and the output of V at the end of $\mathrm{Recon}_{\mathrm{VSS}}$ is equal to s_{V}.

Finally observe that from the construction of Step 2. of $\mathrm{Distr}_{\mathrm{VSS}}$ we can be sure that for every two verifiers $\mathrm{V}_0, \mathrm{V}_1$ (that remained honest) the value s_{V_0} is equal to s_{V_1}. Therefore they output the same value at the end of $\mathrm{Recon}_{\mathrm{VSS}}$. Moreover this value is equal to the value s' that the simulator chosen to be the input of the dealer in the $\mathrm{Distr}_{\mathrm{VSS}}$ phase. □

Combining Lemmas 4.15 and 4.16 we get the following.

Lemma 4.17 Protocol VSS securely evaluates VSS in a WSSZK-hybrid model, with an error negligible in k.

Let VSSReal be a composition $\mathrm{VSS}^{\mathrm{WSSZKReal}}$ (see Section 4.2). Combining Lemma 4.17 with Theorem 4.14 we get the following.

Theorem 4.18
Protocol VSSReal *securely evaluates* VSS*, with an error negligible in* k*.*

Clearly the complexity of VSSReal is polynomial in n,k and the complexity of the Secret Sharing scheme $(\mathrm{Distr}_{\mathrm{SS}}, \mathrm{Recon}_{\mathrm{SS}})$. Therefore we get the following.

Corollary 4.19 Theorem 1.1 (Page 12) is true.

4.4 The combinatorial lemmas

Hypergeometric distribution

Assume we are given an urn with N balls, M of which are red. Suppose we draw randomly (without replacement) n balls from this urn. Let $\mathcal{H}(M, N, n)$ be a random variable denoting the number of red balls drawn. The distribution of $\mathcal{H}(M, N, n)$ is called a *hypergeometric distribution* (see [DP98, Chv79]). By simple combinatorics we have

$$\Pr[\mathcal{H}(M, N, n) = i] = \binom{M}{i}\binom{N-M}{n-i}\binom{N}{n}.$$

Let $H(M, N, n, j)$ denote the probability that the number of red balls drawn is at least j. Clearly

$$\begin{aligned} H(M, N, n, j) &= \sum_{i=j}^{n} \Pr[\mathcal{H}(M, N, n) = i] && (4.8) \\ &= \sum_{i=j}^{n} \binom{M}{i}\binom{N-M}{n-i}\binom{N}{n}. && (4.9) \end{aligned}$$

By the result of [Chv79] we get that

$$H(M, N, n, j) \leq \exp(-2t^2 n) \qquad (4.10)$$

where $t = j/n - M/N$.

Let $G(M, N, n, j)$ denote the probability that the number of red balls drawn is *at most* j. Assume that the non-red balls (there are $N - M$ of them) are painted with green. Clearly $G(M, N, n, j)$ is equal to the probability that the number of green balls drawn is at least $n - j$. By the symmetry we get $G(M, N, n, j) \leq \exp(-2u^2/n)$ where

$$\begin{aligned} u &= (n-j)/n - (N-M)/N \\ &= M/N - j/n \\ &= -t. \end{aligned}$$

And therefore we get a bound identical to (4.10), namely

$$G(M, N, n, j) \leq \exp(-2t^2 n). \qquad (4.11)$$

The cut and choose lemma

Lemma 4.20 Suppose we are given a set $A = \{1, \ldots, 2k\}$ and a subset Bad of it, such that $|\mathrm{Bad}| > k/3$. Suppose we choose a random set $E \subset A$ of a size k. Let p be the probability that $E \cap \mathrm{Bad} = \emptyset$. Then p is at most $\exp(-k/18)$

Proof

Clearly p is equal to $G(|\mathrm{Bad}|, 2k, k, 0)$. Thus it is at most $\exp(-2(|\mathrm{Bad}|/2k)^2 k)$ which is smaller than

$$\exp\left(-2\left(\left(\frac{k}{3}\right)/2k\right)^2 k\right). \qquad (4.12)$$

Clearly (4.12) is equal to $\exp(-k/18)$ and thus we are done. □

The balls and bins lemma

Lemma 4.21 Assume we are given km balls, r of them are red and $km - r$ of them are green. Suppose we make the following experiment: we divide the balls randomly into m bins $B_1, \dots, B_m$, of k balls each. For $i = 1, \dots, m$ let r_i be the number of red balls in the ith bin. Let $q(k, m, r, w)$ be the probability that there exist two bins i' and i', such that $|r_i - r_{i'}| > w$. Then

$$q(k, m, r, w) \leq m(m-1)\exp(-w^2/(2k)) \tag{4.13}$$

Observe, that this bound does not depend on the number of red balls.
Proof
First we consider the case $m = 2$. We give a bound on the probability p that

$$r_1 - r_2 > w \tag{4.14}$$

Clearly by the symmetry we have $q(k, 2, r, t) = 2p$. Since $r_1 + r_2 = r$, then (4.14) is equivalent to

$$r_1 > (w + r)/2. \tag{4.15}$$

Now observe that the number of red balls in B_1 is distributed with the hypergeometric distribution from Section 4.4 (with $N = 2k$, $M = r$ and $n = k$). Thus the probability of (4.15) is equal to $\mathcal{H}(r, 2k, k, (w + r)/2)$, which (by (4.10)) is at most $\exp(-2t^2 k)$, where $t = w/(2k)$. Therefore p is at most $\exp(-w^2/(2k))$ and thus $q(k, 2, r, t) \leq 2\exp(-w^2/(2k))$.

Now, let us go to the general case $m \geq 2$. Fix some two bins B_i and $B_{i'}$. Let $p_{ii'}$ be the probability that $|r_i - r_{i'}| > w$. By the bound for $m = 2$ (and the fact that it does not depend on the number $r_i + r_{i'}$ of red balls) we get that $p_{ii'} \leq 2\exp(-w^2/(2k))$. Since the number of unordered pairs $\{i, i'\}$ is $m(m-1)/2$, by the union-bound we get

$$q(k, m, r, w) \leq \frac{m(m-1)}{2} \cdot 2\exp(-w^2/(2k))$$

what implies (4.13). □

Corollary 4.22 Suppose we are given a sequence of km bits $b_1, \dots, b_{km}$ (each $b_i \in \{0, 1\}$). Take a random permutation $\pi : \{1, \dots, km\} \to \{1, \dots, km\}$. For each $i = 1, \dots, m$ set $\mathrm{segment}_i = \{b_{\pi(i)}\}_{i=(t-1)k+1}^{tk+1}$. Let p be the probability that there exists a bit $b \in \{0, 1\}$ such that for every index i the number of bits b in $\mathrm{segment}_i$ is at least $k/3$. Then $p \geq 1 - m(m-1)\exp(-k/18)$.

Proof
If such a bit b does not exist, then there exists two indices i and i' such that

- the number of bits 0 in $\mathrm{segment}_i$ is at most $k/3$ and
- the number of bits 0 in $\mathrm{segment}_{i'}$ is greater than $2k/3$.

