Diss. ETH No. 14376

Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting

A dissertation submitted to

ETH ZURICH

for the degree of
Doctor of Technical Sciences

presented by

Martin Hirt
Dipl. Informatik-Ing. ETH

born July 26, 1970, in Biel
citizen of Birrhard AG, Switzerland

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Birgit Pfitzmann, co-examiner

2001

Acknowledgments

First and foremost I would like to thank my advisor Ueli Maurer, who introduced me to this fascinating world of riddles and paradoxes, called "cryptography". He supported me extensively during my doctoral studies at ETH, advised me professionally and personally, and encouraged me whenever necessary.

Special thanks go to Matthias Fitzi and to Kazue Sako. Many results in this thesis would not be here without our endless discussions and the resulting cooperations. I also would like to thank all my co-authors, team mates, and colleagues: Daniel Bleichenbacher, Christian Cachin, Jan Camenisch, Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Serge Fehr, Clemens Holenstein, Olaf Keller, Reto Kohlas, Lennart Meier, Bartosz Przydatek, Tal Rabin, Renato Renner, Markus Stadler, Stefan Wolf, and all those that I forgot to mention.

Also, I would like to sincerely thank my co-referee Birgit Pfitzmann. She gave me comprehensive and constructive comments on all parts of this thesis, which helped much to improve the clarity and the integrity of the text.

Finally, I'm indebted to my family for their support during my studies, especially to my wife Berteline and to my little son Joshua.

Lastly, I'm grateful to ETH Zurich and Swiss National Science Foundation, who enabled my studies with their generous financial and administrative support.

Abstract

Secure multi-party computation allows a set of players to jointly compute an arbitrary function of their inputs, without revealing these inputs to each other. For example, the players can compute the average of their incomes in such a way that no player learns the income of any other player. Or, as a more applied example, a set of voters can compute the sum of their votes without revealing any particular vote.

The classical results in the literature state that n players can compute any function in such a way that any subset of up to $t < n/2$ players obtains no (zero) information about the other players' inputs, except for what can be derived from the public function output. If the bad players may deviate from the protocol and try both to obtain information about the other players inputs as well as to falsify the public output of the computation, then up to $t < n/3$ can be tolerated. Both bounds are tight.

The achievements of this thesis are three-fold: First, we investigate the efficiency of multi-party protocols. Especially those protocols that allow the bad players to deviate from the protocol are very inefficient. We show that protocols that tolerate misbehavior of the players are almost as efficient as protocols that require all players to follow the protocol correctly. The framework that allows this speed-up is generic and might be used for other distributed tasks as well.

Second, we generalize the existential results for multi-party computation. We show that one can tolerate certain collusions to be larger than the mentioned threshold t, at the cost that some other collusions of size t cannot be tolerated. This models well the fact that in the real-world, some players might be more likely to cheat than others. For various models, we give complete characterizations of the collusions that can be tolerated.

Third, we study secret-ballot voting, a particular application of multi-party computation. We are especially interested in the so-called "receipt-

freeness" property, which essentially means that voters should not be able to sell their right to vote. Receipt-freeness is known to be a subtle property, because one has to deal with voters who are willing to do anything for convincing the vote-buyer that they have submitted the requested vote. We propose a modular framework for such protocols, and construct two concrete receipt-free voting protocols that are more efficient than any receipt-free voting protocol in the literature.

Zusammenfassung

Sichere Multi-Party Berechnungen erlauben einer Menge von Spielern, gemeinsam eine beliebige Funktion von mehreren Inputs zu berechnen, ohne dass sich die Spieler gegenseitig die Inputs geben müssen. Zum Beispiel können die Spieler den Durchschnitt ihrer Einkommen bestimmen, ohne dass ein Spieler das Einkommen eines anderen Spielers erfährt. Oder eine Menge von Wählern kann die Summe ihrer Stimmen berechnen, ohne dass eine einzige Stimme bekannt wird.

Die klassischen Resultate in der Literatur besagen, dass n Spieler jede beliebige Funktion so berechnen können, dass eine beliebige Teilmenge von bis zu $t < n/2$ Spielern absolut keine Information über die Inputs der anderen Spielern erhält, ausser natürlich, was aus dem publiken Ergebnis der Berechnung gefolgert werden kann. Die Korrektheit des Ergebnisses und die Geheimhaltung der Inputs der ehrlichen Spieler kann selbst dann noch gewährleistet werden, wenn bis zu $t < n/3$ Spieler unehrlich sind und vom vorgeschriebenen Protokoll abweichen.

In dieser Dissertation betrachten wir drei Themenbereiche: Erstens behandeln wir die Effizienz solcher Multi-Party Berechnungen. Insbesonders jene Protokolle, die selbst aktiven Attacken von Spielern widerstehen, sind sehr ineffizient. Wir zeigen in dieser Arbeit, dass Protokolle, die robust sind gegen aktive Attacken, fast so effizient realisiert werden können wie die effizientesten nicht-robusten Protokolle aus der Literatur.

Zweitens verallgemeinern wir die bekannten Schwellenresultate aus der Literatur. Wir führen allgemeine Gegnerstrukturen ein und zeigen, dass gewisse Gegnermengen durchaus grösser als die erwähnte Schwelle t sein dürfen, falls dafür andere Gegnermengen kleiner als t sind. Solche allgemeinen Gegnerstrukturen modellieren zum Beispiel den Umstand, dass gewisse Spieler a priori mit grösserer Wahrscheinlichkeit betrügen

werden als andere Spieler. In diversen Modellen kann man mit allgemeinen Gegnerstrukturen auch echt mehr Betrüger tolerieren als mit den Schwellenresultaten.

Drittens betrachten wir das Problem von sicheren elektronischen Wahlen und Abstimmungen, einer der wohl populärsten Anwendungen von Multi-Party Berechnungen. Dabei konzentrieren wir uns vor allem auf das Problem von Stimmenverkauf und stellen uns die Frage, wie dieser verhindert werden kann. Dazu schlagen wir ein Framework vor, in welchem effiziente und sichere Wahlprotokolle konzipiert werden können, die Stimmenverkauf verhindern. Ausserdem entwickeln wir zwei konkrete Wahlprotokolle innerhalb dieser Umgebung, welche beide effizienter sind als alle bisherigen publizierten Wahlprotokolle.

Contents

Chapter 1

Introduction

Reykjavik, in the fishing harbour. Alice and Bob meet accidentally. Both don't know Reykjavik. Both are tourists. Bob is lost. He would like to ask Alice for the way to his hotel. Guesthouse Isafold. And maybe whether she would like to drink a hot chocolate with him. But he doesn't know her. And if she says no? "I would ask her, if only I knew that she would accept", he thinks. But he is shy. Too shy.

Alice is lost as well. She would like to ask Bob for the way to the youth hostel. And maybe whether Bob would not be willing to accompany her. It's already getting dark. She would of course then invite him for a cup of hot milk with honey. And some banana cake. In order to thank him. And maybe . . . who knows. But what if he says no? Should she dare to ask? "If I knew that he would not laugh at me, I would ask". But Alice is shy. Too shy.

They cross each other. Watching each other. Not asking each other. Finally, they both find their way. Bob to his guesthouse, Alice to the youth hostel. The wrong way. They will never meet again.

If only they would know the techniques of secure multi-party computation.

One of the famous problems in the area of distributed protocols is the so-called "dating problem": Alice and Bob want to find out whether they both want to date each other, but each of them does not want to tell whether s/he wants to date or not. A bit more formally, Alice and Bob both hold a bit x_A and x_B, respectively, and they want to compute the AND of the two bits. And of course, if one of them inputs 0, then s/he should not learn the bit of the other one.

This problem is a very simple instance of a general problem: How can a set of players (e.g., Alice and Bob) compute an arbitrary function (e.g., the AND) of their private inputs (e.g., x_A and x_B) in such a way that they all learn the result, but they all learn nothing additional about the other players' inputs? Surprisingly, this problem can be solved. The technique to solve is called *secure function evaluation (SFE)*.

There are many other attractive problems that are instances of this general paradigm: In Yao's millionaires problem, two millionaires want to find out which of them is richer, without revealing further information about the own fortune. In Chaum's spymaster problem, a set of spymasters want to find all double-spies, without revoking the anonymity of "honest" spies.

An even more powerful concept than SFE is *multi-party computation (MPC)*. Here, the players cannot "only" evaluate functions of their inputs, but they can enlist the assistance of a (fictive) trusted party. This trusted party helps the players in every possible aspect (where securely evaluating functions of the players' inputs is just one possibility). This trusted party of course does not really exist, rather it is *simulated* among the players in a secure way. In general, the players that simulate the trusted party need not be the same as the players that want to use it. Figure 1.1 illustrates the idea of simulation.

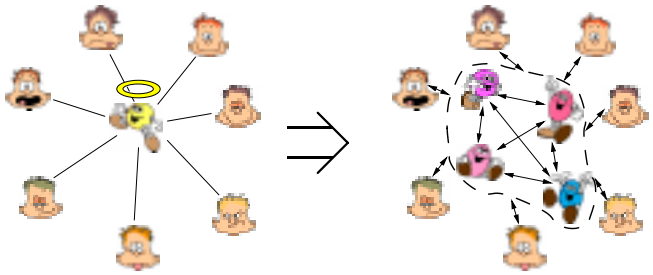


Figure 1.1: Simulation of a trusted party. The faces represent the players that want to cooperate, and the candies represent the players that simulate the fictive trusted party (with the nimbus).

One particular way of using this trusted party is to compute an agreed function of the players' inputs, as mentioned above: Every player sends his private inputs to the trusted party, who then evaluates the function

and hands the output to the authorized players. The goal of secure function evaluation is to imitate this scenario without need for a trusted party. SFE excludes reactive scenarios, where the trusted party is needed in several rounds of interaction, remembering some internal state. We refer to this latter case as *trusted party simulation (TPS).*

Multi-party computation has many practical applications. Using this technique, virtually any system involving a trusted party can be realized without need for such a party. As an example, todays stock markets heavily rely on the honesty and independence of some central service, which every investor must trust. By using multi-party computation, the functionality of this service can be distributed among a set of players (e.g., among the investors themselves), and thereby the strong trust assumption can be weakened substantially.

One of the most prominent applications is *secret-ballot voting.* Voting can be seen as a special case of secure function evaluation. The input of each player is his vote, and the function to be computed is the sum of the votes. First solutions for secure electronic voting have been presented years before first solutions for general MPC were found. Maybe one should rather say that MPC is the generalization of voting, instead of saying that voting is a special case of MPC. And indeed, many new research threads and techniques were first proposed for voting schemes, and later adapted and generalized for MPC.

Surprisingly, multi-party computation even provides an ultimate (yet impractical) solution for the problem of software piracy: The manufacturer's interest is not to give the software to the client (to prevent the pirate from copying the software), and the client's interest is not to give his private documents to the vendor. Although these interests apparently contradict each other, they both can be realized with a secure multi-party computation: The fictive trusted party takes the program from the vendor, and then reactively takes key strokes and mouse clicks from the client and provides him with screen shots from the software. Formally, the trusted party evaluates the universal program, where the software of the vendor and the user activities of the client are inputs, and the screen shots are outputs. Obviously, simulating this trusted party among the vendor and the client would in practice be horribly inefficient.

1.1 Secure Multi-Party Computation

In this section we give a short introduction of the problem of secure multi-party computation. We start by enumerating the various models and parameters used for MPC. With this formalism, we can elaborate more precisely what a MPC protocol is at all. Finally, we summarize the most important literature on this topic and mention some related work.

1.1.1 Models for MPC

1.1.1.1 The communication model

The players communicate with each other over *channels*. Virtually all protocols assume the existence of *pairwise* channels among the players. Often, also broadcast channels are assumed. A *broadcast channel* is a channel from one player to all other players (or to a subset of the players), where it is guaranteed that all recipients receive the same data. The topology of the network of channels can be *complete* or *incomplete*, i.e., the connectivity can be limited.

These channels can be assumed to be *secure* (authentic and secret), *authentic* (but tappable), or *insecure*. Furthermore, every channel is either *synchronous* or *asynchronous*. Synchronous means that the delay of messages in the channel is bounded by a known constant.

The model with pairwise synchronous secure channels among every pair of players is called the *secure-channels model*.

1.1.1.2 The adversary model

The dishonesty of players is modeled by a *central adversary* that corrupts players. We distinguish three corruption modes: A *passively corrupted* player gives all his internal data to the adversary, but continues executing the instructions of the protocol. An *actively corrupted* player is under full control of the adversary and misbehaves in arbitrary manner. A *fail-corrupted* player follows the protocol instructions till the adversary instructs the player to crash; from then on, the player does not send any message to any other player. Note that a fail-corrupted player does not give his internal data to the adversary, unless he is passively corrupted at the same time.

The adversary model defines which players can be corrupted in which mode. In the previous multi-party literature, the adversary is limited to either passively corrupt a certain number of players, or to actively corrupt a certain number of players. Accordingly, we speak of a *passive t-adversary* or an *active t-adversary*, where t denotes the maximum number of corruptions.

Furthermore, one must define the point in time when the adversary is allowed to corrupt players. A *static* adversary must perform all corruptions before the protocol execution, i.e., the set of corrupted players is fixed (but typically unknown) during the whole computation. More generally, the adversary may be allowed to corrupt players during the protocol execution, depending of information gathered so far. Such an adversary is called *adaptive* or *dynamic*. Recently, *mobile* adversaries were also considered. Like an adaptive adversary, a mobile adversary can corrupt players at any time, but he can also "release" corrupted players, regaining the capability to corrupt further players. The concept of a static adversary models the scenario where some of the players are (per se) dishonest, and adaptive corruption rather models the scenario of buying (or coercing) players. Mobile adversaries correspond for example to virus attacks.

Finally, the adversary model also determines the adversary's computing power. Most common assumptions are that the adversary is either *unlimited*, or is *computationally bounded* in a polynomial in a security parameter. For specific protocols, also *storage-bounded* adversaries were considered.

1.1.1.3 The computation model

A secure multi-party computation must be specified in some specification language. Many protocols in the literature are apparently restricted to secure function evaluation, and they require that the function is specified as a circuit over a fixed *finite field* $(\mathbb{F}, +, *)$. The circuit consists of addition gates, which take two inputs and output the sum, and multiplication gates, which take two inputs and output the product (i.e., fan-in is 2). The output of a gate can be connected to the input of any number of gates (i.e., fan-out is unlimited). Note that *any* computable function can be expressed as such a circuit, so this requirement does not limit the functions that can be computed. Some protocols in the literature require that the circuit is over a *Boolean field*, i.e., $|\mathbb{F}| = 2$, other protocols allow (or even require) larger fields.

Other MPC protocols explicitly support more general specifications with involving a fictive trusted party (TPS, reactive MPC). The values held by the trusted party are taken from a finite field $(\mathbb{F}, +, *)$, and the party can send and receive field elements, and can perform arbitrary computations (e.g., expressed as a sequence of gates) on known field elements.

Note that most (but not all) protocols in the literature that consider only SFE can easily be extended to capture TPS.

1.1.2 The Security of MPC

The goal of secure multi-party computation is to obtain a protocol among the players, which achieves essentially the same as would be achieved with using a trusted party. More formally, we consider a *specification* as a protocol among the players and a (fictive) trusted party, and we want to construct a protocol which does not involve the trusted party, but behaves like the specification. The adversary may interact in the protocol, complying with the adversary model. We require that whatever any admissible adversary can achieve in the MPC protocol, he could also achieve in the specification with the trusted party (e.g., he can make the corrupted players input wrong values, forget their outputs, etc, but nothing beyond). In other words, the adversary has no advantage in the protocol with the simulated trusted party compared to a protocol with a real trusted party. When this holds in an absolute sense, we say that the protocol is *perfectly secure*. When the adversary is allowed to have an advantage with some negligible probability, then we say that the protocol is *unconditionally secure*. When the adversary has no advantage only in case he is computationally bound, then the protocol is called *cryptographically secure*.