Let $\overline{p} = 1 - p$ be the probability of this event. Paint all the 0 bits in red and all the 1 bits in green. Call each $\mathrm{segment}_i$ a bin B_i. Then $\overline{p}$ is smaller than the probability that there exist two bins B_i and $B_{i'}$, such that the number of green bits in i is by $k/3$ greater than in i'. Thus, by Lemma 4.21, we get that $\overline{p} \leq m(m-1)\exp\left(-\left(\frac{k}{3}\right)^2/(2k)\right)$ and we are done. □

Lemma 4.23 Suppose we are given two sequences of bits $B^1 = b_1^1, \ldots, b_k^1$ and $B^2 = b_1^2, \ldots, b_k^2$ (each $b_i^j \in \{0,1\}$). For $j = 1, 2$ let bit b^j be such that at least $k/3$ of bits in B^j are equal to b^j. Set $b = b^1 \oplus b^2$. Take a random permutation $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$. Define $X_1 := \{\, i : b_i^1 = b^1 \,\}$ and $X_2 = \{\, \pi(i) : b_i^2 = b^2 \,\}$ and set $X = X_1 \cap X_2$. Let q be the probability that $|X| \leq k/18$. Then

$$q \leq \exp(-k/108) \tag{4.16}$$

Proof
Clearly

$$|Y_1| \geq k/3 \text{ and } |Y_2| \geq k/3 \tag{4.17}$$

If we now say that

- the balls are the indices $\{1, \ldots, k\}$,
- the red balls are the ones in Y_1,
- the drawn balls are the ones in Y_2,

then it is easy to see that $|Y|$ has a hypergeometric distribution (see Section 4.4) $\mathcal{H}(M, N, n)$ where $N = k$, $M = |Y_1|$ and $n = |Y_2|$. Therefore from (4.11) we get that

$$q \leq \exp(-t^2|Y_2|) \tag{4.18}$$

where $t = \left(\frac{k}{18}\right)/|Y_2| - |Y_1|/k$. By (4.17) we get that

$$\begin{aligned} t &\leq \left(\frac{k}{18}\right) / \left(\frac{k}{3}\right) - \left(\frac{k}{3}\right)/k & (4.19)\\ &= -1/6. & (4.20)\end{aligned}$$

Combining (4.17), (4.18) and (4.20) we get (4.16). □

Chapter 5

The MPC protocol

The MPC protocol was already informally defined in Section 1.1. In this section we present it's formal definition: we define MPC as an on-line protocol. The protocol works in a model presented in Section 1.3. Note, that here we are less general than in Chapters 4 and 5, since we work over a *threshold adversary structure*, namely the adversary can corrupt at most $t < n/2$ players (where n is the total number of players). We will assume that we are given a security parameter k and that our protocol works over some finite field $K = \mathrm{GF}(q)$, where $q > \max(n+1, 2^k)$.

Our definition is a generalization of a standard approach (see for example [BOGW88]) where MPC is defined as a secure evaluation of a function f given as an arithmetic circuit.

5.1 Specification

The MPC is an on-line protocol consisting of $m > 1$ phases (for some m). In each phase p one of the following protocols can be executed:

- an $\mathrm{VSS}^p_{\mathrm{MPC}}(\mathrm{D})$ protocol (where D is one of the players) — specified by a function $\mathsf{VSS}^p_{\mathsf{MPC}}(\mathrm{D})$ (such a phase will be called a *sharing phase*),
- an $\mathrm{Sum}^p_{\mathrm{MPC}}(a, b, c_0, c_1)$ protocol (where $a, b \in \{1, \ldots, p-1\}$) — specified by a function $\mathsf{Mult}^p_{\mathsf{MPC}}(a, b, c_0, c_1)$ (a *summing phase*),
- an $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$ protocol (where $a, b \in \{1, \ldots, p-1\}$) — specified by a function $\mathsf{Mult}^p_{\mathsf{MPC}}(a, b)$ (a *multiplication phase*), or
- an $\mathrm{Open}^p_{\mathrm{MPC}}(a)$ protocol (where $a \in \{1, \ldots, p-1\}$ — specified by a function $\mathsf{Open}^p_{\mathsf{MPC}}$ (an *opening phase*).

The input and output of the trusted party is a list $L \in (K \cup \{\texttt{nothing}\})^*$. The $\mathsf{VSS}^p_{\mathsf{MPC}}(\mathrm{D})$ function takes the following input:

- a value $s \in K$ — from player D, and
- a list $L \in (K \cup \{\texttt{nothing}\})^*$ of a length $p-1$ — from the trusted party.

The only output goes to the trusted party: it is a list $L \cdot s$.

Function $\mathsf{Sum}^p_{\mathsf{MPC}}(a, b, c_0, c_1)$ can be called only if a and b were not the opening phases. It takes no input and gives no output to the players. It's only input comes from the trusted party: it is a list L of a length $p-1$. The output of the trusted party is L with an attached element v defined as follows. Let s and u be ath and the bth element of L, respectively. Then $v = c_0 s + c_1 u$.

The definition of the function $\mathsf{Mult}^p_{\mathsf{MPC}}(a, b)$ is almost identical. The only difference is that v is defined to be equal to su.

We will assume that the $\mathsf{Open}^p_{\mathsf{MPC}}(a)$ function can be called only if the ath phase was a multiplying phase[1]. It takes no input from the players. It's only input is the list L from the trusted party (of a length $p-1$). Let s be the $(p-1)$th element of L. Then s is the output of every player and the output of the trusted party is $L \cdot \texttt{nothing}$

5.2 Implementation

We are now going to implement the MPC protocol. We do it in a hybrid model with an access to a trusted party computing the $\mathsf{IC}(\mathrm{P}_i, \mathrm{P}_j)$ (for every pair of players $\mathrm{P}_i, \mathrm{P}_j$). In order to make the protocol more readable we will describe it using the terminology presented in Section 3.2.8.

Definition 5.1
For n and t (such that $t < n$) vector $(s_1, \ldots, s_n)$ is called *t-consistent* if there exists a polynomial f of a degree at most t, such that for each $i \in \{1, \ldots, n\}$[2] we have $f(i) = s_i$. If $s = f(0)$ then we say that $s_1, \ldots, s_n$ *interpolate* s.

Note that the values of $s_1, \ldots, s_n$ are just the shares of s in the secret sharing of Shamir [Sha79]. Clearly the interpolation is possible also when less than $n-t$ values s_i are missing. We will also use the following two facts about bivariate polynomials.

Lemma 5.2 Let f_s be a random bivariate polynomial of a degree at most t in each variable, such that $f_s(0,0) = s$. Let $A \subset K$ be such that $|A| < t$ and $0 \notin A$. Then the values $\{f_s(i,j)\}_{i,j \in A}$ are distributed identically for every s.

Lemma 5.3 Let $B \subset K$ be such that $|B| \geq t$. Then for every set $\{s_{ij}\}_{i,j \in B}$ there exists at most one bivariate polynomial f of a degree at most t such that for every $i, j \in B$ we have $s_{ij} = f(i,j)$.

[1] This assumption is technical and made only for the security proof to be less complicated.
[2] Here $2, \ldots, n$ are some fixed elements of the field K.

The sharing phase

We start with $\mathrm{VSS}^p_{\mathrm{MPC}}(\mathrm{D})$. The protocol is based on the bivariate solution of Feldman [FM88, BOGW88] (omitting the need for error correcting codes). The intuition behind the construction is that the secret will be shared using an $n \times n$ matrix of values, where each row and column is t-consistent. The dealer will commit himself to all these values by signing each value in the matrix. Thus, if he did not act properly this fact would be exposed using the signatures. The consistency property will be verified by the players together. Hence we are guaranteed that all the values held by (yet) uncorrupted players are consistent and define a single secret[3]. In order to prevent the adversary from corrupting the secret at reconstruction time, we also require that each player sign all the values which he holds in a given row. And thus no new values can be injected into the computation in the reconstruction.

$\boxed{\mathrm{VSS}^p_{\mathrm{MPC}}(\mathrm{D})}$

1. The dealer D chooses a random bivariate polynomial $f(x,y)$ of degree at most t in each variable, such that $f(0,0) = s$. Let $s_{ij} = f(i,j)$. The dealer sends to player P_i the values $x_{1i} = s_{1i}, \ldots, x_{ni} = s_{ni}$ and $y_{i1} = s_{i1}, \ldots, y_{in} = s_{in}$. To each value x_{ji}, y_{ij} player D attaches a digital signature $\sigma_{x_{ji}}(\mathrm{D}, \mathrm{P}_i)$, $\sigma_{y_{ij}}(\mathrm{D}, \mathrm{P}_i)$.[4]

2. Player P_i checks that the two sets $x_{1i}, \ldots, x_{ni}$ and $y_{i1}, \ldots, y_{in}$ are t-consistent. If they are not t-consistent, P_i broadcasts these values with D's signature on them. If a player hears a broadcast of inconsistent values with the dealer's signature then D is disqualified and execution is halted.

3. For every pair of players $\mathrm{P}_i, \mathrm{P}_j$ the following is executed:

   $\boxed{\text{Pairwise checking protocol}}$

   (a) P_j sends x_{ij} and a signature $\sigma_{x_{ij}}(\mathrm{P}_j, \mathrm{P}_i)$ which he generates on x_{ij}, privately to P_i.

   (b) Player P_i compares the value x_{ij} which he received from P_j in the previous step to the value y_{ij} received from D. If there is an inconsistency then P_i broadcasts a complaint and a pair $y_{ij}, \sigma_{y_{ij}}(\mathrm{D}, \mathrm{P}_i)$.

   (c) If P_i complained in the previous step then P_j broadcasts a pair x_{ij}, $\sigma_{x_{ij}}(\mathrm{D}, \mathrm{P}_j)$.

4. If for some pair of players in Step 3. a player hears two broadcasts with signatures from the dealer on different values, then D is disqualified and execution is halted.

[3]So far, this results in a WSS which is secure against an adaptive adversary.

[4]Remember that (according to the terminology from Section 3.2.8) what we mean by saying this is that a signing phase of an IC protocol is executed.

For every i the values $y_{i1}, \ldots, y_{in}$ and $x_{1i}, \ldots, x_{ni}$ will be called the *shares* of the player P_i. Observe that if D remains honest then the values $x_{1i}, \ldots, x_{ni}$ interpolate some value S_i and the values $S_1, \ldots, S_n$ interpolate s. Therefore each S_i will be called an *implicit share of a player* P_i *(of a value* s*)*. Moreover observe that if P_i remained honest and the dealer did not get disqualified then for every j one of the following holds:

1. P_i knows a signature of P_j on a share y_{ij} $(= x_{ij})$, or
2. Player P_i complained in Step (3b) and then in Step (3c)
   (a) P_j has broadcasted the value y_{ij} with a valid dealer's signature on it, or
   (b) P_j has not broadcasted a properly signed value.

For simplicity we will assume that if case (2a) happened then this counts the same as P_j giving to P_i a signature for y_{ij} (clearly this signature can be now verified trivially, since y_{ij} is known publicly). If case (2b) occurred then P_j becomes publicly known to be corrupted. In this case we can safely assume that P_i knows the signature of P_j on every value that he wants (i.e. the verification procedure is void). Therefore later on we will assume that after the sharing phase for every y_{ij} player P_i knows a signature of P_j on it. This leads to the following definition.

Definition 5.4
A *proper sharing of a secret* s is a matrix of values $\{s_{ij}\}_{i,j\in\{1,\ldots,n\}}$, such that there exists a bi-variate polynomial f of a degree at most t in each variable, such that for every $i, j \in \{1, \ldots, n\}$ we have $f(i, j) = s_{ij}$ and $f(0, 0) = s$.

We will say that *the players received a proper sharing of a secret* s, if there exists a proper sharing $\{s_{ij}\}_{i,j\in\{1,\ldots,n\}}$, of a secret s and every honest player P_i received the values $x_{1i} = s_{1i}, \ldots, x_{ni} = s_{ni}$ and $y_{i1} = s_{i1}, \ldots, y_{in} = s_{in}$ (each y_{ij} is signed by a player P_j).

Clearly by Lemma 5.3 every proper sharing defines the secret s uniquely. We now claim the following.

Lemma 5.5 In every execution of $\mathrm{VSS}_{\mathrm{MPC}}(\mathrm{D})$, the honest players receive a proper sharing of some secret s'. If the dealer remained honest then s' is equal to his input value s.

Proof
The second claim follows directly from the construction of Step 1. of the protocol.

Now suppose that the dealer got corrupted. If the players disqualified the dealer then we are done. Otherwise we will show that there is a fixed value s' defined by the distribution. Define s' to be the secret which interpolates through the shares held by the set T of the first $t + 1$ players who have not been corrupted during the simulation of this phase. Their shares are trivially t-consistent, and there are correct signatures for their shares, and thus the value s' is well defined with an underlying implicit polynomial $f'(x, y)$.

Let us now look at another uncorrupted player outside the set T. He has corroborated his shares with all these $t + 1$ players and has not found an inconsistency with them. Thus, his shares interpolate (at the minimum) through $f'(x, y)$ and hence are at least t-consistent. But this player has also verified that his shares are t-consistent. Hence, when this player's shares are added to the initial set of players' shares the set remains t-consistent, thus defining the same secret s'. □

The summing phase and the multiplying phase

The general idea behind the implementation of the $\mathrm{Sum}^p_{\mathrm{MPC}}(a, b, c_0, c_1)$ and $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$ phases is as follows. We assume that the players received proper sharings in phases a and b of some secrets s and u respectively. We will show a protocol whose result is a proper sharing of a secret $v = c_0 s + c_1 u$ (in case of $\mathrm{Sum}^p_{\mathrm{MPC}}(a, b, c_0, c_1)$), or $v = su$ (in case of $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$). Clearly if in any of the phases a and b the players decided that the shared secret is equal to 0[5] then the protocols become trivial, so we do not need to be concerned about it.

The $\mathrm{Sum}^p_{\mathrm{MPC}}(a, b, c_0, c_1)$ phase is implemented in a non-interactive way. Let $x^a_{1i}, \ldots, x^a_{ni}, y^a_{i1}, \ldots, y^a_{in}$ be the shares of player P_i from phase a and let and $x^b_{1i}, \ldots, x^b_{ni}, y^b_{i1}, \ldots, y^b_{in}$ be the shares of player P_i from phase b. The new shares (for the current phase p) are obtained in the following way. For every $j \in \{1, \ldots, n\}$ the player P_i sets $y^p_{ji} := c_0 y^a_{ji} + c_1 y^b_{ji}$ and $x^p_{ij} := c_0 x^a_{ij} + c_1 x^b_{ij}$. By the linearity property the signatures for the new shares can be obtained from the signatures of the shares from the phases a and b.

The $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$ is slightly more involved. Let $S_1, \ldots, S_n$ and $U_1, \ldots, U_n$ be the implicit shares of s and u respectively. Let f_s and f_u be the respective polynomials. We apply the method from [GRR98]. This method calls for every player to multiply his implicit shares of s and u and to share the result of this using VSS (i.e. a protocol for the sharing phase). This results in proper sharings of $S_1 U_1, \ldots, S_n U_n$. Now observe that these values lie on a polynomial $f_s \cdot f_u$ (which is of a degree at most $2t$). Clearly $(f_s \cdot f_u)(0, 0) = su$. Thus a proper sharing of the result v can be computed as a fixed linear combination of the sharings of $S_1 U_1, \ldots, S_n U_n$ (i.e. each player computes a linear combination of his shares from the n VSS's). Since our sharing protocol is linear, like the one used in [GRR98], the same method will work for us, provided we can show that player P_i can share a secret V_i using $\mathrm{VSS}_{\mathrm{MPC}}(\mathrm{P}_i)$, such that it will hold that $V_i = S_i U_i$ and to prove that he has done so properly. Later we will show that if P_i fails to complete this process the players will always be able to reconstruct S_i and U_i and compute $S_i U_i$ openly.