1.1.3 Generic Approach for MPC

There is a general and natural approach for MPC, which is used by almost all MPC protocols in the literature. This approach is based on *secret sharing*. A secret-sharing scheme describes how a value (a secret) can be split into pieces (shares), such that any small-enough set of pieces gives no information about the secret, but any large-enough set of pieces uniquely determines the secret. Such a sharing scheme allows a player (the dealer) to distribute a secret among the players in such a way that the adversary

(which can corrupt only few players) does not learn anything about the secret, but still enough players together can reconstruct it.

The key idea for realizing an MPC protocol is that every value held by the trusted party is instead kept secret-shared among the players. When a player is to send a value to the trusted party, then instead the player secret shares the value among all players, i.e., gives a piece of this value to each player. The computation that the trusted party should perform is then realized as a protocol among the players, where from two sharings a sharing of the sum (or the product) is obtained. Finally, when the trusted party should send a value to a particular player, then the secret is reconstructed for that player, i.e., every player sends his piece of the output to the designated receiver.

1.1.4 History and Literature

The problem of general-purpose multi-party computation was first suggested by Yao [Yao82]. As a first general solution, Goldreich, Micali, and Wigderson [GMW87] presented a passively secure protocol that allows n players to securely compute any given function even if a passive adversary corrupts any $t < n$ players, and an actively secure protocol that tolerates an active adversary corrupting any $t < n/2$ of the players. The security of the protocols is cryptographic, that is the adversary is assumed to be polynomially bounded. Chaum, Damgård, and van de Graaf [CDG87] improved the bound for the active model in the sense that the input of one player can even be information-theoretically hidden. Galil, Haber, and Yung [GHY87] considered efficiency and several corruption types in the cryptographic model. Ben-Or, Goldwasser and Wigderson [BGW88] proved that in the secure-channels model without broadcast, perfect security for n players can be achieved even if the adversary can corrupt any set of less than $n/2$ players (passive case) or, alternatively, any set of less than $n/3$ players (active case). These bounds are tight. The same results were obtained independently by Chaum, Crépeau and Damgård [CCD88] in an unconditional model with exponentially small error probability. The bound for the active model was improved by Rabin and Ben-Or [RB89] and independently by Beaver [Bea91b] by assuming a broadcast channel and tolerating a negligible error probability. They proposed protocols that provide unconditional security against an active adversary that may corrupt any $t < n/2$ of the players. Combining the advantages of unconditional security (against an adversary that corrupts a certain fraction of the players) and cryptographic security (against an

adversary with limited computing power), Chaum [Cha89] presented a protocol which provides unconditional security against an adversary that is limited by the number of players he can corrupt, and provides cryptographic security against an adversary who may corrupt any number of players.

The types of tolerable adversaries have recently been generalized in a number of directions: [OY91, CH94] consider mobile adversaries, [CFGN96] give adaptively secure protocols, and [CG96] consider incoercibility. Various minimality and complexity criteria are considered in [Kus89, BB89, Bea89, FY92, FKN94, Rab94, CGT95, CKOR97]. Asynchrony was studied in [BCG93, Can95].

Excellent overviews on various aspects of multi-party computation are given in the theses of Franklin [Fra93] and Canetti [Can95].

1.1.5 Related Work

There are two lines of research which are closely related to general MPC. One is the research on secret-sharing schemes, and the other one is the research on broadcast protocols.

1.1.5.1 Secret Sharing

Secret-sharing is one of the central ingredients of almost any MPC protocol. First secret-sharing schemes have been reported independently by Blakley [Bla79] and Shamir [Sha79]. Secret-sharing that is secure with respect to active corruption was proposed by Chor, Goldwasser, Micali, and Awerbuch [CGMA85]. Ito, Saito, and Nishizeki [ISN87] and Benaloh and Leichter [BL88] introduced the notion of general (non-threshold) access structures for secret sharing for the passive model, and Gennaro [Gen96] extended it to the active model with broadcast (and error probability).

1.1.6 Broadcast

A very important primitive for realizing MPC protocols is *broadcast* or *Byzantine agreement*. A broadcast protocol is a protocol that allows a dealer to send an arbitrary message to all players, where the protocol ensures that all players receive the same message, even if the dealer is malicious. The first broadcast protocol was proposed by Pease, Shostak, and

Lamport [PSL80, LSP82], but this protocol had running time exponential in the number of players. More efficient broadcast protocols were presented in [DFF+82, FM88, BGP89, CW89]. With unconditional security (and pairwise synchronous secure channels), broadcast can be achieved if and only if up to $t < n/3$ of the n players misbehave. In the cryptographic model, up to $t < n/2$ players (with synchronous channels) or up to $t < n/3$ players (with asynchronous channels, [Can95, CKS00]) may misbehave.

In the active model with broadcast (where MPC is possible for $t < n/2$), broadcast must either be simulated with a cryptographic broadcast protocol, and the unconditional security of the MPC protocol is lost. As an alternative, there exist broadcast protocols with unconditional security for $t < n/2$, under the assumption that in an *initialization phase* broadcast is available [BPW91].

1.2 Voting

Secret-ballot voting is one of the most attractive applications of multiparty computation. A secure voting scheme guarantees the correctness of the tally while preserving the privacy (and independence) of the votes. The function to be securely computed is the sum of the votes.

1.2.1 Models for Voting

Most voting schemes in the literature are for the cryptographic model with insecure channels. Additionally, it is common to assume a *bulletin board*. A bulletin board is a public storage of data, where every participant can write to (into his own section), but nobody can delete from. The bulletin board can be considered as an authenticated broadcast channel with memory.

Some voting protocols additionally assume *anonymous channels*, where the recipient of a message does not know (and cannot find out) who is the sender.

Voting protocols can also be categorized according to the kind of vote they support. Some voting schemes are restricted to yes / no-votes. More general protocols support 1-out-of-L votes, or even K-out-of-L votes, where every voter may vote for any K candidates out of a list of L candidates. Some schemes capture also more general settings (e.g., each voter

can assign five votes, where to each candidate up to two votes can be assigned, except for those candidate that are currently in office, they may receive up to three votes).

1.2.2 Security Requirements

The security of voting schemes can (and should) be defined analogously to the security definition of MPC, namely with help of a fictive trusted party, which is to be simulated by a protocol. In the specification, each voter sends his vote to the trusted party, who then selects those votes that are valid, adds them up, and publishes the tally. A voting protocol is secure if the adversary cannot achieve more than what he could achieve in this specification.

This kind of security definition is rather unusual in the voting research community. The standard approach to define security of a voting scheme is to give a list of properties that must be satisfied. For conformance with most of the literature, we follow this approach as well.

In the following, we give a list of common security requirements for secure voting schemes:

- SECRECY. It is infeasible to find out which voter has submitted which vote. Either the votes are never seen in clear, or they are available in clear, but it is unknown which vote belongs to which voter. Secrecy should also be satisfied for partial information on votes, as well as for relation between votes of several voters.
- ANONYMITY. It is infeasible to find out whether or not a particular voter has participated the vote. Note that this requirement can hardly be achieved by electronic voting schemes, unless some physical or organizational assumptions are taken.
- ELIGIBILITY. Only entitled voters are able to submit a vote (respectively, the votes of unauthorized voters are not counted).
- NO DOUBLE-VOTING. Every entitled voter can cast only one single vote.
- VALIDITY. Only valid votes are counted, e.g., “yes” and “no” votes. This property is of particular importance when votes are represented as number (e.g., 0 for “no” and 1 for “yes”), and the tally is the sum of votes.

- CORRECTNESS. The tally that pops up at the end of the vote is the correct sum of all valid votes.
- LOCAL VERIFIABILITY. Every voter can verify whether his vote is included in the published tally.
- GLOBAL VERIFIABILITY. Anyone can verify that all valid votes have been counted, and that the published tally is correct.

An additional requirement that prevents vote-selling will be discussed in Section 1.2.4.

1.2.3 Approaches

Apparently, the secrecy of the votes contradicts the ability of adding up votes. There are two approaches to reconcile these requirements: In the first approach, all ballots are published (and tallying is trivial). Secrecy is achieved by hiding which ballot belongs to which voter. This is either achieved with using a *mix-net*, where the set of ballots of all voters is shuffled consecutively by a set of mixers, or by assuming *anonymous channels*. In the second approach, the ballots are published only in an encrypted form (and the secrecy of the ballots is obvious). A special kind of encryption function, so-called *homomorphic encryption*, allows to add up the encrypted ballots without decrypting them, and only the sum is decrypted.

1.2.3.1 Voting schemes based on mix-nets

In mix-net voting schemes, the ballots of all voters are shuffled through a mix-net, such that afterwards, it is unclear which ballot belongs to which voter. More precisely, the voter encrypts his vote consecutively with the public key of each mixer, then all these multi-encrypted votes are sequentially decrypted and permuted by each mixer. The output of the last mixer is a shuffle of the votes in clear, but in random order. These votes can then be tallied publicly. In order to guarantee correctness of the tally, every mixer must additional publish a proof that he did not modify or substitute votes. This process is illustrated in Figure 1.2.

For the first mixer, the secrecy of the votes is based on the infeasibility of decrypting, and with each mixer, the level of encryption is decreased, but the level of anonymity of the voter is increased. The correctness is

based on the soundness of the public proofs. Because at the end all votes are known in clear, invalid votes can be discarded easily.

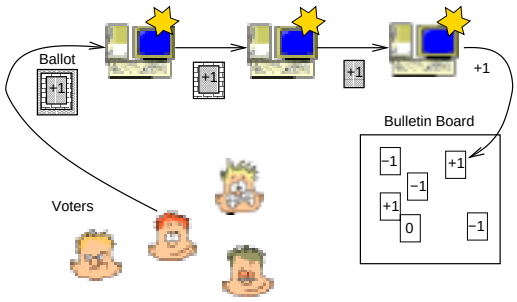


Figure 1.2: A mix-net voting scheme with three mixers. Each mixer decrypts and permutes a collection of multi-encrypted votes, and hands it to the next mixer. The asterisk stands for the public proof of correct mixing.

1.2.3.2 Voting schemes based on blind signatures

This approach can be seen as an abstraction of mix-net voting schemes. Here, each voter casts his vote unencrypted, but anonymously. The security of this scheme is based on hiding which vote belongs to which voter. Additional efforts are needed to ensure that only entitled voters can cast a vote: Each voter is to get a blindly signed public key from a registration authority, with which he signs his vote and sends it over an anonymous channel to the bulletin board. This approach is illustrated in Figure 1.3.

1.2.3.3 Voting schemes based on homomorphic encryption

A homomorphic encryption scheme supports addition of ciphertexts without knowledge of the secret key. With this primitive, one can construct very efficient voting protocols: The voter encrypts his vote with a homomorphic encryption scheme and posts it to the bulletin board. The authorities can tally the encrypted votes and decrypt the sum (see Figure 1.4). In addition, each voter must prove the validity of the submitted vote. In this approach, it is clear which (encrypted) vote belongs to which voter, and the secrecy of the votes is based on the infeasibility of decrypting votes.

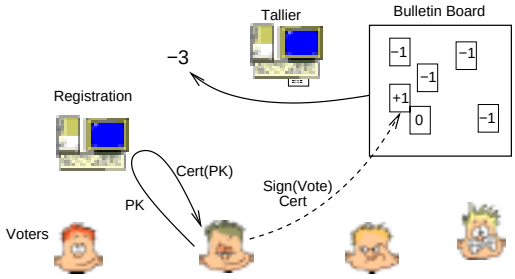


Figure 1.3: A voting scheme based on blind signatures. Each voter obtains a certificate from the registration authority for a temporary public key, signs his vote with the corresponding secret key, and send it together with the certificate over an anonymous channel to the bulletin board.

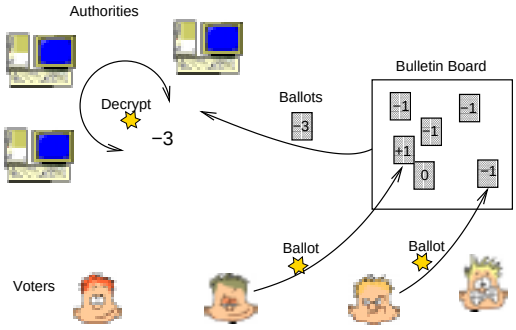


Figure 1.4: A voting scheme based on homomorphic encryption. The (encrypted) tally can be publicly computed given the (encrypted) votes. The asterisks stand for the validity proofs and the decryption proof.

1.2.3.4 Comparison of the approaches

Each approach has its advantages and disadvantages. We summarize the main parameters in Table 1.1. We emphasis that for both mix-net and blind-signature schemes write-in votes are possible, but not for homomorphic schemes. On the other hand, in contrast to the other approaches, homomorphic voting protocols allow incremental tallying (the encrypted votes can be added as soon as they arrive), which allows publication of

the tally much faster than with the other approaches.

	mix-net	blind signatures	homomorphic encryption
mathematical structure	medium	none	much
flexibility	write-ins	write-ins	inflexible
voter-interactivities	≥ 1 rounds	> 1 rounds	1 round
incremental tallying	no	no	yes
costs			
voting phase	medium	small	high
tallying phase	medium	very small	small
verification phase	high	local only	small

Table 1.1: Summary of the main advantages and disadvantages of several approaches to secret-ballot voting.

1.2.4 Receipt-Freeness

An important concept that was neglected so far is vote-buying, respectively vote-selling. A secure voting scheme should disable the voters from selling their vote. It is inevitable that a voter can accept money for promising his vote (but this can also be considered as a strong kind of propaganda), but at least the voting scheme should prevent that the voter receives the money only in case he keeps his promise. In other words, the voting scheme should not give the voter any mean to prove which particular vote he has cast. Voting protocols that disable the voter from proving the cast vote are called *receipt-free*.

Receipt-freeness is not well understood. There are many flavors and variants, and no clear definition is known. This might also be one reason why many proposed receipt-free voting protocols later turn out to be insecure, or at least to have some security-relevant weaknesses.

1.2.5 Previous Work

Secret-ballot voting protocols were first proposed by Chaum [Cha81], based on the idea of a mix-net. Cohen (Benaloh) and Fischer [CF85] suggested the approach with homomorphic encryption. The first voting scheme based on blind signatures and anonymous channels was proposed by Fujioka, Okamoto, and Ohta [FOO92]. Later, many schemes based on these approaches were published [BY86, Ive91, PIK93, Sak94, SK94, CFSY96, CGS97].

The concept of receipt-freeness was first introduced by Benaloh and Tuinstra [BT94]. Based on the assumption of a voting booth that physically guarantees secret communication between the authorities and each voter, they first proposed a single-authority voting protocol which, while being receipt-free, fails to maintain vote secrecy. Then they extended this protocol to a multi-authority scheme which does maintain secrecy. However, we show that this scheme is *not* receipt-free, in contrast to what is claimed in the paper (cf. [HS00]).