[5]Recall that this happens when the dealer gets disqualified in a sharing protocol. It can also propagate further (for example if the dealer was disqualified in phase a then the result of every $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$ is also 0 by default)

In order to eliminate sub-indices let us recap our goal stated from the point of view of a player $\mathrm{Deal} = \mathrm{P}_i$. Let $S = S_i$ and $U = U_i$. For $j = 1, \ldots, n$ let s_j be equal to $f_s(j,i)$ and let u_j be equal to $f_u(j,i)$. Recall that each player P_j knows a signature of Deal on s_j and u_j. The goal for the player Deal is to share a value $V = SU$ (using the sharing protocol) and convince the other players that he did it correctly. He will do it using the $\mathrm{ProveMult}$ protocol presented below.

Before we go to the real protocol let us first present informally the main idea, which is very general and can be applied for an arbitrary linear commitment scheme[6] First, the dealer chooses some random value β, commits to it and to βU (let $\overline{\beta}$ and $\overline{\beta U}$ be the respective values he actually committed to). Then the players choose some random value r and ask the dealer to reveal the value of $rS + \overline{\beta}$ and to prove that $rV + \overline{\beta U} = (rS + \overline{\beta})U$ (by linearity he can do it without revealing any further information). We will show later (Lemma 5.7) that with high probability if the dealer managed to prove it, then $V = SU$. Let us now present the protocol $\mathrm{ProveMult}$ in detail.

$\boxed{\mathrm{ProveMult}}$

1. First, the dealer shares S. He uses the protocol $\mathrm{VSS}_{\mathrm{MPC}}(\mathrm{Deal})$ with a following constraint. The bi-variate polynomial f_S that he chooses in Step 1. is such that for every $i \in \{1, \ldots, n\}$ we have $f_S(i, 0) = s_i$. If this values do not agree for some i then (after the $\mathrm{VSS}_{\mathrm{MPC}}(\mathrm{Deal})$ is completed) player P_i complaints and broadcasts all the values that he got from Deal in this phase together with their signatures. Other players look at it and disqualify Deal if the complaint was justified.

   The same procedure applies to the value U.

2. Deal shares the value $V = SU$ using the protocol for an $\mathrm{VSS}_{\mathrm{MPC}}$ phase. If he is corrupted then he may share some other value. Denote it by $\overline{V}$ (clearly if Deal is honest then $\overline{V} = V$).

3. Deal chooses a random $\beta \in K$ and shares β and βU using protocol form an $\mathrm{VSS}_{\mathrm{MPC}}$ phase. Let $\overline{\beta}$ and $\overline{\beta U}$ denote the actual values that he has shared.

4. The players jointly generate, using standard techniques, a random value r, and expose it. We will comment more about this point later.

5. Applying the protocol for phase $\mathrm{Sum}_{\mathrm{MPC}}$ the players compute the sharing of $rS + \overline{\beta}$ and open it using the protocol for phase $\mathrm{Open}_{\mathrm{MPC}}$ (see Section 5.2.).

6. Applying twice $\mathrm{Sum}_{\mathrm{MPC}}$ the players compute the sharing of $rV + \overline{\beta U} - (rS + \overline{\beta})U$ and open it using $\mathrm{Open}_{\mathrm{MPC}}$. If the opened value is 0 then players accept. Otherwise they disqualify the dealer.

[6] The commitment scheme is *linear* if, whenever the dealer is committed to some values x_1, x_2, and x_3 then, for arbitrary constant c he is able to prove (without revealing any other information about x_1 and x_2) that $cx_1 + x_2 = x_3$ (if it is true).

Lemma 5.6 If the dealer remained honest then he does not get disqualified in the above procedure.

Proof
Clearly an honest dealer does not get disqualified until Step 6. Observe that if he remained honest then $\overline{\beta U} = \beta U$, $\overline{\beta} = \beta$ and $V = SU$. By the simple arithmetic the equation checked in Step 6. holds. $\Box$

Lemma 5.7 If $V \neq SU$ in the above protocol then, whatever are the actions of the adversary, the chances that the dealer did not get disqualified is at most $\frac{1}{|K|}$.

Proof
If the dealer does not get disqualified in Step 6. then it has to be the case that $r(V - SU) = \overline{\beta}U - \overline{\beta U}$. Suppose $V \neq SU$. Thus $V - SU \neq 0$ and we get $r = (\overline{\beta}U - \overline{\beta U})(V - SU)^{-1}$. Thus before Step 4. there exists exactly one value of r such that the dealer will not get disqualified in Step 6. Since $r \in K$ is chosen randomly in Step 4. the adversary succeeds with probability at most $1/|K|$. $\Box$

Let us now say what happens if some $\mathrm{Deal} = \mathrm{P}_i$ gets disqualified in the above procedure. Clearly the set of not-disqualified players is of a size at least $t + 1$. Now observe that since the procedure is executed for $\mathrm{Deal} = \mathrm{P}_1, \ldots, \mathrm{P}_n$, then at least $t + 1$ values out of $S_1, \ldots, S_n$ and $U_1, \ldots, U_n$ were shared successfully in Step 1. Thus the value of S_i can be obtained as a linear combination of those shares (since they lie on the polynomial f_S of a degree t). Therefore we can compute it (by applying $t - 1$ times $\mathrm{Sum}_{\mathrm{MPC}}$ and then $\mathrm{Open}_{\mathrm{MPC}}$). The same applies to U_i. Thus if P_i refuses to cooperate the players can compute T_iU_i openly and some default sharing of T_iU_i is taken by the players.[7]

Let us now say a few words about the generation of a challenge r in Step 4. This is done by a following simple protocol.

Generating a random number

1. Every player P_i chooses a random value $r \in K$ and shares it using a protocol from the phase $\mathrm{VSS}_{\mathrm{MPC}}$.
2. The players reconstruct the secrets using a protocol $\mathrm{Open}_{\mathrm{MPC}}$ (see Section 5.2.). The output is the sum of the reconstructed secrets.

Clearly the output is a random value. To get a better efficiency of the entire protocol we will use the following trick. Recall that $\mathrm{ProveMult}$ is executed n times in every invocation of $\mathrm{Mult}_{\mathrm{MPC}}$. Instead of executing them in a sequential way, we will execute them in parallel. Thus, instead of generating a challenge n times, it will be enough to generate it once. As a straightforward extension of Lemma 5.7. we get that the chances of the adversary to cheat without being disqualified in any of the invocations of $\mathrm{ProveMult}$ is at most $n/|K|$.

[7]Originally in [CDD+99] we were using a less efficient solution, namely we were instructing the players to restart the protocol, and to simulate the corrupted player P_i openly. The more efficient solution used here was suggested by Serge Fehr [Hir99].

The opening phase

The $\mathrm{Open}^p_{\mathrm{MPC}}(a)$ is implemented as follows. We assume that in phase a the players have received a proper sharing of some secret, what resulted in every player P_i holding the values $y_{i1} = s_{i1}, \ldots, y_{in} = s_{in}$ (each y_{ij} with a signature $\sigma_{y_{ij}}(\mathrm{P}_j, \mathrm{P}_i)$ from P_j).