Another receipt-free voting protocol based on a mix-net channel was proposed by Sako and Kilian [SK95]. In contrast to [BT94], it assumes only *one-way* secret communication from the authorities to the voters. The heavy processing load required for tallying in mix-net schemes, however, is a significant disadvantage of this protocol. Furthermore, this scheme is vulnerable to the so-called randomization attack [Sch99]: The coercer can force a voter to vote randomly. The relevance of this attack depends on the vote policies; in certain settings, the attack might be significantly more powerful than coercing the voter not to vote at all (this kind of coercion cannot be prevented anyway). See also [MH96] for other security-relevant remarks and improvements of this protocol.

Later, a receipt-free voting scheme using blind signatures was given by Okamoto [Oka96]. Here, the assumption was that of *anonymous* one-way secret communication from each voter to each authority. Achieving communication that is both secret *and* anonymous would, however, be extremely difficult. Also, this scheme requires each voter to be active in *three rounds* (authorization stage, voting stage, and claiming stage), which is a big disadvantage in many realistic settings. The receipt-freeness property of the first scheme was broken, and a modified protocol was proposed in [Oka97].

Finally, a receipt-free voting protocol based on homomorphic encryption was presented in [HS00]. This protocol is more efficient than other

receipt-free protocols. However, also this protocol is vulnerable to the randomization attack [Sch99].

Another stream of research which relates to receipt-freeness is incoercible multi-party computation. Without any physical assumption, deniable encryption [CDNO97] allows an entity to lie *later* about how the ciphertext decrypts, and this technique is used to achieve incoercible multi-party computation [CG96]. However, the concept of incoercibility is weaker than receipt-freeness. It would allow a voter to lie about his vote, but it cannot help against a voter who *wants* to make his encryption undeniable and is willing to deviate from the protocol, and hence it cannot prevent vote-buying.

1.3 Contributions of this Thesis

1.3.1 Mixed Models for MPC

We consider unconditionally secure multi-party computations for the secure-channels model, where the adversary has unlimited computing power. In the literature, protocols for both the passive model and for the active model were known, and tight bounds for both models were proven. In the passive model, secure MPC is possible if and only if at most $t < n/2$ of the n players are corrupted, and in the active model, secure MPC is possible if and only if at most $t < n/3$ players are corrupted. If we assume existence of authentic broadcast channels, then the necessary and sufficient condition in the active model is that $t < n/2$.

We unify the classical passive model and active model by introducing the *mixed model*, where the adversary *simultaneously* actively corrupts, passively corrupts, and fail-corrupts some of the players. We characterize an adversary by three thresholds: a (t_a, t_p, t_f)-adversary may actively corrupt up to t_a players and take full control over them, may passively corrupt up to t_p players and read their internal information, and may fail-corrupt up to t_f players and make them crash irrevocably. Note that the same player can be both passively corrupted and fail-corrupted at the same time.

We study multi-party protocols for the mixed model, with and without assuming broadcast channels, with unconditional and with perfect security, and prove tight bounds on these thresholds for secure MPC protocols to exist.

Special cases of adversaries that perform different kinds of player corruptions have previously been studied in the literature. Chaum [Cha89] considered general multi-party protocols that are secure against an adversary that can corrupt either up to d players actively or, alternatively, can corrupt up to c players passively, but in his model only one type of corruption occurs at the same time.[1] In other words, the adversary chooses between active or passive corruption and all corrupted players are corrupted in the same way. The protocols achieve correctness (with negligible error probability) with respect to any d actively corrupted players, or privacy with respect to any c passively corrupted players, where $2c + d < n$ is required. Meyer and Pradhan [MP91], and Garay and Perry [GP92] treated the case of simultaneous active and fail-corruptions for Byzantine agreement. The simultaneous presence of active and passive player corruption was considered by Dolev et al. [DDWY93] in the context of secure message transmission over general networks. In contrast to [Cha89], the models of [MP91, GP92, DDWY93] (as well as this text) consider a *mixed* adversary that *simultaneously* performs different kinds of corruption.

The classical passive and active models can be seen as special cases of this mixed model. Note, however, that there is a subtle difference between the passive model and the mixed model with $t_a = t_f = 0$. In both models, the adversary is allowed to corrupt up to $t < n/2$ of the players that participate in the simulation of the trusted party. However, inputs can be received from "external" players not involved in the simulation. In the passive model, these external players are assumed to follow the instructions of the protocol (i.e., being at most passively corrupted). In the mixed model, any number of these players may even be actively corrupted and misbehave arbitrarily.

The above results have been published in [FHM98].

1.3.2 Efficient Multi-Party Protocols

Since the introduction of secure multi-party computation, all proposed multi-party protocols that provide security against misbehaving players suffer from high communication complexity. This is in sharp contrast to their private (but non-resilient) counterparts, for which reasonably efficient solutions are known. The communication overhead of resilient

[1] For instance, the thresholds $c = 1$ and $d = \lfloor (n-1)/2 \rfloor$ given in the example in Section 3.1 of [Cha89] cannot be tolerated if simultaneous passive and active corruption occur.

multi-party protocols over private protocols results mainly from the sophisticated techniques to achieve resilience against faults. Specifically, these techniques make extensive use of a broadcast primitive, which must be realized with a protocol for Byzantine agreement. Such protocols are known to be very communication-intensive. The necessity of the broadcast channel is independent of whether or not actual faults occur: often broadcast is used to complain about an inconsistency, but in case that no inconsistency is detected, still the players must broadcast a confirmation message (the inherent information of the message is one bit).

The most efficient broadcast protocols in the literature require $\mathcal{O}(n^2)$ bits to be communicated in order to agree on a single bit among n players [BGP89, CW89]. However, these protocols require n rounds of communication. The most efficient broadcast protocols with a constant number of rounds require $\mathcal{O}(n^4)$ bits of communication [FM88]. These high complexities indicate that broadcast is an efficiency bottleneck, in both the information-theoretic setting and the cryptographic setting; reducing the number of broadcast invocations is therefore important for reducing the overall communication complexity of distributed protocols.

There is a line of research that focused on reducing the communication complexity of multi-party protocols. First, several works [BB89, BMR90, BFKR90, IK00] concentrated on reducing the round complexity of such protocols. However, the price for the low round complexity is a substantially increased bit complexity. With the current results, namely $\mathcal{O}(n^6)$ field elements per multiplication, the main efficiency bottleneck seems to be the communication complexity rather than the round complexity. First steps towards lower bit complexities were taken in [BFKR90]. The proposed protocol is very efficient, but it only tolerates t-adversaries with $t = \mathcal{O}(\log n)$. Protocols with optimal resilience were proposed in [FY92] and in [GRR98]. Their approach is to first perform a private protocol with fault-detection (for the whole protocol in [FY92], and for a part of the protocol in [GRR98]), and only in case of faults to repeat the computation with a slow but resilient protocol. Although this approach can improve the best-case complexity of the protocol (when no adversary is present), it cannot speed up the protocol in the presence of a malicious adversary: a single corrupted player can persistently enforce the robust but slow execution, annihilating (and even inverting) any efficiency gain.

We propose a framework for efficient resilient distributed protocols. This framework is general and is not restricted to MPC. The main idea of the framework is as follows: Whenever a fault occurs (and slows down

the protocol execution), a set of players which contains at least a certain number of corrupted players (but possibly also some honest ones) is identified and eliminated from the further protocol execution. Note that only players that simulate the trusted party are eliminated, and not players that send input or receive output. These eliminations ensure that faults occur only rarely, namely at most t times during the entire computation, which in turn allows to reduce the number of consistency checks performed in the protocol: Rather than after each gate, the consistency checks are performed only after a *series* of gates, a so-called *segment*. During the entire computation up to t segments can fail and require re-computation, but with an appropriate size of the segments, the total cost of re-computation will be much smaller than the savings due to the reduced number of the checks.

Rigorously applying this framework to the active protocol of [BGW88] yields an actively secure protocol which communicates $\mathcal{O}(n^3)$ field elements per multiplication, in contrast to the most efficient previous protocols that communicate $\Omega(n^6)$ field elements. A later result (not contained in this thesis), which is based on the same framework, even achieves a communication complexity of $\mathcal{O}(n^2)$ field elements per multiplication [HM01].

The above results have been published in [HMP00].

1.3.3 General Adversaries in MPC

All prior results in secure multi-party computation specify the sets of potentially corrupted players by their cardinality, i.e., by a threshold. We define more generally the security of a multi-party computation protocol with respect to an *adversary structure*, a monotone set of subsets of the players, where the adversary may corrupt the players of *one* set in this adversary structure. An adversary structure is monotone in the sense of being closed with respect to taking subsets. Non-threshold adversaries have been introduced in the context of secret-sharing (e.g., [ISN87, BL88, Gen96]).

As an example of an adversary structure, consider the set $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ of players and the adversary structure

$$\mathcal{Z} = \Big\{\emptyset, \{P_1\}, \{P_2\}, \{P_3\}, \{P_4\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}\Big\}.$$

In this example, the adversary can choose either to corrupt no player, or to corrupt a single player, or to corrupt P_1 and one additional player.

General adversaries are not necessarily restricted to a single corruption type. Most generally, an adversary structure is a set of triples of subsets of the player set, where the first subset in the triple denotes which players may be actively corrupted, the second subset denotes which players may be passively corrupted, and the third subset denotes the set of players that may be fail-corrupted. In this text, we will discuss only univariate structures (for passive, active, or fail-corruption), and bivariate adversary structures that capture simultaneous active and passive corruption. For general adversary structures that also capture fail-corruption, tight bounds for secure multi-party computation protocols to exist are not known yet and are subject of ongoing research. Interestingly, when considering general adversaries with simultaneous active corruption and fail-corruption, the necessary condition for trusted-party simulation TPS is strictly stronger than the condition for secure function evaluation SFE.

We prove tight bounds on adversary structures for secure multi-party protocols to exist, and present a construction for protocols for all admissible adversary structures, for various flavors of the secure-channels model. The bound in the passive model is that no two sets in the adversary structure cover the full player set, and in the active model that no three sets cover the player set. For example, the adversary structure from the previous example satisfies the condition that no two sets cover the player set, and hence there exists a passively secure protocol for this structure.

We introduce the technique of *player simulation* and show that with this technique, secure MPC protocols can be constructed for any admissible adversary structure. To simulate a player of a given MPC protocol means that all activities of this player are simulated in a MPC subprotocol. Of course, any number of players involved in an MPC protocol can be simulated by other MPC protocols, and even those player that simulate a player can again be simulated, and so forth. We derive and rigorously prove the adversary structure tolerated by the resulting protocol as a function of the adversary structure of the basic protocol and the adversary structure of the simulation protocols.

Then, with this technique in hand, we propose an algorithm which takes any admissible adversary structure as input, and outputs a simulation hierarchy such that the resulting protocol will tolerate the desired adversary structure. Here, the starting protocol and each simulation protocol is a well-known threshold protocol; and still, the resulting protocol tolerates the general adversary structure as required.

As corollary of our result, one can derive broadcast protocols and secret-sharing schemes for general adversary structures. The problem of broadcast for general adversary structures was later solved more efficiently [FM98]. The problem of secret-sharing for general adversary structure was solved earlier for the passive model [ISN87, BL88] and for the active model with broadcast and error probability [Gen96]. Our solution was the first for the active model without broadcast.

The results on general adversary structures have been published in [HM97, FHM99, HM00].

1.3.4 Voting

We study the problem of receipt-freeness in secret-ballot voting. The body of known receipt-free voting protocols is pretty small, and only few researchers investigate in this subject. Known receipt-free voting protocols have two major disadvantages when compared with other voting protocols: They tend to be slow, and they tend to be involved. The second property, though not necessarily a disadvantage by itself, increases the risk of overseeing attacks and insecurities. Indeed, many published receipt-free voting protocols are either completely or partially broken.

In this thesis, we restrict to receipt-free voting schemes based on homomorphic encryption. We present a modular approach for constructing such protocols, and realize two concrete protocols. Furthermore, we show for two published voting schemes that they are not receipt-free, opposed to what is claimed in the papers and was believed before.

The general approach for achieving receipt-freeness is that the voter does not generate his encrypted ballot by himself, but rather in cooperation with the authorities. This will ensure that even the voter cannot decrypt his own ballot, and hence cannot prove the vote to a vote-buyer.

In the first proposed protocol, a special representation of the ballot is used, which allows *ballot shuffling*. This means that for any given encrypted ballot for some (unknown) candidates and a permutation on the set of all valid candidates, one can compute a new ballot for the permuted candidates and with different randomness, without need for decrypting the original ballot. For example, if an encrypted ballot for candidates $\{1, 5\}$ is shuffled with the permutation $(1, 3, 4)(2, 5, 6)$, then the new encrypted ballot will be for the candidates $\{3, 6\}$. With consecutively using this primitive, the voter can guide the authorities to construct an en-

crypted ballot of the vote he wants to cast, and still the authorities do not learn the vote (and the voter cannot decrypt the encrypted ballot).

The second scheme is based on the idea of randomization. The voter sends an encryption of his vote to a designated authority, the *randomizer*, who changes the randomness used for encryption, such that the voter cannot decrypt his own encrypted ballot. This new encrypted vote is then tallied. The main problem to be solved is the verification of the validity of the new encryption. We present a protocol which allows the voter and the randomizer to jointly generate an non-interactive validity proof of the new encryption, without the randomizer learning the vote, and without the voter learning the randomness of the new encryption.

Both voting schemes are more efficient than any receipt-free voting scheme in the literature, where the second approach is significantly more efficient than the first approach. However, in the first approach, in certain (well-defined) cases, even authorities cannot coerce the voter, whereas in the second approach, such coercion is possible.

Finally, we show that the voting protocols of [BT94] and [LK00] are not receipt-free, opposed to what was believed before.

Both proposed voting schemes are yet unpublished, but the scheme based on ballot shuffling is inspired by the protocol of [HS00]. The security analysis of the protocol of [BT94] is also contained in that paper; the analysis of the protocol of [LK00] is yet unpublished.

Chapter 2

Formal Definition of MPC

In this section, we formally state our model for secure multi-party computation, and define security for such protocols. These definitions are stronger than required, and often are restricted to the settings that we consider. In particular, we restrict to the so called secure-channels model [BGW88, CCD88], where pairwise secure and authentic synchronous channels between every pair of players are assumed. The adversary is allowed to have unlimited computing power and may be adaptive. We define security of MPC protocols with using the trusted party approach, i.e., we compare the protocol with a specification involving a trusted party, and require that the adversary cannot achieve anything in the protocol that he could not achieve in the specification as well. This security definition is highly inspired by Canetti's definitions [Can00]. For the sake of simplicity, the definitions (as well as the security proofs in the following chapters) only capture static adversaries. However, both the definitions and the proofs can be naturally extended to also cover the case of adaptive adversaries. The definitions proposed in this section were originally developed in [HM00]. Alternative definitions of security can be found in [Bea91b, MR91, MR98].

2.1 Players, Processors, and Communication

Players are assumed to perform two entirely different tasks: On one hand, they provide input and receive output, depending on some real-world concerns, and on the other hand, they are supposed to perform the

operations of the actual protocol. It is necessary to clearly distinguish between these two tasks. Therefore, in the sequel we refer to a *player* only as the entity that provides input and receives output, and to the associated *processor* as the entity that performs the operations of the protocol. This distinction is important for taking into account the fact that in a general multi-party specification with several input stages, the players' inputs can depend on information obtained during the execution of, but outside of the protocol (e.g. insider information in a stock-market protocol). Intuitively, one might think of processors as computers, and of players as the humans using the computers.

Processors are denoted by P_i. A processor can send values to and receive values from other processors, and can perform arbitrary computations on his known values. Formally (and without loss of generality) we assume that each processor can compute arithmetic operations in a fixed finite field $(\mathbb{F}, +, *)$, can select elements from this field at random, and can communicate with other processors over perfectly authenticated and confidential synchronous channels. We assume that the field $\mathbb{F}$ is the same for all processors.

In addition to processors associated with players, we also introduce the abstract concept of a *virtual processor*, which offers the same functionality as a processor but only appears in the construction of a protocol. In particular, the trusted party of a specification (or other simulated processors) are virtual processors. One can think of virtual processors as of computer servers.

Formally, a processor can be modeled as a probabilistic Turing Machine, with a (read-only) input tape, a (write-only) output tape, and a (read-write) working tape. The player associated with a processor can write his input tape and can read his output tape. The input and output tapes of virtual processors are not used. Every pair of processors can communicate via a pair of tapes, where one tape is read-only for the first and write-only for the second processor, and the other tape is write-only for the first and read-only for the second processor. If broadcast channels are assumed, then furthermore for every processor there exists a tape which is write-only for this particular processor and read-only for all other processors. All tapes (in particular the communication tapes) are private and authentic, i.e., only the involved processors can read from (or obtain any information about) or write to a tape. Furthermore, communication tapes are synchronous: We assume the existence of a global clock, and data written to a communication tape can be read at the next clock cycle.

2.2 Variables and Views

We assume a global *variable space* $\mathcal{X}$, consisting of all variables used during the protocol execution. A *variable* $x \in \mathcal{X}$ can take on a value from the given finite field $(\mathbb{F}, +, *)$. Every quantity ever generated during a protocol execution, including inputs, local data (e.g. shares) and outputs, is assigned to a variable. For a particular protocol execution each variable takes on only one particular value, i.e., variables are not to be understood in the sense of an imperative programming language but rather as labels for values or, more precisely, as a fixed binding between a name and a value.

In order to guarantee that variables take on only one particular value, every variable can be seen only by one designated processor. This is modeled by associating a visibility space with every processor P, capturing the set of variables known to P. The set of visibility spaces of all processors define a partition on the variable space $\mathcal{X}$. During the protocol execution, values are bound to variables. The *view* $\nu(P)$ of a processor is the set of bindings of all variables in his visibility space. The view $\nu(B)$ of a set B of processors is the union of the views of the processors in B. Note that a processor may have full or partial knowledge about a variable even if the variable is not in his view. Somewhat sloppily, we also denote the visibility space of P by $\nu(P)$.

2.3 Protocols, Specifications, and Protocol Generators

A *protocol* π among a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of processors that involves variables from a variable space $\mathcal{X}$ is a sequence $d_1, \ldots, d_l$ of statements. There are four types of *statements*: An *input statement* $\mathit{input}(P_i, x)$ instructs the processor $P_i \in \mathcal{P}$ to read a value from his input tape (i.e., from his associated player) and to assign the value to the variable $x \in \mathcal{X}$. A *transmit statement* $\mathit{transmit}(P_1, P_2, x_1, x_2)$ instructs the processor $P_1 \in \mathcal{P}$ to send the value of the variable $x_1 \in \mathcal{X}$ to the processor $P_2 \in \mathcal{P}$, who then assigns the received value to variable $x_2 \in \mathcal{X}$. An *output statement* $\mathit{output}(P, x)$ instructs the processor $P \in \mathcal{P}$ to output the value of the variable x to his associated player. Finally, *computation statements* are of one of three forms: A $\mathit{comp}(P, +, x, x_1, x_2)$-statement (a $\mathit{comp}(P, *, x, x_1, x_2)$-statement) instructs the processor P to add (multiply) the values of

the variables x_1 and x_2 and to assign the result to the variable x. A $comp(P, \mathsf{ran}, x)$-statement instructs the processor P to select an element from the field at random and to assign the value to the variable x.

Assigning a value to a variable (in an input or computation statement) means to define its (global) value and to include it in the processor's view, and is only admissible if no value has previously been assigned. A processor can only use (in a computation, transmit, or output statement) the values of variables that are globally defined and are included in the processor's view.

A *multi-party computation specification* (or simply called specification) formally describes the cooperation to be performed and the processors that give input to, or receive output from the computation. Intuitively, a specification specifies the cooperation in an ideal environment involving a trusted party. Formally, a specification is a pair (π_0, τ) consisting of a protocol π_0 among a set $\mathcal{P}_0$ of processors, and the name of a virtual processor $\tau \in \mathcal{P}_0$. The protocol π_0 of the specification is also called the *ideal protocol*.

A *multi-party protocol generator* G for the set $\mathcal{P}_G$ of processors is a function that takes as input a multi-party computation specification (π_0, τ) involving processors from a set $\mathcal{P}_0$ and returns a protocol π for the processors $\big(\mathcal{P}_0 \backslash \{\tau\}\big) \cup \mathcal{P}_G$. A *statement index function* for a specification (π_0, τ) and protocol π is a strictly monotone function f,

$$f : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\} \,,$$

where $f(1) = 1$ and $f(|\pi_0| + 1) = |\pi| + 1$.

The intuition is that a protocol generator G simulates the virtual trusted processor τ by a multi-party computation protocol among the processors in $\mathcal{P}_G$. Each statement of the ideal protocol π_0 is expanded into a sequence of statements, and all these sequences are concatenated to the resulting protocol π. In order to keep track of which subsequence of π resulted from a given statement d_i of the ideal protocol π_0, the statement index function maps the index i of each statement d_i in π_0 to the index $f(i)$ of the first statement in the corresponding expansion in π, i.e., the i-th statement of π_0 "is computed" by the sequence $f(i)$ to $f(i+1)-1$ of statements of π (since $i = |\pi_0|$ is possible, the domain of f includes $|\pi_0| + 1$).

Furthermore, we say that a protocol generator is *natural* if the simulation of a statement only involves variables that also occur in the statement itself, plus optionally some auxiliary variables which are uniquely

associated with one of the involved variables. Formally, a protocol generator with statement index function f is natural if there exists a surjective function from the variable space of the real protocol π onto the variable space of the ideal protocol π_0, where for every $i = 1, \ldots, |\pi_0| + 1$ and $j = f(i), \ldots, f(i+1) - 1$, every variable involved in the j-th statement of the real protocol π maps to a variable involved in the i-th statement of the ideal protocol π_0.

Note that throughout this text, we will only consider protocol generators that are natural. Some of the techniques we will use (so-called processor simulation, Section 4.1) are only secure for natural protocol generators. To the best of our knowledge, all protocol generators in the literature satisfy this property.

2.4 Classes and Structures

We consider three corruption types: active corruption, passive corruption, and fail-corruption. Adversaries are classified according to which processors may be corrupted by which type. We define an adversary class C to be a triple of subsets of the processor set $\mathcal{P}$, i.e., $C \in 2^{\mathcal{P}} \times 2^{\mathcal{P}} \times 2^{\mathcal{P}}$. An adversary of class $C = (D, E, F)$ actively corrupts the processors in D (disruption), passively corrupts the processors in E (eavesdropping), and fail-corrupts the processors in F (see next Section for a more formal definition of corruption). We require that D is disjoint from both E and F, but E and F may overlap.

We define the notion of subclasses: A class $C' = (D', E', F')$ is a *subclass* of a class $C = (D, E, F)$, denoted $C' \subseteq C$, if and only if every adversary of class C' can also be considered as an adversary of class C:

$$(D', E', F') \subseteq (D, E, F) \Leftrightarrow D' \subseteq D, E' \subseteq (E \cup D), \text{ and } F' \subseteq (F \cup D).$$

To *restrict* a class (D, E, F) to a processor set $\mathcal{P}'$ means to only consider the processors in $\mathcal{P}'$:

$$(D, E, F)\big|_{\mathcal{P}'} = (D \cap \mathcal{P}', E \cap \mathcal{P}', F \cap \mathcal{P}').$$

Furthermore, we consider collections of such adversary classes: An *adversary structure* $\mathcal{Z}$ is a set of adversary classes closed under taking subclasses, i.e., $C \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}} \times 2^{\mathcal{P}}$, where $C \in \mathcal{Z}, C' \subseteq C \Rightarrow C' \in \mathcal{Z}$. For a

structure $\mathcal{Z}$, $\overline{\mathcal{Z}}$ denotes the *basis* of the structure, i.e., the set of all maximal classes in $\mathcal{Z}$:

$$\overline{\mathcal{Z}} = \{C \in \mathcal{Z} : \nexists C' \in \mathcal{Z} : C \subset C'\}$$

For a set B of classes, $\langle B \rangle$ denotes the *closure* of the set with respect to taking sub-classes, i.e.,

$$\langle B \rangle = \{C' \subseteq C : C \in B\}.$$

Particularly, the closure of a basis is the original structure, i.e., for all adversary structures $\mathcal{Z}$, it holds that $\mathcal{Z} = \langle \overline{\mathcal{Z}} \rangle$.

To restrict an adversary structure $\mathcal{Z}$ means to restrict every class in $\mathcal{Z}$:

$$\mathcal{Z}|_{\mathcal{P}'} = \left\{C|_{\mathcal{P}'} : C \in \mathcal{Z}\right\}.$$

In some cases, we will restrict ourself to certain corruptions types. When only active and passive corruption is considered, then we will define a class C to be a pair (D, E), and accordingly, a bivariate adversary structure $\mathcal{Z}$ to be a set of such pairs, i.e., $\mathcal{Z} \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}}$. In addition, when only a single corruption type is considered (active or passive), then the adversary class is simply the set of the corruptible processors, and the univariate adversary structure $\mathcal{Z}$ is a set of such sets, i.e., $\mathcal{Z} \subseteq 2^{\mathcal{P}}$. All above definitions also apply to these restricted classes and structures.

2.5 Adversaries and Definition of Security

We consider three types of corruptions:

- Passive corruption: The adversary can read the internal information of the corrupted processors.
- Active corruption: The adversary can take complete control over these processors, and can make them (mis-)behave in any desired manner.
- Fail-corruption: The adversary can make the processor fail, i.e., can ultimately prevent the processor from sending any more messages to any other processor.

An adversary is characterized by the subset of processors that can be corrupted in each corruption type: A (static) *adversary of class* (D, E, F) (for short: a (D, E, F)-adversary) is a (probabilistic) program (or strategy), which passively corrupts the processors in E (eavesdrop), actively corrupts the processors in D (disrupt), and fail-corrupts the processors in F. The variables held by the passively and actively corrupted players (i.e., $\nu(E \cup D)$) are included in the adversary's view. Furthermore, the adversary can take full control over the communication tapes of the actively corrupted processors, i.e., can send messages in their name. Finally, the adversary can make any fail-corrupted processor crash at any time. Once a processor crashed, it cannot be re-activated. The adversary can perform an arbitrary computation on the values in his view and extend his view by the computed values.[2] All decisions taken by the adversary may depend on his actual view.

We do not give a more precise definition of the adversary's view but it is understood that it consists of random variables with a well-defined range. For instance, if the adversary is modeled as a Turing machine, the view consists of the content of all tapes. The complexity of an adversary is not assumed to be polynomial.

For an adversary A, a protocol A-*securely computes* a specification if, whatever A does in the protocol, the same effect could be achieved by an adversary (with a modified strategy, but with similar costs) in the ideal protocol of the specification. More precisely, we require that the joint distribution of the variables that the adversary sees in the real protocol and those that any uncorrupted processors sees in the ideal protocol are equally distributed in the real protocol and the ideal protocol. This means that the adversary sees in the real protocol at most as much as what he would see in the ideal protocol, and the players see in the real protocol at least as much as they would see in the ideal protocol. This must hold right before every statement of the ideal protocol and the corresponding statement in the real protocol.

Formally, for an adversary A of class (D, E, F), and a specification (π_0, τ) for the set $\mathcal{P}_0$ of processors, the protocol π A-securely computes the specification (π_0, τ) if there exists a statement index function $f_\pi : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\}$ and an adversary A_0 for the ideal

[2] The values in the adversary's view need neither be elements of the finite field nor be assigned to variables of the global variable space. However, such a restriction could be made without loss of generality.

protocol π_0 with[3] $(D_0, E_0, F_0) = (D, E, F)\big|_{\mathcal{P}_0 \setminus \{\tau\}}$ such that for all inputs and for every $i = 1, \ldots, |\pi_0| + 1$ the joint distribution of the variables known to real adversary A and the uncorrupted processors in the specification is the equal before the i-th statement of the ideal protocol π_0 (with the adversary A_0 present) and before the $f(i)$-th statement of the real protocol π (with the adversary A present). Moreover, the complexity of A_0 must be polynomial in the complexity of A. This corresponds to the definition of on-line security of [Can00]. The adversary A_0 can be seen as a kind of simulator and is called the *ideal adversary* of A.

For the special case of secure function evaluation, the only effect that an adversary A can achieve in a protocol that A-securely computes this specification corresponds to a modification of the inputs and outputs of the corrupted processors in the ideal protocol (which of course cannot be prevented).

For an adversary structure $\mathcal{Z}$ and a specification (π_0, τ), a protocol π *$\mathcal{Z}$-securely computes* the specification (π_0, τ) if for every adversary A of class $C \in \mathcal{Z}$, the protocol π A-securely computes the specification (π_0, τ). Whenever the specification is clear from the context, we also say that a protocol *tolerates* an adversary A (a structure $\mathcal{Z}$) instead of saying that the protocol A-securely ($\mathcal{Z}$-securely) computes the specification.

A protocol generator G for the set $\mathcal{P}$ of processors is *A-secure* for a given adversary A (i.e., *G tolerates A*), if for every specification, the protocol that results by applying the generator to this specification A-securely computes the specification. For a structure $\mathcal{Z} \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}} \times 2^{\mathcal{P}}$, a protocol generator G for the set $\mathcal{P}$ of processors is *$\mathcal{Z}$-secure* (or *tolerates $\mathcal{Z}$*) if for every adversary A of class C with $C\big|_{\mathcal{P}} \in \mathcal{Z}$, the protocol generator is A-secure.

2.6 Restricted Models of Special Interest

In this section, we mention several restricted models that were of particular interest in the past, and show how they can be viewed as special case of our definitions.

[3] It is necessary to explicitly exclude τ because it is possible that the name τ is used in π, and even if A may corrupt the processor called τ (which thus is a simulating processor), it cannot be tolerated that A_0 corrupts τ (which is the trusted party of the specification). At this point, this technicality appears to be pedantic, but in later recursive constructions it will be necessary.