$\boxed{\mathrm{Open}^p_{\mathrm{MPC}}(a)}$

1. Each player P_i broadcasts the values $y_{i1}, \ldots, y_{in}$, each y_{ij} together with a signature $\sigma_{y_{ij}}(\mathrm{P}_j, \mathrm{P}_i)$.

2. Each player P_i checks whether player P_j's shares broadcasted in the previous step are t-consistent and all the signatures are valid. If not then P_j is disqualified. Otherwise let Y_i be the value interpolated by the shares $y_{i1}, \ldots, y_{in}$.

3. All the values Y_i reconstructed in Step 2. are taken and interpolated to compute the secret s. Every player outputs s.

Lemma 5.8 Suppose that in some phases $a, b \leq p - 1$ the players received proper sharings of secrets s and u respectively. Then the following holds.

1. If the pth phase is $\mathrm{Mult}^p_{\mathrm{MPC}}(a, b)$ then in this phase the players receive a proper sharing of su, unless the adversary succeeded in cheating in one of the executions of $\mathrm{ProveMult}$.

   Moreover the bi-variate polynomial f defined by this shares is chosen uniformly at random (out of the polynomials of a degree at most t, such that $f(0, 0) = su$).

2. If the pth phase is $\mathrm{Sum}^p_{\mathrm{MPC}}(a, b, c_0, c_1)$ (for some $c_0, c_1 \in K$) then in this phase the players receive a proper sharing of $c_0 s + c_1 u$.

3. If the pth phase is $\mathrm{Open}_{\mathrm{MPC}}(b)$ then the value output by every (honest) player at the end of this phase is equal to u.

Proof of Point 1. The first claim follows from the remarks (and Lemmas 5.6 and 5.7) from Section 5.2.

The second claim follows from the fact that f is a sum of n polynomials out of which at least one was chosen uniformly at random. This is because at least one (actually, at least $t + 1$) player always remains honest.

Proof of Point 2. This point follows immediately from the linearity of polynomials: for every two bi-variate polynomials f and g of a degree at most t in each variable, and every two field elements $c_0, c_1 \in K$ a function $h(x, y) = c_0 f(x, y) + c_1 g(x, y)$ is also a polynomial of a degree at most t in each variable.

Proof of Point 3. Let f_u be the bi-variate polynomial defined by the proper sharing of u. Clearly for every i the (single-variate) polynomial $f_u(i,\cdot)$ has to be equal to the one interpolated by the shares $y_{i1},\ldots,y_{in}$. This is because the degree of $f_u(i,\cdot)$ is at most t, and for at least $t+1$ elements $j=1,\ldots,m$ (more precisely: for every j such that P_j remains honest) we have $f_u(i,j)=y_{ij}$. Thus we get that for every i such that P_i was not disqualified in Step 2., it is the case that $f_u(i,0) = Y_i$. Again: since there are at least $t+1$ (the number of honest players) and $f_u(\cdot,0)$ is of a degree t, the value s interpolated in Step 3. is equal to $f_u(0,0)$ (and thus equal to u).

□

5.3 Construction of the simulator

Again, we use the standard simulation (see Section 2.2). First, wlog let us assume that at least one player got corrupted.

Let us start with $\mathrm{VSS}_{\mathrm{MPC}}$. The simulator will stay in the first corruption stage until the protocol terminates. If it happened that the dealer D got corrupted then the simulator looks at the shares received by the players. By Lemma 5.5 they form a proper sharing of some secret s' (by Lemma 5.3 this secret is uniquely defined). Thus the simulator simply has to examine the shares of the honest players and interpolate the value of s'. He then goes to the computation stage, submitting s' as the input of D.

The simulation of $\mathrm{Mult}_{\mathrm{MPC}}$ and $\mathrm{Sum}_{\mathrm{MPC}}$ phases is straightforward since the input and output of the players is empty. The simulator simply stays in the first corruption stage until the protocol halts. Then he goes to the computation stage.

The simulation of $\mathrm{VSS}_{\mathrm{MPC}}$ goes as follows. The simulator goes immediately to the computation stage. In this way he learns the value of the secret s to be reconstructed in this phase (since we assumed that at least one player is corrupted) and updates the simulated protocol in a standard way. Then, he continues the simulation in the second corruption stage, until it halts.

5.4 Proof of security

Define ERROR to be the event that the adversary managed to cheat in some $\mathrm{ProveMult}$ subprotocol. By Lemma 5.7 (and the remarks after the implementation of $\mathrm{ProveMult}$) we know that the probability of ERROR is at most $nm/|K|$ (where m is the number of phases).

Lemma 5.9 The simulation described in Section 5.3 works secretly and correctly unless ERROR occurs.

Proof
The proof will go by induction on the number of phases p. We will actually prove a stronger claim, namely except of the secrecy and the correctness, we will also prove the following.

Invariant 1 *In every phase p (that was not an opening phase), the players received a proper sharing of some secret s_p. Moreover let L be the simulated output of the trusted party from the last phase. Then the pth element of L is equal to s_p.*

Now, take some phase p and assume that we know that until the phase $p-1$ the simulation works correctly (observe that correctness is a concern only in the opening phase) and secretly and Invariant 1 holds. Consider the following cases.

Sharing phase If the pth phase is $\mathrm{VSS}^p_{\mathrm{MPC}}(\mathrm{D})$ (for some player D) then, if the dealer D gets corrupted then the secrecy holds trivially. Otherwise the most important point is that the polynomial f in Step 1. is chosen uniformly at random (out of polynomials f such that $f(0,0)=s$). Therefore (by Lemma 5.2) as long as D remains honest the values received by the players in Step 1. do not depend on s. It is also easy to see that in Steps 2.–4. the corrupted players do not receive any more information then they did in Step 1. This finishes the secrecy part. The Invariant 1 follows from Lemma 5.5.

Summing phase If the pth phase is a summing phase then the secrecy holds trivially (since the protocol involves no interaction). The Invariant 1 follows from Lemma 5.8 (Point 2.) and the assumption that it held until phase $p-1$.

Multiplying phase If the pth phase is $\mathrm{Sum}^p_{\mathrm{MPC}}(a,b,c_0,c_1)$ then the only nontrivial case is the $\mathrm{ProveMult}$ subprotocol when the dealer remained honest. In this case it is not difficult to see that all the values coming to the corrupted players in Steps 1.—4. are random. In Step 6. the values that are broadcasted are a random sharing of value 0. Thus the secrecy holds. The Invariant 1 follows from Lemma 5.8 (Point 3.) and the assumption that it held until phase $p-1$.

Opening phase If the pth phase is $\mathrm{Open}^p_{\mathrm{MPC}}(a)$ then let s be the value output by the players in the ideal execution. By Invariant 1 we know that in phase a the players received a proper sharing of s. Also, since we assumed that a has to be a multiplying phase, we get (by Lemma 5.8, Point 1) that the sharing is chosen randomly (out of all the sharings of s). Therefore the values broadcasted in Step 1 do not reveal any information other than the value of s. The correctness follows easily from Invariant 1 and Lemma 5.8.

□

Therefore we get:

Lemma 5.10 Protocol MPC securely evaluates MPC in an IC-hybrid model, with an error probability $nm/|K|$ (where m is the number of phases and n is the number of players).

5.5 Putting things together

Let $\mathrm{MPCReal}$ denote the composed protocol $\mathrm{MPC}^{\mathrm{ICReal}}$ (where ICReal is defined in Section 3.2.5). Recall that we have set $K = \mathrm{GF}(q)$ (where $q > \max(n+1, 2^k)$). Thus, we get the following.