2.6.1 Passive and Active Threshold Model

In the past, multi-party computation protocols were considered uniquely for two models: The passive model, and the active model, both with a threshold adversary. In the passive model, the adversary is allowed to passively corrupt up to t out of the n processors. No processors may be actively corrupted or fail-corrupted. In this model, it was proven that perfectly t-secure protocols exist for $t < n/2$ [BGW88, CCD88]. For a set $\mathcal{P}$ of processors with $|\mathcal{P}| = n$, the adversary structure for the passive model is as follows:

$$\mathcal{Z} = \{(\emptyset, E, \emptyset) \colon E \subseteq \mathcal{P}, |E| \leq t\}\,.$$

Accordingly, in the active model, the adversary may actively corrupt any t processors. In this model, perfectly t-secure multi-party computation is possible for $t < n/3$ [BGW88], and, when additionally assuming the existence of broadcast channels, unconditionally t-secure protocols exist for $t < n/2$. The adversary structure for the active threshold model is as follows:

$$\mathcal{Z} = \langle\{(D, \emptyset, \emptyset) \colon D \subseteq \mathcal{P}, |D| \leq t\}\rangle\,.$$

2.6.2 General Adversaries

Historically, the first step of generalizing adversaries for multi-party computation was to introduce non-threshold security for the passive and for the active model [HM97]. A general adversary for the passive model is characterized by a passive (univariate) adversary structure $\mathcal{Z}_p \subseteq 2^{\mathcal{P}}$. The corresponding general adversary structure is as follows:

$$\mathcal{Z} = \{(\emptyset, E, \emptyset) \colon E \in \mathcal{Z}_p\}\,.$$

The characterization of a general adversary for the active model is analogous: For an active (univariate) adversary structure $\mathcal{Z}_a$, the corresponding general adversary structure is:

$$\mathcal{Z} = \langle\{(D, \emptyset, \emptyset) \colon D \in \mathcal{Z}_a\}\rangle\,.$$

2.6.3 Mixed Adversaries

Later, the notion of mixed adversaries was introduced [FHM98]. A mixed adversary is characterized by three thresholds, t_a, t_p, and t_f, specifying the upper bound on the number of active corruptions, of passive corruptions, and of fail-corruptions, respectively. A (t_a, t_p, t_f)-adversary is characterized by the following adversary structure:

$$\mathcal{Z} = \left\langle \left\{ (D, E, F) \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}} \times 2^{\mathcal{P}} : |D| \leq t_a, |E| \leq t_p, |F| \leq t_f \right\} \right\rangle .$$

2.6.4 Mixed General Adversaries

Finally, mixed general adversaries were considered in [FHM99], where fail-corruption was not captured. There, a mixed adversary structure $\mathcal{Z}_m$ is a set of pairs of subsets of the processor set, i.e., $\mathcal{Z}_m \subseteq 2^{\mathcal{P}} \times 2^{\mathcal{P}}$. Such a mixed adversary structure can be converted to a general adversary structure as follows:

$$\mathcal{Z} = \{ (D, E, \emptyset) : (D, E) \in \mathcal{Z}_m \} .$$

Chapter 3

MPC Protocols with Threshold Security

In this section, we focus on multi-party computation protocols with threshold security, i.e., the set of corruptible processors is characterized by its cardinality. We first consider several models, characterized by corruption type(s), by whether or not broadcast channels are available, and by whether the security must be perfect or unconditional. Finally, we will focus on the efficiency of threshold multi-party protocols.

As a warm-up, we first consider the passive model. In this model, the adversary is characterized by the upper bound t_p of processors that he may passively corrupt. A *passive t_p-adversary* is an adversary that may passively corrupt any up to t_p of the processors, and a protocol is *t_p-private* if it is secure with respect to every passive t_p-adversary. The main result in this model is due to Ben-Or, Goldwasser, and Wigderson [BGW88], and independently by Chaum, Crépeau, and Damgård [CCD88]: In the passive model with n processors, t_p-private protocols can be found for any specification exactly if $t_p < n/2$. We will present the protocol of [BGW88], with an algebraic simplification due to Gennaro, M. Rabin, and T. Rabin [GRR98], in Section 3.1.

Afterwards, in Section 3.2, we consider the active model. Here, the adversary is characterized by the upper bound t_a of processors that he may actively corrupt. An *active t_a-adversary* may actively corrupt any up

to t_a processors, and a protocol is t_a-*robust* if it tolerates every active t_a-adversary. The main result in the active model with n processors states that perfectly secure protocols exist exactly if $t_a < n/3$ [BGW88].[4]

If broadcast channels are available and a negligible probability of failure is accepted, then this bound can even be improved to $t_a < n/2$ [RB89, Bea91b, CDD+99]. Models with broadcast channels are out of the scope of this thesis, and we only present the main ideas of the protocol for this model (Section 3.3).

For completeness, we also consider the fail-stop model. Here, the adversary is characterized by a threshold t_f, and he may fail-corrupt any t_f processors. Indeed, in this model, the adversary can be allowed to fail-corrupt any number $t_f < n$ of processors. Details are given in Section 3.4.

Then, we bear down this classical distinction between the passive and the active model. In the so called *mixed model*, the adversary is characterized by three thresholds t_a, t_p, and t_f, where t_a denotes the upper bound on the number of processors that the adversary may actively corrupt, t_p denotes the upper bound on the number of processors that the adversary may passively corrupt, and t_f denotes the upper bound on the number of processors that can be fail-corrupted. Formally, a (t_a, t_p, t_f)-adversary may *simultaneously* actively corrupt any t_a, passively corrupt any (other) t_p processors, and fail-corrupt another t_f processors, and a protocol is (t_a, t_p, t_f)-secure if it tolerates every (t_a, t_p, t_f)-adversary.

Note that the mixed model with $t_a = 0$ and $t_f = 0$ is more powerful than the passive model. In a general setting, there are processors that participate in the simulation of the virtual trusted party, and there are processors that only provide input or receive output. In the passive model, also processors that only provide input must be only passively corrupted, whereas in the mixed model, these processors may be corrupted of any type.

We distinguish four cases of the mixed model: perfect and unconditional security with and without assuming broadcast channels. Perfect security can be achieved if and only if $3t_a + 2t_p + t_f < n$, independently of whether or not broadcast channels are available. This is stated and proven in Section 3.5.

With unconditional security, when a negligible probability of failure is tolerated, then assuming broadcast channels helps improving the

[4] The same result was also achieved by [CCD88], but their protocol has some (negligible) probability of error.

conditions for multi-party computation. When broadcast channels are given, then the necessary and sufficient condition for secure MPC is $2t_a + 2t_p + t_f < n$. A protocol achieving this bound will be sketched in Section 3.6, and the necessity of the bound will be proven.

If broadcast channels are not available, then invocations of the broadcast primitive must be simulated, and the necessary condition for multi-party computation must be strengthened. Additionally to the above condition $2t_a + 2t_p + t_f < n$, we must require that $3t_a + t_f < n$. This model will be considered in Section 3.7.

All results are summarized in Table 3.1. It turns out that every classical model can be considered as a special case of the mixed model, and that the conditions for secure multi-party computations generalize naturally and continuously. Surprisingly, the upper bound t_f on the number of fail-corruptions appears with weight 1 in all conditions, which means that fail-corruptions can be tolerated "for free". Any strategy of the adversary about which processors to fail-corrupt in which circumstances is void; no strategy is more powerful than make fail the first t_f processors in the very beginning of the protocol execution (and tell all honest processors about). This allows to illustrate the conditions graphically in only two dimensions for a constant t_f. Figure 3.1 shows the necessary and sufficient conditions for unconditionally secure multi-party computation in various models. The brightly shaded area is achievable with perfect security, and the semi-dark area is achievable when accepting an exponentially small failure probability. Finally, the dark shaded area is achievable if additionally broadcast channels are available.

	$\varepsilon = 0$		$\varepsilon > 0$	
Model	no BC	BC	no BC	BC
passive	$2t_p < n$			
active	$3t_a < n$			$2t_a < n$
fail-stop	$t_f < n$			
mixed	$3t_a+2t_p+t_f < n$		$2t_a+2t_p+t_f < n$ $3t_a+t_f < n$	$2t_a+2t_p+t_f < n$

Table 3.1: Summary of the sufficient and necessary conditions for unconditional MPC.

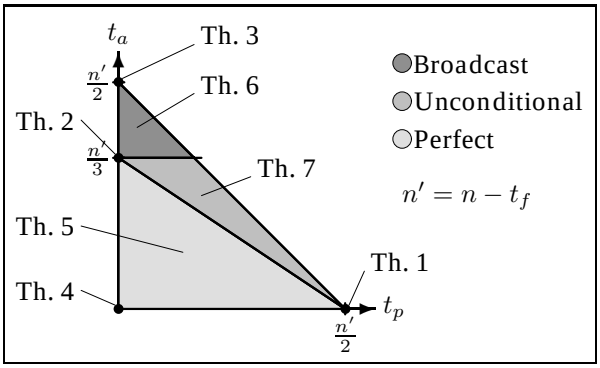


Figure 3.1: Graphical representation of the results.

Finally, we consider the efficiency of MPC protocols. We first present a framework for very efficient resilient protocols (Section 3.8). This framework is not restricted to general multi-party computations, but can also be applied to other distributed protocols. In this framework, resilience is added on top of an (efficient) private protocol, and the overhead for resilience can be reduced substantially compared to all previous approaches.

We then apply these techniques to the multi-party protocol for the active model, and obtain a protocol with a significantly lower communication complexity than any previous multi-party computation protocol (Section 3.9). Asymptotically, the communication complexity of the resulting protocol is $\mathcal{O}(n^3)$ field elements per multiplication, compared to $\mathcal{O}(n^6)$ field elements of previous protocols with unconditional security. The new protocol improves even on the most efficient cryptographically secure protocol [GRR98], which communicates $\mathcal{O}(n^4)$ field elements per multiplication (but tolerates up to $t < n/2$ corruptions). A later protocol with cryptographic security [CDN01] achieves essentially the same communication complexity as our protocol (but tolerates more corruptions). Subsequently, a new protocol based on the framework of Section 3.8 was presented, achieving unconditional security (with negligible error probability) with communicating $\mathcal{O}(n^2)$ field elements per multiplication. The round complexities of all considered protocols are essentially equal. All stated complexities include the costs of simulating the broadcast channels by a sub-protocol for Byzantine agreement.

Applying the framework to the mixed model with perfect security

is straight-forward and can easily be done. However, applying it to the protocol for the active model with broadcast seems more difficult and is an open problem. So far, the protocol of Section 3.3 [CDD+99] is the most efficient known protocol for this model.

3.1 Passive Model

In this section, we give a brief description of the protocol of Ben-Or, Goldwasser, and Wigderson [BGW88], with an algebraic simplification in the multiplication protocol due to Gennaro, M. Rabin, and T. Rabin [GRR98]. The achievement of this protocol is formally stated in the following theorem. The necessity of the condition $t < n/2$ is a corollary of Theorem 5 and will be proven in Section 3.5.

Theorem 1 *In the passive model, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification (perfectly) securely if and only if the adversary corrupts at most $t < n/2$ of the processors. The computation is polynomial in n and linear in the size of the circuit.*

According to the definition of a specification, we have to give realizations (simulations) of *transmit*- and *comp*-statements. The following table shows with which sub-protocol statements involving the virtual trusted party τ will be simulated. Statements that do not involve τ (not shown in the table) are not replaced.

Statement	Simulation
transmit(P, τ, x)	Secret-sharing, where P acts as dealer
transmit(τ, P, x)	Secret-reconstruction towards P
comp$(\tau, +, x, x_1, x_2)$	Addition sub-protocol
comp$(\tau, *, x, x_1, x_2)$	Multiplication sub-protocol
comp(τ, ran, x)	Randomness sub-protocol

In the sequel, we will describe realizations of these five sub-protocols. But first we have to define what it means that a variable is shared among the processors. We assign a unique non-zero element $\alpha_i \in (\mathbb{F}\backslash\{0\})$ to each processor P_i. Then, a variable x is *t-shared* among the processors, if every processor P_i holds a share x_i, and there exists a polynomial $f(x)$ of degree at most t such that $f(0) = x$ and $f(\alpha_i) = x_i$ for $i = 1, \ldots, n$. Furthermore, we require that $f(x)$ is chosen at random and independently from any other values.

3.1.1 Secret Sharing

Secret sharing is based on Shamir's secret sharing [Sha79]. The dealer P who wants to share a secret s selects a random polynomial

$$f(x) = s + r_1 x + \ldots + r_t x^t, \;\; r_i \in_R \mathbb{F},$$

of degree at most t, and sends to each processor P_i his respective share $s_i = f(\alpha_i)$.

3.1.2 Secret Reconstruction

Every processor P_i sends his share s_i of the value s to the processor P who is to reconstruct the secret. According to Lagrange's formula, the following expression gives the polynomial $f(x)$ with lowest degree which interpolates the given shares:

$$f(x) = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j} s_i.$$

This immediately gives the formula to compute the secret s:

$$s = f(0) = \sum_{i=1}^{n} w_i s_i, \text{ where } w_i = \prod_{j=1, j\neq i}^{n} \frac{\alpha_j}{\alpha_j - \alpha_i}.$$

3.1.3 Addition and Linear Functions

Given two shared secrets a and b, which are shared with the polynomials $f(x)$ and $g(x)$, respectively, i.e., $f(0) = a$ and $h(0) = b$. Each processor P_i holds a share a_i of a and a share b_i of b. We define the polynomial $h(x) = f(x) + g(x)$, and observe that $h(0) = a + b$. Hence the shares $c_i = h(\alpha_i) = a_i + b_i$ of each processor P_i define a sharing of $a + b$.

Along this line, any linear function on shared values can be computed by computing the function on each processor's shares.

3.1.4 Multiplication

First, we observe that secret reconstruction is a linear computation of the shares. This immediately leads to the multiplication protocol of [GRR98]: Assume that the values a and b are shared among the processors with the polynomials $f(x)$ and $g(x)$, respectively, each of degree t. That is, $f(0) = a$, $g(0) = b$, and processor P_i holds the shares $a_i = f(\alpha_i)$ and $b_i = g(\alpha_i)$. Then, the product shares $d_i = a_i b_i$ define a polynomial $h(x) = f(x)g(x)$ of degree $2t$, where $h(0) = ab$. Note that $2t < n$ implies that the n product shares of $h(x)$ are sufficient for interpolating $h(0)$. Hence, a t-sharing of $c = ab$ can be achieved as follows: Every processor P_i secret shares his product share d_i with a polynomial of degree t. Then, a degree-t sharing of ab can be computed as a linear combination of these sharings, according to Lagrange interpolation. More precisely, the shares $d_1, \ldots, d_n$ are shared with the degree-t polynomials $g_1(x), \ldots, g_n(x)$, respectively, where every processor P_j holds the share-shares $d_{1j}, \ldots, d_{nj}$. Then the polynomial

$$g(x) = \sum_{i=1}^{n} w_i g_i(x)$$

shares c (i.e., $g(0) = c$), and every processor P_j can compute his degree-t share of c as

$$c_j = g(\alpha_j) = \sum_{i=1}^{n} w_i g_i(\alpha_j) = \sum_{i=1}^{n} w_i d_{ij}.$$

3.1.5 Random Field Elements

In order to select an element $x \in \mathbb{F}$ at random, every processor P_i locally selects one element $x_i \in \mathbb{F}$ at random, shares it among the processors (by using the above sub-protocol for secret-sharing), and x is computed as the sum of those shared local elements (by using the above sub-protocol for addition). A slight speed-up can be achieved by having only (the first) $t + 1$ processors select and share a random element.

3.2 Active Model

In the active model, where corrupted processors may deviate from the given instructions, the adversary may be allowed to corrupt at most

$t < n/3$ of the processors [BGW88]. This is formally stated by the following theorem, and will be proven in the remaining of this section. The construction is based on the protocol of [FHM98]. Again, the necessity of the condition $t < n/3$ is a corollary of Theorem 5 and will be proven in Section 3.5.

Theorem 2 *In the active model, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification (perfectly) securely if the adversary corrupts at most $t < n/3$ of the processors. The computation is polynomial in n and linear in the size of the circuit.*

The protocol for the active model is essentially the same as the one for the passive model, with some additional ingredients that allow to verify that the processors honestly perform the required computations. This is achieved as follows: Every processor is committed to every value of the passive protocol. When a processor is to send a value to another processor, then he opens the commitment towards the recipient. When a processor is to perform some computation on these values, then the processor must prove (in zero-knowledge) that indeed the commitment of the outcome is compatible with the commitments of the inputs, according to the computed function. This technique allows to detect whenever a processor deviates from the protocol, and hence the only relevant cheating is to refuse cooperation. However, such a cheating is easy to tolerate.

Along the lines of the passive model, we have to give realizations of sub-protocols for secret-sharing, secret-reconstruction, addition, multiplication, and selecting random field elements. The sub-protocols presented below make use of a broadcast primitive. Such a primitive is not assumed to exists. But in a model where at most $t < n/3$ of the processors are actively corrupted, broadcast can be simulated with a broadcast sub-protocol [BGP89, CW89].

The sharing is based on Shamir's secret-sharing scheme [Sha79], extended to a two-dimensional sharing [GHY87, BGW88, CCD88, RB89, FHM98, CDM00]. To each processor P_i, a unique non-zero element $\alpha_i \in (\mathbb{F} \setminus \{0\})$ is assigned. In contrast to [BGW88, GRR98], no additional mathematical structure on the values of α_i is required. Each value is shared among the processors with a polynomial of degree t, and each share is again shared among the processors with a polynomial of degree t. The second level of the sharing is to be understood as a commitment of the respective processor to his share. The processor can open the commitment (and thereby verifiably reveal his share) by broadcasting the

corresponding polynomial. If more than t processors complain that this polynomial doesn't match their share-share, then obviously the opening processor is corrupted and the opening fails. If at most t processors complain, then at least $t+1$ honest processors did not complain, and their share-shares uniquely define the polynomial, hence it must be the correct one.

Formally, a value s is *t-shared* among the processors if there exist degree-t polynomials $f(x)$ and $f_1(x), \ldots, f_n(x)$ with $s = f(0)$ and $f_i(0) = f(\alpha_i)$. The information held by processor P_i is the share $s_i = f(\alpha_i)$, the polynomial $f_i(x)$, and the share-shares $s_{ji} = f_j(\alpha_i)$ (for $j = 1, \ldots, n$). The polynomials in the sharing must be randomly chosen such that any set of t processors do not obtain any information about the secret.

3.2.1 Verifiable Secret Sharing

The following VSS protocol is a slight variation of the protocol proposed in [BGW88]. The protocols allows the dealer P to share a secret s verifiably among the set $\mathcal{P}$ of processors. More formally, in the beginning, the dealer holds a secret s, and after the protocol execution, s is t-shared among the processors in $\mathcal{P}$. The consistency of the resulting sharing is guaranteed as long as at most the dealer and any up to t processors in $\mathcal{P}$ are actively corrupted.

1. DISTRIBUTION. The dealer P selects at random a polynomial $p(x,y) = \sum_{i,j=0}^{t} r_{ij} x^i y^j$ of degree t in both variables, where $p(0,0) = s$, and sends the polynomials (respectively their coefficients) $f_i(x) = p(x, \alpha_i)$ and $\widetilde{f}_i(y) = p(\alpha_i, y)$ to processor P_i (for $i = 1, \ldots, n$).[5] This implicitly defines the polynomial $f(x) = p(0,x)$.
2. CONSISTENCY CHECKS. Each pair of processors P_i, P_j (for $1 \leq i, j \leq n$) checks whether $f_i(\alpha_j) \stackrel{?}{=} \widetilde{f}_j(\alpha_i)$. For this, P_i sends $f_i(\alpha_j)$ to P_j, and P_j checks whether the received value is equal to $\widetilde{f}_j(\alpha_i)$.
3. COMPLAINT STAGE. Every processor P_i broadcasts a message (containing one bit) indicating whether all consistency checks were successful or at least one test failed. In case of a complaint, the processor afterwards broadcasts a bit-vector, where the j-th bit indicates

[5] An efficiency gain of a factor 2 can be achieved by setting $r_{ij} = r_{ji}$, and hence $f_i(x) = \widetilde{f}_i(x)$. One can prove that privacy is not violated by this technique. See [CDM00] for more details.

whether or not the processor has observed an inconsistency with processor P_j. The dealer answers each complaint (i, j) by broadcasting the correct value $p(\alpha_i, \alpha_j)$.

4. FIRST ACCUSATION STAGE. Every processor P_j who observes more than t inconsistencies, or discovers that the dealer's answers to complaints contradict his own values as received from the dealer earlier, broadcasts an accusation against the dealer. In such a case the dealer broadcasts both polynomials $f_j(x)$ and $\widetilde{f}_j(y)$.
5. SECOND ACCUSATION STAGE. Every processor who discovers that a polynomial broadcast by the dealer in the previous stage is inconsistent with his own values broadcasts an accusation. These accusations need not be replied by the dealer.

If in total more than t processors have accused (in both accusation stages), or if the dealer did not answer all the complaints and accusations, clearly the dealer is corrupted, and a default sharing (e.g., the constant sharing of 0) is taken.

In the protocol of [BGW88], the share of processor P_i is $s_i = f(\alpha_i) = f_i(0)$, and the second dimension (i.e., the $\widetilde{f}_i$ polynomials) of the sharing is not used. In our scheme, the share of processor P_i is the polynomial $f_i(x)$ (and in particular $s_i = f_i(0)$), as well as the share-shares $s_{ji} = \widetilde{f}_i(\alpha_j) = p(\alpha_i, \alpha_j)$ (for $j = 1, \ldots, n$).

In order to analyze the security of this secret-sharing protocol we distinguish two cases: (a) If the dealer is honest, all shares and share-shares of honest processors will be consistent, and only values held by corrupted processors can be published. No honest processor will accuse the dealer, hence there will be at most t accusations. Clearly, in this case the outcome will be a proper t-sharing. (b) If the dealer is corrupted, all shares and share-shares of honest processors that did not accuse the dealer will be consistent and lie on a polynomial $p'(x, y)$ of degree t. If there were k accusation in the first accusation stage, then still there are at least $n-t-k$ honest non-accusing processors. If the polynomials broadcast by the dealer as reply to an accusation lie in p', then no honest processor will accuse the dealer in the second accusation stage, and we are done. If they do not lie in p' (but have degree t), then each polynomial has at most t points in common with p', hence at least $n - t$ points are different, and at least $n - 2t - k$ honest processors (that did not yet accuse the dealer) will detect this inconsistency and accuse the dealer in the second accusation stage. This sums up to a total of at least $n-2t-k+k > t$ accusations, and

the dealer is disqualified. In this case it is legitimate to take some default value as the dealer's secret.

3.2.2 Secret Reconstruction

Let P be the designated processor supposed to receive a value s that is t-shared among the processors in $\mathcal{P}$ with the polynomials $f(x)$ and $f_1(x), \ldots, f_n(x)$. The idea of secret reconstruction is that every processor P_i opens his committed share s_i towards P: First, every processor $P_i \in \mathcal{P}$ sends the polynomial $f_i(x)$ and the share-shares $s_{1i}, \ldots s_{ni}$ to P. Then, P interpolates the secret s from those shares $s_i = f_i(0)$ where $f_i(x)$ is consistent with all but (at most) t share-shares s_{ij} (i.e., the ones that match with the commitment). Note that this protocol needs neither error correction nor broadcast.

There are two privacy issues to be considered: First, no processor but P should learn anything new in this protocol, and second, P should not learn anything about any other shared value than s. The first issue is satisfied trivially, because only P receives any data. The second issue is satisfied because only shares and share-shares of s are involved in the protocol, and the sharing of s is independent of any other sharing.

The correctness can be proven as follows: At most t (corrupted) processors hand a bad polynomial $f_i'(x) \neq f_i(x)$ to P. However, these bad polynomials either have degree greater than t, or they match at most at t positions with $f_i(x)$, and hence at least $n - t$ processors have an incompatible share-share, and at least $n - 2t > t$ have handed the correct (incompatible with $f_i'(x)$) share-share to P. Hence, P can uniquely distinguish between good polynomials $f_i(x)$ and bad polynomials $f_i'(x)$. There are at most t bad polynomials, hence at least $n - t > t$ good ones, and the shares $f_i(0)$ of these good polynomials are sufficient to interpolate the secret s.

3.2.3 Addition and Linear Functions

Multiplication by scalars and addition of shared values, and hence the computation of any linear function, can be performed (without communication) by each processor doing the corresponding computation on his shares and share-shares and keeping the result as a share of the new value.

3.2.4 Multiplication

The idea of the protocol for computing the t-shared product c of two t-shared values a and b is the same as in the passive model: Every processor secret-shares the product of his respective shares of a and b, and then a valid sharing of c is obtained as a linear combination (according to Lagrange interpolation) of these sharings. However, in the active model, these computations must be verifiable.

Therefore, first every processor P_i upgrades his commitment to his share a_i to a full-fledged (two-dimensional) t-sharing of a_i, and does the same for b_i. Then, P_i t-shares the product $d_i = a_i b_i$ (using the VSS protocol of Section 3.2.1), and proves that he shared the right value d_i. Finally, a t-sharing of c can be computed as a linear combination of the t-sharings of $d_1, \ldots, d_n$, and hence, every processor can compute his share and his share-shares of c as a linear combination of his shares and his share-shares of $d_1, \ldots, d_n$. Some (up to t) of the processors may be disqualified when upgrading the commitments a_i an b_i, or in the equality proof. Then, only the product shares d_i of the other (at least $n - t > 2t$) processors are used for interpolation.

In the sequel, we present the necessary sub-protocols. For simpler presentation, we omit one level of indices, and denote the values the processor P is committed to by a and b, and the product to be shared by d.

3.2.4.1 Upgrading a commitment to a sharing

In order to upgrade the commitment of a (with polynomial $f_a(x)$) to a t-sharing of a, the VSS protocol of Section 3.2.1 is applied, where the dealer's random polynomial $p(x, y)$ satisfies $p(0, x) = f_a(x)$. During the VSS protocol, up to t processor accuse the dealer and subsequently have to adjust their share. However, the at least $t + 1$ honest processors who do not have to adjust their share uniquely define the commitment polynomial f, and hence, if the dealer is not disqualified, then the secret of the sharing is equal to the secret of the commitment.

3.2.4.2 Proving that $d = ab$

Proving that indeed the sharing of d contains the product of a and b is done as follows: Let $f_a(x)$, $f_b(x)$, and $f_d(x)$ denote the polynomials,

which a, b, and d, respectively, are shared with, where additionally, every processor is committed to all his shares. The idea of the proof is to verify that the polynomial $f_a(x)f_b(x) - f_d(x)$ is a polynomial (of degree $2t$) with a zero free-coefficient. Therefore, the dealer P is to share t random values with random polynomials $f_1(x), \ldots, f_t(x)$ such that

$$f_d(x) + xf_1(x) + \ldots + x^t f_t(x) = f_a(x)f_b(x).$$

These polynomials are chosen at random, except that the above equation must hold. Then, every processor P_i verifies whether the equation holds at α_i, and broadcasts either a confirmation or an accusation. If a processor accuses the dealer, he then must open his commitments for his share of a, his share of b, his share of d, and for all his shares of the t random sharings. Then, every other processor can verify whether the accusation is legitimate and the dealer P is to be disqualified.

When the dealer P is honest in this sub-protocol, then no accusation against him will be accepted, and he passes the protocol. On the other hand, whenever the (honest or malicious) dealer passes the protocol, then the above equation holds at least at $n-t$ points, which for polynomials of degree $2t < n-t$ means that they are equal, and hence $f_c(0) = f_a(0)f_b(0)$.

3.2.5 Random Field Elements

A sharing of a random field elements can be achieved in the same way as in the passive model, except that the corresponding resilient sub-protocols are used.

3.3 Active Model with Broadcast

The protocol for the active model with broadcast, tolerating a minority of the processors to be actively corrupted, is out of the scope of this text. Just for completeness, we give a very sketchy overview of the protocol of [CDD$^+$99]. Other protocols that are secure under the same assumptions (but are less efficient) were proposed in [RB89] and [Bea91b]. The achievements of this protocol is formally stated in the following theorem:

Theorem 3 *In the active model with broadcast, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification unconditionally securely if the adversary corrupts at most $t < n/2$ of the processors. The computation is polynomial in n and linear in the size of the circuit.*

The construction of this protocol is along the lines of the protocol for the active model, as described in Section 3.2, but there is one level more of sharings: The secret is (Shamir-) shared with a polynomial of degree t, each of the shares is again shared with a polynomial of degree t, and every processor is committed to each of his share-shares. Commitments are realized with a technique called *information checking*: In the VSS protocol, the dealer P selects at random a bivariate polynomial $p(x, y)$ and hands to every P_i the share-shares $p(\alpha_i, \alpha_1)$, ... , $p(\alpha_i, \alpha_n)$ and $p(\alpha_1, \alpha_i)$, ... , $p(\alpha_n, \alpha_i)$. Additionally, he commits P_i with respect to every other processor P_j to the sent share-shares. In order to do so, for every processor P_j, the dealer hands some proving information to P_i and some checking information to P_j. Later, P_i can hand any of his share-shares to P_j, plus the proving information for this share-share, and P_j is able to verify (by using his checking information) whether indeed the received value originates from the dealer. This proving and checking information is linear for each triple of processors: When P has committed P_i with respect to P_j to several values, then P_i can send and prove any linear combination of these values to P_j.

Secret reconstruction is then trivial: Every processor P_i broadcasts his polynomial $f_i(x)$, where $f_i(0)$ is his share. Then, all other processors verify whether their respective share-share lies on this polynomial, and if not, they reveal and *prove* (by using information checking) their share-share towards all other processors. Against at least $n - t$ of the broadcast polynomials $f_i(x)$, no valid complaint will be cast, and the $n - t > t + 1$ shares $f_i(0)$ are sufficient to interpolate the secret.

Additions and linear functions on shared values can be computed non-interactively. Both the secret-sharing and the information-checking scheme are linear, and in order to compute a linear function of shared values, every processor computes his share, his share-shares, his proving and his checking information for the function value by applying the linear function on the corresponding values of the functions inputs.

In the multiplication protocol, first every processor shares (using VSS) the shares of the factors, and proves that the shared value is indeed his factor share. Then, each processor shares the product of his factor shares, and proves that the shared value is indeed the product. Finally, a valid sharing of the product is obtained as a linear combination (according to Lagrange's formula) of these product sharings.

In the multiplication sub-protocol, a processor can refuse cooperation and can fail the proofs. This can be detected, and the processor in charge

can be disqualified. However, the multiplication protocol cannot be proceeded. In [CDD+99] it is suggested that when processors are disqualified, then the whole protocol is restarted from the beginning, without involving the disqualified processors. This is inefficient, and furthermore, it does not capture the case of general trusted party simulation. The much more natural solution is to reconstruct the shares (of both factors) of the disqualified processor (similar to [RB89]), and to repeat the multiplication protocol with the reconstructed shares as public constants. This method was suggested by Fehr [Feh00].

3.4 Fail-Stop Model

In the fail-stop model, the adversary may only make processors crash. Privacy is no issue. The adversary can be allowed to fail-corrupt any number of processors, as stated by the following theorem:

Theorem 4 *In the fail-stop model, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification perfectly securely if the adversary corrupts any $t < n$ of the processors. The computation is polynomial in n and linear in the size of the circuit.*

The protocol for this model is based on replication. A specification (π_0, τ) is turned into a protocol π as follows: Whenever a processor P_i sends a value to τ, then instead he broadcasts the value to all processors in $\mathcal{P}$ (e.g., by using the broadcast protocol of [LF82]). The broadcast protocol ensures that either every or no (uncorrupted) processors in $\mathcal{P}$ learns the value, even when P_i fails during the sub-protocol. All computations of τ are evaluated by all processors in $\mathcal{P}$ individually. Both the security and the efficiency of this protocol are obvious.