Theorem 5.11
For $t < n/2$ the $\mathrm{MPCReal}$ protocol securely evaluates MPC, with an error probability $m2^{-k+O(\log(n))}$ (where m is the number of phases, n is the number of players, t is the maximal number of corrupted players, and k is the security parameter).

Proof
But the remarks from Section 2.1.7 the error probability ξ of $\mathrm{MPCReal}$ is at most equal to $\delta + \epsilon$, where

- $\delta = nm/|K|$,
- ϵ equal to the error probability of each ICReal times the number of ICReal phases executed by MPC.

By Lemma 5.10 we have $\delta = nm/|K|$. It is easy to see that the number of IC-phases executed in each phase of MPC is polynomial in n. Thus, (by Theorem 3.11) we have $\epsilon = p(n)m/(|K|-1)$ (where p is some polynomial). Therefore we get

$$\xi \leq nm/|K| + p(n)(m/(|K|-1)).$$

Since $|K| = \max(n+1, k)$ we get $\xi = m2^{-k+O(\log(n))}$ as desired. □

Let us now calculate the complexity of $\mathrm{MPCReal}$ (using the facts about the complexity of ICReal from Section 3.2.6). Let m be the number of phases, n be the number of players and $k := \log(|K|)$. We get the following.

Sharing phase From Lemma 3.12 it easily follows that for every s_{ij} the number of messages sent during each sharing phase is $O(\log(|K|)n)$. Therefore the total message complexity is $O(\log(|K|)n^3)$.

Summing phase Clearly the message complexity of each summing phase is 0, since the phase is non-interactive.

Multiplying phase It is easy to see that for every execution of $\mathrm{ProveMult}$ the message complexity is at most c times the complexity of the sharing phase (where c is some fixed constant) plus the complexity of generating the random number in Step 4. The complexity of generating this number is equal to n times the complexity of a sharing phase. As argued before this can be amortized by executing the $\mathrm{ProveMult}$ in parallel. Since $\mathrm{ProveMult}$ is executed n times, the total complexity of each multiplying phase is $O(\log(|K|)n^4)$.

Opening phase It is easy to see that the message complexity of each opening phase is at most $O(kn^2)$.

It is easy to see that the complexity of the multiplying phase dominates. Therefore we get the following.

Theorem 5.12
The message complexity of $\mathrm{MPCReal}$ *is* $O(mkn^4)$*, where* m *is the number of phases,* k *is the security parameter, and* n *is the number of players.*

Originally in [CDD+99] (Theorem 2.) we had a complexity $O(mkn^5)$, instead of $O(mkn^4)$. The improvement here is due the the observation by S. Fehr (see Footnote 7 on Page 93).

Chapter 6

Impossibility of constructing MPC from SS

In this section we prove our impossibility result, Theorem 1.2, which states that there exist families $\mathcal{F}$ of $\mathcal{Q}_2$ adversary structures, such that no polynomial-time SS-oracle protocol computes f_{AND} securely against $\mathcal{F}$.

Before doing so, we first point out, as claimed earlier, that secure computation of linear functionals can be efficiently handled using black-box SS, both in the passive and active models. As a consequence, Theorem 1.2 can also be interpreted as an impossibility result essentially regarding secure multiplication, or equivalently, Oblivious Transfer.

In the passive case, the secure computation of linear functionals is trivial: each input bit b is split randomly into $b = b_1 \oplus b_2 \oplus \cdots \oplus b_n$, and b_i is given to player P_i. Each player then computes the desired linear function locally on the b_i's and publish the result. The global result is then the xor of the local results. Note that the black-box SS-scheme is not needed for this. In the active model, we can first establish a situation where the input bits and the b_i's are verifiably secret shared. The players then prove using general techniques that they performed their local computations correctly (see Section 4.3).

To prove Theorem 1.2, let us first recall the standard argument [HM97] showing the impossibility result when no SS-oracle is given. As mentioned, a $\mathcal{Q}_2$ adversary structure X is called *maximal* if there does not exist a $\mathcal{Q}_2$ adversary structure $X' \neq X$ such that $X \subset X'$. For the sake of contradiction suppose that for every maximal adversary structure $\mathcal{A}$ there exists a protocol which runs in time bounded by some polynomial, and which computes f_{AND} securely against $\mathcal{A}$. Since the number of maximal $\mathcal{Q}_2$ adversary structures is double-exponential and the number of polynomial-time protocols (represented as polynomial-size circuts) is single exponential then (by a counting argument) there must exist a protocol π computing f_{AND} securely against two different maximal $\mathcal{Q}_2$ adversary structures X and Y. This means that π is secure against $Z = X \cup Y$. By maximality of X and Y we have that Z is not $\mathcal{Q}_2$. Therefore

f_{AND} cannot be computed securely against it, and we have a contradiction.

In our case the situation is more difficult because the behavior of the players may depend on the oracle answers. Observe that when the SS-oracle is asked by a set of players A to reconstruct some secret then the protocol gets the information whether A is a member of the adversary structure. Thus we may assume that together with reconstruction request comes a query about a membership in the adversary structure and that together with the SS-oracle we have a membership oracle.

Therefore for two different adversary structures X and Y the same protocol may behave in two different ways if it happens to ask a membership query about a set in a symmetric difference of X and Y. Intuitively the biggest combinatorial difficulty of the proof is to show that there always exist two different maximal $\mathcal{Q}_2$ adversary structures A_S and A_T and a protocol δ working against both of them, such that δ will, with large probability, not ask a membership query about any set in a symmetric difference of X and Y.

More precisely, the proof proceeds as follows. It is enough to prove that for any polynomial $p()$ the collection of oracle protocols of size $p(n)$ cannot handle all maximal $\mathcal{Q}_2$ adversary structures on n players. So for the sake of contradiction suppose that there is a polynomial $p()$ such that for every set of players P of size n and every maximal $\mathcal{Q}_2$ adversary structure $\mathcal{A} \subseteq \mathcal{P}(P)$ there exists an SS-oracle protocol $\pi(\mathcal{A})$ of size at most $p(n)$ computing f_{AND} securely against $\mathcal{A}$. All such protocols can be specified by a polynomial number of bits, and hence the total number of such protocols is at most a single exponential in n. We will then show (in Section 6.1):

Lemma 6.1 Let $\pi(A)$ be defined as above. Then for every n large enough there exist two adversary structures $A_S, A_T \subseteq \mathcal{P}(P_n)$ such that

1. the size of the set of players P_n is $2n+2$,
2. $\pi(A_S) = \pi(A_T)$, and
3. $\pi(A_S)$ asks a membership query about a set in the symmetric difference of A_T and A_S with probability at most $2^{-1.5n}$. This probability is taken over all random choices made in the protocol, and over a random choice of the $2n+2$ input bits. Moreover, A_S contains a set A, and A_T a set B, such that $|A| = |B| = n+1$ and $A \cup B = P_n$

Before proving this, let us show how the existence of such A_T and A_S yields the contradiction. We will construct from the protocol $\pi_0 := \pi(A_S) = \pi(A_T)$ a new protocol for two players, Alice and Bob, with input bits b_A, b_B that will compute securely (and with negligible error probability) $b_A \wedge b_B$. This is well known to be impossible, even if only passive cheating occurs, and the honest Alice and Bob are allowed unbounded computing power.

Consider the sets $A \in A_S$ and $B \in A_T$ guaranteed by the lemma. We let Alice and Bob simulate an execution of π_0, where Alice controls the players in A and Bob those in B. Alice selects as input bits for players in A a random set of $n + 1$ bits such that the AND of all of them equals b_A. Similarly for Bob. Then we execute π_0, where Alice (Bob) executes the algorithms of players in A (B). Every message from a player in A to a player in B causes Alice to send the message to Bob, and vice versa.

Note that although efficiency of protocols plays a crucial role in proving the above lemma, we do not need to be concerned about efficiency at this point anymore, because we are now headed towards establishing a contradiction by building a protocol for a problem for which no protocol exists, even if unbounded computing is allowed. Hence we need no SS-oracle, we can use an arbitrary (inefficient) secret sharing scheme for $A_S \cup A_T$. Also, we may assume that Alice and Bob each have a list of the sets in A_T and A_S. They will use it to answer membership queries as follows: in most cases π_0 asks about a set which is in both A_S, A_T or in neither of them, so it is clear what the answer should be. In the unlikely event that the question is about a set in the symmetric difference, the protocol stops, we say it crashes. Alice and Bob use the result computed by π_0 as their output if the protocol finishes; if it crashes they let the output but be 1.

Observe that in case $b_A = 0$ or $b_B = 0$, the probability of a crash is at most $2^{-n/2+2}$: if, say, $b_A = 0$, we choose randomly between a set of at least 2^n inputs, namely all those inputs to players in A, where at least one bit is 0. These cases constitute at least a fraction 2^{-n-2} of the overall probability space, so even restricted to this case, the crash probability is at most $2^{-1.5k}/2^{-n-2} = 2^{-n/2+2}$. This immediately implies that Alice and Bob compute correctly $b_A \wedge b_B$ except with negligible probability in n. It also implies that privacy is satisfied: it is enough to argue that if $b_A = 0$, Alice learns a negligible amount of information about b_B. To see this, consider an idealized scenario, where there are no crashes and all membership queries are answered according to A_S. Then since π_0 is secure against A_S, it follows that whenever the players in A (alias Alice) have 1 or more zeros in their input, they learn almost no information about the inputs of players in B. However, the only difference between the actual protocol we specified for Alice and Bob and the idealized case is the crashes. And since crashes occur with negligible probability, it follows that Alice's view of the actual protocol is statistically indistinguishable from what she sees in the idealized case.

This completes the proof that Alice and Bob would be able to compute the AND function securely, and so we have our contradiction.

6.1 Proof of Lemma 6.1.

Let us now show the existence of A_S and A_T satisfying the conditions 1.–3. It will be enough to restrict ourselves to a certain class of maximal $\mathcal{Q}_2$ structures. For a given n we will construct a class $\mathcal{C}_n$ of $2^{2^{1.9n}}$ such structures as follows. Take a set of players P_n such that $|P_n| = 2(n+1)$. Define *split* to be a pair $(X, P \setminus X)$ such that $|X| = |P \setminus X| = n + 1$. Fix in an arbitrary way a set of splits SP_n that has a property that $(X, Y) \in \mathrm{SP}_n$ if and only if $(Y, X) \notin \mathrm{SP}_n$ (for example: fix a player $p_0 \in P_n$ and define SP_n to be a set of all splits $(X, P \setminus X)$ such that $p_0 \in X$).

For a technical reason we make a further restriction, and choose an arbitrary subset R_n of SP_n of a size $2^{1.9n}$ (we have that $|\mathrm{SP}_n|$ is $\Omega(2^{2n}/\sqrt{n})$ by standard combinatorics, therefore this operation is always possible).

Now observe that every subset $S \subseteq R_n$ determines a unique adversary structure A_S in the following way: a set of players $Z \subseteq P_n$ belongs to A_S if and only if one of the following conditions is satisfied:

- $|Z| < n + 1$,
- $(Z, P \setminus Z) \in S$, or
- $(P \setminus Z, Z) \notin S$.

Such an adversary structure will be called a *split structure*. We define $\mathcal{C}_n$ to be the set of all split structures A_S (where $S \subseteq R_n$). Clearly every split structure is a maximal $\mathcal{Q}_2$ structure.

To avoid to many subscripts we fix n. From now on we will consider only protocols running against the split structures (what we can safely assume because we are proving a negative result). Let prot be the set of all the protocols assigned to the set of all split structures by π (i.e. $\mathrm{prot} = \pi(\mathcal{C}_n)$). It is easy to see that now all the membership queries about the sets of a size at smaller than $n + 1$ are always answered positively. Similarly all queries about the sets of a size bigger than $n + 1$ are always answered negatively. The only queries which give some information about the adversary structure are the queries about the sets of the size exactly $n + 1$. Therefore we can now assume that instead of a membership oracle for A_S every protocol is given a membership oracle for S. This assumption simplifies a bit the notation and views the problem in a more abstract way.

Let t be the maximum of the expected number of queries asked by the protocols from the set prot (more precisely let $t = \max_{A \in \mathcal{C}_n}$(expected number of queries asked by $\pi(A)$ when it runs against the structure A)). Let $s = 2^{1.7n}$ (the choice is somewhat arbitrary, what matters is that s is much bigger than t, but much smaller that $|R_n|$).

Divide R_n into s blocks of equal size in an arbitrary way (this operation will always be possible for big enough n). Let $B_1, \ldots, B_s$ be the resulting blocks. We will say that a set X is *blinking in a block B_j* iff there exists a set Y such that all the following conditions are satisfied:

- the protocols assigned by π to X and Y are the same, i.e. $\pi(X) = \pi(Y)$
- $X \cap B_j \neq Y \cap B_j$, and
- $X \cap (R_n \setminus B_j) = Y \cap (R_n \setminus B_j)$.

The last two conditions mean in other words that X and Y differ on a set B_j and do not differ elsewhere. The intuition here is that the protocol $\pi(X)$ may have some difficulty in deciding if it is running against X or Y, since it must ask a membership query in B_j to find out.

Lemma 6.2 For every big enough n there exists a set blinking everywhere (i.e. there exists $S \subseteq R_n$ such that S is blinking for every block $B_1, \ldots, B_s$).

Proof
For every block B_j let N_j denote the family of sets *not* blinking for B_j. What we need to show is that $\cup_{j=1}^{s} N_j \neq \mathcal{P}(R_n)$. We will actually prove a stronger fact, namely

$$\sum_{j=1}^{s} |N_j| < 2^{2^{1.9n}}. \tag{6.1}$$

Fix an arbitrary B_j. Take an arbitrary set $Z \subset R_n \setminus B_j$. Now take the family $\mathcal{G} = \{ W \subseteq R_n : W \setminus B_j = Z \}$ (in other words $\mathcal{G}$ is a family of all sets whose projection on $R_n \setminus B_j$ is equal to Z). If two different sets in $\mathcal{G}$ were assigned the same protocol, then they would both blink in B_j, so it follows that the size of the family of sets in $\mathcal{G}$ that are not blinking in B_j cannot be bigger than the number $|\mathrm{prot}|$ of different protocols. Therefore after summing over all possible sets $Z \subset R_n$ we have that $|N_j| \leq |\mathrm{prot}| 2^{2^{1.9n}(s-1)/s}$. Since the choice of B_j was arbitrary we get that the left-hand-side of (6.1) is not bigger than $s|\mathrm{prot}|2^{2^{1.9n}(s-1)/s}$. Therefore to prove (6.1) is enough to show that

$$s|\mathrm{prot}|2^{2^{1.9n}(s-1)/s} < 2^{2^{1.9n}}$$

which is equivalent to

$$s|\mathrm{prot}| < 2^{\frac{2^{1.9n}}{s}} = 2^{2^{0.2n}} \tag{6.2}$$

The left hand side of (6.2) is single exponential in n and so for big enough n the inequality (6.2) (and hence (6.1)) holds. $\square$

Let now n be big enough that the blinking everywhere set exists. Let $S \in R_n$ be such a set. Thus for every block B_j there exists a set $\mathrm{blink}(B_j) \in \mathcal{C}_j$ such that

- $\pi(A_S) = \pi(\mathrm{blink}(B_j))$,
- $S \cap B_j \neq \mathrm{blink}(B_j) \cap B_j$, and
- $S \setminus B_j = \mathrm{blink}(B_j) \setminus B_j$.