3.5 Mixed Model with Perfect Security

The following theorem states the bounds for the mixed model with perfect security [FHM98].[6]

[6] Note that the theorem in the original paper [FHM98] for the model with perfect security turned out to be false [Dam99], and we include a corrected version of the theorem and of the proof.

Theorem 5 *In the mixed model with or without broadcast channels, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification perfectly (t_a, t_p, t_f)-securely if and only if $3t_a + 2t_p + t_f < n$. The computation is polynomial in n and linear in the size of the circuit.*

For the protocol construction, we do not assume the existence of broadcast channels (they will be simulated with a broadcast protocol [GP92]); however, we show that the stated condition is even necessary when assuming the existence of broadcast channels.

The construction of perfectly secure protocols for the mixed model is based on the protocol for the active model of Section 3.2. To each processor P_i, a unique non-zero element $\alpha_i \in (\mathbb{F} \backslash \{0\})$ is assigned. The degree of all sharings (also of the second-level sharings) is $t = t_a + t_p$. This degree ensures the privacy even when the actively corrupted processors reveal all their secrets.

3.5.1 Verifiable Secret Sharing

The VSS protocol of Section 3.2.1 is slightly modified to ensure security in the mixed model:

1. DISTRIBUTION. The dealer P selects at random a polynomial $p(x, y)$ of degree $t = t_a + t_p$ in both variables, where $p(0, 0) = s$, and sends the polynomials $f_i(x) = p(x, \alpha_i)$ and $\widetilde{f}_i(y) = p(\alpha_i, y)$ to processor P_i (for $i = 1, \ldots, n$).
2. CONSISTENCY CHECKS. Each pair of processors P_i, P_j (for $1 \leq i, j \leq n$) checks whether $f_i(\alpha_j) \stackrel{?}{=} \widetilde{f}_j(\alpha_i)$. For this, P_i sends $f_i(\alpha_j)$ to P_j, and P_j checks whether the received value is equal to $\widetilde{f}_j(\alpha_i)$. If a processor P_j does not receive any value of a processor P_i, then P_j *does not complain* about this cross-over point (it is sufficient to verify the cross-over points of all honest processors).
3. COMPLAINT STAGE. Every processor broadcasts a message (containing one bit) indicating whether all consistency checks were successful (or no value was received at all) or at least one test failed. In case of a complaint, the processor afterwards broadcasts a bit-vector, where the j-th bit indicates whether or not the processor has observed an inconsistency with processor P_j. The dealer answers the complaints by broadcasting the corresponding correct values.

4. FIRST ACCUSATION STAGE. Every processor P_j who observes more than t_a inconsistencies or discovers that the dealer's answers to complaints contradict his own values broadcasts an accusation. In such a case the dealer broadcasts both polynomials $f_j(x)$ and $\widetilde{f}_j(y)$.
5. SECOND ACCUSATION STAGE. Every processor who discovers that a polynomial broadcast by the dealer as reply to an accusation in the first accusation stage contradicts his own shares, broadcasts an accusation. This accusation need not be replied by the dealer.

If in total more than t_a processors have accused (in both accusation stages), or if the dealer did not answer all the complaints and accusations, clearly the dealer is corrupted, and a default sharing (e.g., the constant sharing of 0) is taken.

In order to analyze the security of this secret-sharing protocol we distinguish two cases: (a) If the dealer is honest, all shares and share-shares of honest processors will be consistent, and only values held by actively corrupted processors can be published. Only actively corrupted processors will accuse the dealer, hence there will be at most t_a accusations. Clearly, in this case the outcome will be a proper t-sharing. (b) If the dealer is corrupted, all shares and share-shares of honest processors that did not accuse the dealer will be consistent and lie on a polynomial $p'(x, y)$ of degree t. If there were k accusations in the first accusation stage, then still there are at least $n - t_a - t_f - k \geq 2t_a + 2t_p + 1 - k$ (honest or passively corrupted) processors with consistent shares. The polynomials broadcast by the dealer as reply to an accusation either lie in p', in which case no honest and no passively corrupted processor will accuse the dealer in the second accusation stage, or does not lie in p', in which case at least $2t_a + 2t_p + 1 - k - t \geq t_a + 1 - k$ processors will accuse in the second stage, and in total there were at least $t_a + 1$ accusations, and the dealer is disqualified.

3.5.2 Secret Reconstruction

In order to reconstruct a shared secret, all processors open their committed share towards the processor P who is to learn the secret. The commitments are opened as follows: Every processor P_i sends the polynomial $f_i(x)$ and the share-shares s_{ji} (for $j = 1, \ldots, n$) to P. Then P verifies each polynomial $f_i(x)$: If he received more than t_a share-shares s_{ij} not satisfying $f_i(\alpha_j) \stackrel{?}{=} s_{ij}$, then obviously $f_i(x)$ is bad, and else, $f_i(x)$ is good.

Then, P interpolates the secret s from the given shares $f_i(0)$ of the good polynomials $f_i(x)$ by using Lagrange's formula.

3.5.3 Multiplication

The multiplication protocol is along the lines of the multiplication protocol of Section 3.2.4, where all invocations of the VSS protocol are replaced by invocations to the VSS for the mixed model, given above.

3.5.4 Tightness

In order to prove the necessity of the condition $3t_a + 2t_p + t_f < n$ even when broadcast channels are available, assume for the sake of contradiction that for some t_a, t_p, and t_f with $3t_a + 2t_p + t_f \geq n$ every function can be computed perfectly (t_a, t_p, t_f)-securely. Trivially, each such protocol is also a (t_a, t_p)-secure protocol among $n' = n - t_f$ processors, simply by considering the case that the adversary fail-corrupts the last t_f processors at the very beginning of the protocol. Then one can construct a protocol for three processors P_1, P_2, and P_3, where P_1 plays for $t_a + t_p$ processors, P_2 plays for $t_a + t_p$ other processors, and P_3 plays for the remaining at most t_a processors. This new protocol is secure with respect to an adversary that either passively corrupts P_1, or passively corrupts P_2, or actively corrupts P_3.

Assume that the specification requires to compute the logical AND of two bits x_1 and x_2 held by P_1 and P_2, respectively, and assume for the sake of contradiction that a protocol for this specification is given. Due to the requirement of perfect privacy, P_1 must not send any (joint) information about his bit x_1 to P_2, neither over the direct channel from P_1 to P_2, nor over the broadcast channel (if available), before he knows x_2 (once P_1 knows that $x_2 = 1$ he can reveal x_1). Similarly, P_2 will not send any information about x_2 to P_1 before he knows x_1. Hence the only escape from this deadlock would be to use P_3. However, as the transcripts of the channel from P_2 to P_1 and the transcript of the broadcast channel do not contain any joint information about x_2, there exist some behavior for P_3 which makes P_1 receive the wrong output. Hence, an actively corrupted P_3 can just behave randomly (ignore all received messages and send random bits whenever a message must be sent), and with some (possibly negligible) probability, P_1 will receive the wrong output, contradicting the perfect correctness of the protocol.

3.6 Mixed Model with Unconditional Security and Broadcast

In this section, we consider the mixed model with unconditional security and broadcast. The main result in this model is stated by the following theorem [FHM98]:

Theorem 6 *In the mixed model with broadcast channels, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification unconditionally (t_a, t_p, t_f)-securely if and only if $2t_a + 2t_p + t_f < n$. The computation is polynomial in n and linear in the size of the circuit.*

As models with broadcast are not in the main focus of this thesis, we only sketch the protocol for that model. Based on the protocol for the active model with broadcast (sketched in Section 3.3), we outline how to construct an unconditionally secure protocol for the mixed model with broadcast.

3.6.1 Protocol

First, consider the case that $t_f = 0$. In this scenario, the protocol of Section 3.3 can be employed. This protocols tolerates up to $t < n/2$ actively corrupted processors in the stated model. Trivially, every (t_a, t_p)-adversary can be seen as an active t-adversary with $t = t_a + t_p$ (which is not making use of all his power), and hence this protocol is also (t_a, t_p)-secure for any t_a, t_p with $2t_a + 2t_p < n$.

In order to tolerate adversaries with $t_f > 0$, we need an extended protocol. We start with the protocol of Section 3.3. We set the degree of all sharings to $t = t_a + t_p$. In contrast to the active model, in the mixed model a disqualified processor can either be actively corrupted or fail-corrupted, and his shares of the factors must not be reconstructed if the processor is fail-corrupted. Therefore, we need a more involved solution for treating disqualified processors.

If during the multiplication protocol a processor is disqualified, then he is excluded from all further computation. Generally, the shares of a disqualified processor must not be revealed. Indeed, the shares of up to t_f disqualified processors can simply be omitted (the interpolation of the degree-$2t$ polynomial is possible with $n - t_f > 2(t_a + t_p)$ processors).

Thus, the first t_f times that some processor is disqualified the processor is excluded from the computation, but no further steps are taken. If more than t_f processors are disqualified, then there are at most t_f fail-corrupted processors among them, and the other disqualified processors (say k) must be actively corrupted. Hence, among the remaining (non-disqualified) processors there are at most $t_a - k$ actively corrupted processors. At this point, every intermediate value is reshared using a scheme for $n' = n - k$ processors, tolerating only $t'_a = t_a - k$ actively corrupted processors (and t_p passive corruptions). In order to do so, for every intermediate value, every processor verifiably secret shares his share of that value among the n' remaining processors by using polynomials of degree $(t_a - k) + t_p$, and proves that the shared share is equal to his original share. This can easily be verified by computing the difference of these two shares (in a distributive manner, i.e., every processor subtracts one share-share from the other) and by verifying that the difference is zero. Finally, the secret is interpolated in a distributive manner (interpolation is linear). This results in every processor having a reduced-degree share of the secret which is committed by a reduced-degree polynomial. If during this degree-reduction protocol some processors refuse to cooperate, these processors are also disqualified and the reduction protocol restarts (with an increased k). The number of restarts is limited by t_a because at most $t_a + t_f$ processors can ever be disqualified.

3.6.2 Tightness

In order to prove the necessity of the condition $2t_a + 2t_p + t_f < n$ assume for the sake of contradiction that for some t_a, t_p, and t_f with $2t_a + 2t_p + t_f \geq n$ every function can be computed perfectly (t_a, t_p, t_f)-securely. Trivially, each such protocol is also a (t_a, t_p)-secure protocol among $n' = n - t_f$ processors, simply by considering the case that the adversary fail-corrupts the last t_f processors at the very beginning of the protocol. Hence, we have constructed a protocol which is secure against a passively-corrupted majority, contradicting Theorem 2 of [BGW88].

3.7 Mixed Model with Unconditional Security without Broadcast

The following theorem states the necessary and sufficient conditions for secure MPC to exist in a model with unconditional security and without

broadcast.

Theorem 7 *In the mixed model without broadcast channels, a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n processors can compute every specification unconditionally (t_a, t_p, t_f)-securely if and only if $2t_a + 2t_p + t_f < n$ and $3t_a + t_f < n$. The computation is polynomial in n and linear in the size of the circuit.*

This security is achieved by the protocol of the previous section, when simply all invocations to the broadcast primitive are replaced by invocations to a sub-protocol for Byzantine agreement. For $3t_a + t_f < n$, such sub-protocols are exist [MP91, GP92]. The necessity of these bounds follows immediately from the necessity of the bound in the unconditional model with broadcast and the necessity of $3t_a < n$ for broadcast (and the fact that broadcast is a particular function for MPC).

3.8 Framework for Efficient Resilient Protocols

3.8.1 Introduction

Distributed protocols resilient against misbehavior of some of the processors require in general much more communication than their private (but non-resilient) counterparts, even when no cheating occurs. The reasons for this contrast are two-fold: First, in a model where processors might deviate from the protocol, expensive consistency checks must be performed frequently, and agreement must be reached on whether or not faults occurred. Second, if indeed at least one processor misbehaves, then inconsistencies will occur, and costly fault-recovery procedures must be applied. Note that the consistency checks are necessary even when no cheating occurs, whereas fault recovery is necessary only when at least one processor misbehaves.

In this section, we describe a new framework for efficient resilient protocols that overcomes these disadvantages. The key idea is to eliminate at least one malicious processor (and potentially some honest processors) each time a fault is detected. Hence the number of fault-recovery invocations is bounded by the maximal number of corrupted processors and is independent of the length of the protocol. Furthermore, the resulting rare occurrence of faults allows to reduce the frequency of consistency checks and thereby to significantly reduce the communication-overhead caused by them.

The techniques presented in this section apply to many applications in several models, including those relying on intractability assumptions. The adversary can be static or adaptive, but not mobile: A mobile adversary may release some of the corrupted processors during the protocol execution and thereby regain the capability of corrupting new processors, which contradicts the idea of elimination of corrupted processors.

3.8.2 Incorporating Resilience into a Private Protocol

We consider a private protocol that proceeds in rounds (e.g., in each round one gate is evaluated), and wish to perform this protocol in a resilient manner. In contrast to the classical approach to resilient protocols, where after each round some consistency checks are performed and agreement on whether or not a fault occurred is reached, we divide the protocol into *segments*, each consisting of a sequence of rounds, and only at the end of each segment the consistency of the data held by the processors is checked and the processors agree on whether or not a fault occurred (*fault detection*). If a fault is detected, then a set containing at least a certain number of cheaters is identified (*fault localization*), the processors in the set are eliminated from the further protocol execution (*processor elimination*), and the failed segment is repeated (*fault correction*). Depending on the protocol, it might be necessary to take special measures to ensure privacy after a failed round. In this case, after each round some checks must be performed, but no agreement must be reached on the fact whether or not a fault occurred (*weak fault detection*).

During a protocol consisting of m rounds, the classical approach invokes m times fault detection and, if at least one processor misbehaves permanently, m times fault-recovery. In our approach, where the protocol is divided into segments of m_s rounds, only the weak fault detection is invoked m times. Fault detection is performed m/m_s times, and fault localization, processor elimination, and fault correction are invoked at most t times. By selecting m_s appropriately, the overhead for the (in total up to t) repetitions of a segment will not dominate the total complexity of the protocol, and the costs for fault detection and fault localization are independent of m (and polynomial in n). In many applications, this will significantly reduce the overall complexity of the protocol.

We now describe the steps in more detail:

1. **Private computation with weak fault detection.** All rounds of the segment are computed according to the private computation. The

computation of this step has to be *verifiable*, i.e., it must be possible to check later (see below) whether or not any faults occurred, but *robustness* is not required, i.e., if faults occur, then the computation may fail (in such a case it must be possible to perform an appropriate fault localization, see below). If privacy is required, then after each round, consistency checks are performed, and every processor sends to every other processor one bit indicating whether or not he observed an inconsistency. A processor who observed or was informed about an inconsistency will use random (or default) dummy values unrelated to the actual values in all further rounds of the segment.

2. **Fault detection.** The goal of fault detection is to reach agreement on whether or not a fault occurred during the current segment. Typically, fault detection is achieved by having every processor broadcast (with a protocol for Byzantine agreement) a binary message according to whether or not he observed or was informed about an inconsistency in any round of the current segment. The following Steps 3. to 5. are performed if and only if a fault is detected.

3. **Fault localization.** The purpose of fault localization is to find out which processors are corrupted or, because agreement about this can usually not be reached, at least to narrow down the set of processors containing the cheaters. The output of an (r, p)-*localization* is a set $\mathcal{D}$ with $|\mathcal{D}| = p$ processors, guaranteed to contain at least r cheaters.