Consider the runs of $\pi(A_S)$ against the adversary structure A_S and consider the probability distribution of these runs over a random choice of the input bits to the computation as well as the random coins used. For every B_j let $\mathrm{pr}(B_j)$ be the probability that $\pi(S)$ asks a query about some element in B_j. It is easy to see that $\sum_j \mathrm{pr}(B_j)$ is the expected number of queries asked by $\pi(A_S)$. Recall that this expected number of queries is polynomial in n, and hence it is, for all large enough n, smaller than s by a factor of at least $2^{1.5n}$. Therefore

$$\sum_{j=1}^{s} \mathrm{pr}(B_j) < \frac{s}{2^{1.5n}}$$

Thus the average value of $\mathrm{pr}(B_j)$ is at most $2^{-1.5n}$. Let B_l be such that $\mathrm{pr}(B_l) \leq 2^{-1.5n}$. In other words, with probability at least $1 - 2^{-1.5n}$ the machine $\pi(A_S)$ will never ask about any element in B_l. Therefore if we set $T = \mathrm{blink}(B_l)$ then the protocol $\pi(A_S)$ (which is by the way equal to $\pi(A_T)$) with a probability $1 - 2^{-1.5n}$ will not distinguish between A_S and A_T.

Chapter 7

Error Free Protocols and the Open Problems

In this thesis, we have dealt with the situation where a broadcast channel (in addition to the private ones) is available and access structures are $\mathcal{Q}_2$. It is known [HM97] that if the adversary structure is $\mathcal{Q}_3$ (no three sets in the adversary structure covers the player set) and no broadcast is given, then VSS and MPC with zero error probability is possible. Thus it is natural to ask if in this model we are given an *error free* SS scheme, can we build an *error free* VSS scheme with polynomially related efficiency?

We sketch here how to build an error-free commitment scheme. The construction requires a broadcast channel, however, such a channel can be simulated, given an efficient way to decide membership in the adversary structure (see [FM98]), and the secret sharing scheme we assume gives precisely such a decision procedure.

The commitment scheme works as follows: the committer shares his secret s to get shares $s_1, \dots, s_n$. He further shares each s_i to get sets of subshares $\{s_{ij}\}$. He sends $s_{1j}, \dots, s_{nj}$ to P_j and sends s_i plus the random bits used in sharing s_i to P_i. This allows both P_i and P_j to compute s_{ij}, so they can privately compare their values and ask the committer to publicly announce s_{ij} if there is a mismatch. If some P_i realizes that the committer is corrupt, P_i accuses the committer, who must then make public all data sent to P_i.

In the same way as in the VSS from [BOGW88], this will ensure that the committer is always either disqualified because there are too many complaints, or the honest players agree on all subshares they know.

To open the commitment, the committer will make s and the shares $\{s_i\}$ public. Also each P_i will make his share s_i and the subshares $\{s_{ij}\}$ public. Each P_j announces if he agrees with s_{ij} or not. The share announced by P_i approved if all players except a non-qualified subset agree with the subshares he claims. The opening is accepted, if and only if the shares claimed by the committer are consistent and agree with all approved shares announced by the other players.

Since all players who have remained honest agree on subshare values, all honest players will always have their shares approved. Therefore, if the honest players have inconsistent shares, the commitment cannot be opened convincingly. But if the honest players do have consistent shares, the $\mathcal{Q}_3$ property implies that the commitment can only be opened in one way. On the other hand, if the dealer remains honest, then the honest players will have consistent subshares of s_i, even if P_i is corrupt. Hence P_i can (again by the $\mathcal{Q}_3$ property) only have the correct s_i approved, so an honest committer can always open successfully.

Naturally, this commitment scheme can be used to build a VSS scheme based on zero-knowledge techniques as shown earlier. But this scheme will have a non-zero error probability. We do not know how to build error free VSS efficiently from error free SS for this scenario, and leave this as an open problem.

Bibliography

[Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[Bei96] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. Ph.d.-thesis, Technion, Haifa, 1996.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, Illinois, 2–4 May 1988.

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

[CCD88] D. Chaum, C. Crepeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th Annual Symp. on the Theory of Computing*, pages 11–19, New York, NY 10036, USA, 1988. ACM Press.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations with dishonest minority. In *Advances in Cryptology — Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326, 1999.

[CDD00] Ronald Cramer, Ivan Damgård, and Stefan Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 325–334. ACM, May 2000.

[CDD+01] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 262–279. Springer-Verlag, May 2001.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology — Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334, 2000.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *ACM Symposium on Theory of Computing*, pages 639–648, 1996.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th annual Symposium on Foundations of Computer Science, October 21–23, 1985, Portland, OR*, pages 383–395. IEEE Computer Society Press, 1985.

[Chv79] V. Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

[CvdGT95] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computations. In D. Coppersmith, editor, *Advances in Cryptology: Proceedings of Crypto '95*, volume 963 of *LNCS*, pages 110–123. Springer Verlag, 1995.

[DP98] Devdatt D. Dubhashi and Alessandro Panconesi. Concentration of measure for analysis of randomised algorithms. Manuscript available at http://www.cs.unibo.it/~pancones/papers.html, October 1998.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 148–161, Chicago, Illinois, 2–4 May 1988.

[FM98] M. Fitzi and U. Maurer. Efficient byzantine agreement secure against general adversaries. In Springer Verlag, editor, *Proc. of Distributed Computing (DISC '98)*, volume 1499 of *LNCS*, pages 134–148, 1998.

[Gen95] R. Gennaro. *Theory and Practice of Verifiable Secret Sharing*. Ph.d.-thesis, Massachusetts Institute of Technology, 1995.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — A completeness theorem for protocols with honest majority. In *Proc. 19th Annual Symp. on the Theory of Computing*, pages 218–229, 1987.

[GRR98] R Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold. In *Seventeenth Annual ACM Symposium on Principles of Distributed*

Computing, pages 101–111, Puerto Vallarta, Mexico, June 1998. ACM.

[Hir99] Martin Hirt. personal communication. Jul 1999.

[HM97] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 25–34, Santa Barbara, California, 21–24 August 1997.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structures. In IEEE, editor, *IEEE/IEICE Global Telecommunications Conference: conference record, Nov. 15–18, 1987, Tokyo, Japan [GLOBECOM Tokyo '87]*, pages 99–102. IEEE Computer Society Press, 1987.

[KW93] M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 102–111, San Diego, California, 18–21 May 1993. IEEE Computer Society Press.

[Nao91] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[Rab94] Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of the ACM*, 41(6):1089–1109, November 1994.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In ACM, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing: Edmonton, Alberta, Canada, August 14–16, 1989*, pages 73–85. ACM Press, 1989.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Yao82] A. Yao. Protocols for secure computation. In *Proc. 23th Annual Symp. on Foundations of Computer Science*, pages 160–164. IEEE Computer Society Press, 1982.