4. **Processor elimination.** The set $\mathcal{D}$ agreed upon during fault localization is eliminated from the further computation. In general, after eliminating the processors in $\mathcal{D}$, the protocol cannot be continued immediately, but it must be transformed to capture the new setting with $n - p$ processors and at most $t - r$ cheaters. Of course, p and r must be such that this transformation is possible and such that the transformed protocol can be safely continued. Often, the transformation of the protocol is not performed immediately, rather in each round only those parts of the protocol are transformed that are needed.

5. **Fault correction.** Since some processors are eliminated whenever a fault is detected, faults can be corrected by repeating the current segment of the protocol.

3.9 Efficient MPC Protocol with Perfect Security

In this section we present a construction for efficient multi-party computation in the secure-channels model (without broadcast channels), based on the framework with processor-elimination from the previous section and the protocol for the active model from Section 3.2. We restrict to the active model, but the protocol can easily and naturally be extended to withstand mixed adversaries. Furthermore, we only consider threshold adversaries, although the framework is not restricted to threshold security. Finally, we focus on secure function evaluation (SFE) where the function is specified as an arithmetic circuit over the field, but the necessary modifications to capture TPS will be mentioned.

The main result of this section is stated in the following theorem:

Theorem 8 *A set of n processors with at most $t < n/3$ of them being actively corrupted, can securely compute a function over a finite field $\mathbb{F}$, using $\mathcal{O}(d+n^2)$ communication rounds and with total communication complexity $\mathcal{O}(n_I n^4 + mn^3 + n_O n^2)$ field elements, where n_I and n_O denote the number of inputs and outputs, respectively, m denotes the number of multiplications and d the multiplicative depth of the circuit computing the function.*

3.9.1 Main Protocol

The protocol follows the classical approach for secure multi-party computation: First, each processor secret-shares his input(s) among the processors. Second, the circuit is evaluated with the shared values. Third, the output value(s) are reconstructed towards the authorized processors.

According to the framework of Section 3.8, the circuit will be divided into segments. If the evaluation of a segment fails, then some processors are eliminated and the segment is repeated. We will uniquely use $(1,2)$-localizations, i.e., a set with two processors is identified and eliminated, where one of the two processors is guaranteed to be corrupted. Clearly, *all* processors must be able to provide input and receive output, including processors that are eliminated in the protocol evaluation (also honest processors can be eliminated). Therefore, the sharing of the input values is performed before that first segment in a resilient manner (without using the processor elimination technique). Secret-reconstruction can be performed towards an eliminated processor (this processor only receives values and cannot cause inconsistencies).

3.9.1.1 Sharing

The sharing is the one that was introduced in the active model (see Section 3.2): To each processor P_i a unique public value $\alpha_i \in \mathbb{F} \setminus \{0\}$ is assigned. A value s is *t-shared* among the processors if there exist degree-t polynomials $f(x)$ and $f_1(x), \ldots, f_n(x)$ with $s = f(0)$ and $f_i(0) = f(\alpha_i)$, and the information held by processor P_i is the share $s_i = f(\alpha_i)$, the polynomial $f_i(x)$, and the share-shares $s_{ji} = f_j(\alpha_i)$ (for $j = 1, \ldots, n$).

Note that during the protocol execution, all sharings will be of degree t. Even once processors are eliminated (and the number of corrupted players is strictly smaller than t), still all sharings are of degree t.

3.9.1.2 Segmentation

Due to the linearity of the secret-sharing scheme, linear functions of shared values can be computed non-interactively, and hence only multiplication gates are relevant for the communication complexity.

The goal is to divide the circuit with m multiplication gates and multiplicative depth d into segments, such that

- the number of multiplication gates in each segment is at most $m_s = \lceil m/n \rceil$, and
- the multiplicative depth of each segment is at most $d_s = \lceil d/n \rceil$,

where of course the number of segments should be small. We present a very simple algorithm which results in less than $2n$ segments.

First, we assign a level to each multiplication gate, according to the graphical representation of the circuit: The *level* of a gate is the number of multiplication gates on the longest path from an input of the circuit to the gate. Furthermore, we define a full order on the gates, which satisfies the partial order given by the levels (but otherwise is chosen arbitrary). Finally, we iteratively define the segments. The first segment contains the first k_1 gates (according to the above order), where k_1 is the maximal number such that both above conditions are still satisfied for this segment. The second segment contains the next k_2 gates, where k_2 is the maximal number still satisfying the above two conditions, and so on.

One can easily verify that the number of segments is smaller than $2n$: Every segment (but the last) either contains m_s gates, or it contains all (remaining) gates of d_s levels of the circuit. Obviously, n segments

of either type contain all gates of the circuit, hence there can be at most $n + (n-1)$ segments, as claimed.

Due to player elimination, up to t segment evaluations will fail and must be repeated. Hence, in total less than $3n$ segments are evaluated, with less than $3m$ multiplication gates and with depth less then $3d$.

3.9.1.3 Protocol overview

Let $\mathcal{P}$ denote the set of processors, where $n = |\mathcal{P}|$, and $t < n/3$ the upper bound on the number of cheaters. During the computation, processors can be eliminated, and then $\mathcal{P}'$ will denote the set of remaining processors, $n' = |\mathcal{P}'|$, and t' the upper bound on the number of cheaters in this set.

0. Set $\mathcal{P}' := \mathcal{P}$, $n' := n$, $t' := t$.
1. Input stage: Every processor P providing input secret-shares his input-value (Sect. 3.9.2).
2. Computation stage (Sect. 3.9.3): For each segment of the circuit:
   2.1 For each $P_i \in \mathcal{P}'$: set $\mathit{fault\text{-}detected}_i := \mathit{false}$.
   2.2 For each gate in the segment (all gates at the same level can be evaluated in parallel):
   - If the gate is linear: Call the sub-protocol for the evaluation of linear functions (Sect. 3.9.3.1).
   - If the gate is a multiplication gate: Call the multiplication sub-protocol (Sect. 3.9.3.2). Processors with $\mathit{fault\text{-}detected}_i = \mathit{true}$ use default shares. If P_i observed a fault in the multiplication sub-protocol, or was informed about a fault in weak fault detection, then set $\mathit{fault\text{-}detected}_i := \mathit{true}$.

   2.3 For each $P_i \in \mathcal{P}'$, broadcast $\mathit{fault\text{-}detected}_i$. If at least one processor broadcasts $\mathit{fault\text{-}detected}_i = \mathit{true}$, then the segment fault-localization procedure is invoked to find a $(1,2)$-localization $\mathcal{D}$, and $\mathcal{P}'$ is set to $\mathcal{P}' \setminus \mathcal{D}$, n' is set to $n' - 2$, t' is set to $t' - 1$, and step 2. is restarted for the same segment.
3. Output stage: For every processor P that is to receive output: Call the sub-protocol for receiving output (Sect. 3.9.4).

3.9.1.4 Trusted Party Simulation TPS

In the context of TPS, the segmentation must be chosen more carefully. For the sake of simplicity, we assume that the specification only contains transmit statements from or to the trusted party τ, but no transmissions between "normal" processors.[7] This captures the most relevant specifications of on-going cooperations among processors.

As for secure function evaluation, the segmentation is chosen such that each segment has limited length and depth. Furthermore, we must ensure that repetition of a segment does not compromise security of the computation. Therefore, we require that each segment either cannot fail (there is no multiplication gate in the segment), or the trusted party τ behaves like a black whole in this segment (processors send values to τ, who does arbitrary probabilistic computations, but no processor receives anything from τ), or τ behaves like a deterministic emitter (arbitrary deterministic computations, send values to processors). These requirements are formalized in the sequel:

- The number of multiplication statements for τ (i.e., statements of the form $\mathit{comp}(\tau, *, \cdot, \cdot, \cdot)$) in each segment is at most $m_s = \lceil m/n \rceil$,
- the multiplicative depth of these statements (i.e., the length of the longest path in the dependency graph) of each segment is at most $d_s = \lceil d/n \rceil$, and
- (at least) one of the three following conditions is satisfied, namely either
  - the segment contains no multiplication statement for τ ($\mathit{comp}(\tau, *, \cdot, \cdot, \cdot)$), or
  - the segment contains no transmit statement from τ ($\mathit{transmit}(\tau, \cdot, \cdot, \cdot)$), or
  - the segment contains neither a transmit statement to τ ($\mathit{transmit}(\cdot, \tau, \cdot, \cdot)$), nor a statement instructing τ to choose a random field element ($\mathit{comp}(\tau, \mathsf{ran}, \cdot)$).

The first two conditions are just equivalent to the conditions in the scenario of SFE, and they ensure that the repetitions of any up to t segments will not dominate the overall complexity of the protocol. The third condition ensures that every segment either cannot fail, or that it can be repeated smoothly. The reason for this to hold is as follows: A segment without multiplication statements for τ cannot fail, hence there is nothing

[7] Conditions for the case with transmissions between processors can be derived, but they are complicated. Furthermore, any specification can be manipulated to satisfy this requirement by using τ as a "hub".

to take care of. A segment that contains no transmit statements from τ can be repeated, as the adversary gains no information during the failed execution (there was no *transmit*-statement to a potentially corrupted processor in this segment). During the execution of a segment containing transmit statements from τ to processors, the adversary can gain additional information about the trusted party's view, but when repeating the segment, the trusted party will perform exactly the same operations on the same data, and the joint view of all uncorrupted processors (including τ) is equally distributed after the first run and after the repetition.

Note that these additional requirements influence the number of segments to be evaluated. However, in most applications of on-going computations, the specification contains blocks where processors hand inputs to τ, blocks where τ securely performs some computation on these values, and blocks where τ hands some outputs to some or all of the processors, and these additional requirement do not increase the number of segments significantly. However, we will not analyze the exact complexity of such on-going computations.

3.9.2 Input Stage

In the input stage, every processor secret-shares his input(s). We use the verifiable secret-sharing scheme from the active protocol, as described in Section 3.2.1.

In the context of TPS, providing input can happen to be performed after $k > 0$ times processors have been eliminated. In this case, the VSS protocol must be slightly modified: The processor set is $\mathcal{P}'$ (with $n' = |\mathcal{P}'| = n-2k$), and the degree of all involved polynomials is t. The sharing is accepted if there are at most $t' = t - k$ accusations, else it is rejected. Let ℓ denote the number of accusations in the first complaint stage. Then there are still at least $n'-t'-\ell$ honest non-accusing processors, and a false polynomial broadcast by the dealer would cause at least $(n' - t' - \ell) - t$ further accusations, which adds up to a total of $n' - t' - t = n - 2t - k$ accusations, which is greater than $t' = t - k$ for $t < n/3$.

3.9.3 Computation Stage

The computation of the circuit proceeds segment by segment. We denote the current set of processors with $\mathcal{P}'$, where $n' = |\mathcal{P}'|$, and the current upper bound on the number of cheaters in $\mathcal{P}'$ with t'. Without loss of

generality, we assume that $\mathcal{P}' = \{P_1, \ldots, P_{n'}\}$. Note that t' denotes the upper bound on the number of cheaters, but still the degree of all sharings will be t.

A segment is computed as follows: First, the gates of the segment are computed. Linear functions can be computed robustly (as no communication is needed). In contrast, the computation of multiplication gates is private and verifiable, but not robust. At the end of each multiplication sub-protocol, the (honest) processors inform each other whether or not they observed an inconsistency in a weak fault detection procedure. If a processor observed such an inconsistency, or was informed about one in weak fault detection, then he continues the computation of the segment with default values independent of the actual shares. At the end of each segment, fault detection is performed, and if necessary, fault localization, processor elimination and fault correction.

3.9.3.1 Linear functions

Linear functions are evaluated exactly like in the active protocol of Section 3.2.3. Let $\mathcal{L}$ be a linear function, and assume that the values $a, b, \ldots$ are t-shared with polynomials $f(x), f_1(x), \ldots, f_{n'}(x)$, $g(x), g_1(x), \ldots, g_{n'}(x)$, $\ldots$, respectively. Due to the linearity of $\mathcal{L}$, the polynomials $h(x) = \mathcal{L}\big(f(x), g(x), \ldots\big)$ and $h_i(x) = \mathcal{L}\big(f_i(x), g_i(x), \ldots\big)$ define a t-sharing of $c = \mathcal{L}(a, b, \ldots)$. Hence, processor P_i can compute his share of c as $h_i(x) = \mathcal{L}(f_i(x), g_i(x), \ldots)$ and $c_{ji} = \mathcal{L}(a_{ji}, b_{ji}, \ldots)$ (for $j = 1, \ldots, n'$). The privacy of this protocol is trivial (there is no communication), and the correctness is due to the linearity of the sharing.

3.9.3.2 Multiplication

The crucial sub-protocol for multiplication is a *re-sharing* protocol. A re-sharing protocol is a protocol that takes a γ-sharing of a value s and generates an independent δ-sharing of s. This re-sharing is possible in a verifiable (but non-robust) manner if $t' < n' - \gamma$. Privacy can be guaranteed if $t' \leq \gamma$ and $t' \leq \delta$.

The protocol for computing the t-shared product c of two t-shared values a and b proceeds in three steps: First, both inputs a and b are re-shared with degree t'. Second, every processor locally multiplies his respective shares and share-shares of a and b, resulting in a $2t'$-sharing of c. And third, this $2t'$-sharing of c is re-shared to a t-sharing.

We have to show that the necessary (and sufficient) conditions for all re-sharings are satisfied: After a sequence of k $(1,2)$-localizations and eliminations, we have $n' = n - 2k$ and $t' = t - k$. The requirements for the re-sharing are $t' < n' - t$, respectively $t' < n' - 2t'$, and both are satisfied for $3t < n$.

Re-sharing protocol The goal of re-sharing is to transform a γ-sharing of a value s into a proper and independent δ-sharing of s, where $t' < n' - \gamma$, $t' \leq \gamma$ and $t' \leq \delta$. The re-sharing sub-protocol can fail in the presence of malicious processors. However, if it fails, all (honest) processors will learn so, and at the end of the segment, agreement on whether or not such a fault occurred will be reached and the segment will be repeated if necessary.

Roughly speaking, our re-sharing protocol works along the lines of degree reduction of [BGW88, GRR98], but it is significantly more efficient, due to various techniques in the spirit of the processor-elimination framework (cf. Section 3.8).

Assume that s is γ-shared with the polynomials $f(x)$ and $f_1(x), \ldots, f_{n'}(x)$, and processor P_i holds the polynomial $f_i(x)$ (hence his share $s_i = f_i(0)$), and his share-shares $s_{ji} = f_j(\alpha_i)$ (for $j = 1, \ldots, n'$). The value s can be expressed as a linear combination (Lagrange interpolation) of the values $s_1, \ldots, s_{n'}$ [BGW88, GRR98]. Therefore, once the values $s_1, \ldots, s_{n'}$ are δ-shared, the required δ-sharing of s can be computed by a distributed evaluation of the appropriate linear function (as described above). Thus, the re-sharing can be performed as follows: Every processor δ-shares his share s_i, proves that the shared value is indeed s_i, and computes his degree-δ share of s as a linear combination of the received shares of $s_1, \ldots, s_{n'}$.

We describe the steps in more detail:

1. NON-ROBUST VSS. Every processor P_i shares his share s_i with the degree-δ polynomials $h^{(i)}(x)$, $h_1^{(i)}(x), \ldots, h_{n'}^{(i)}(x)$ in a non-robust but verifiable manner. The protocol works like the first two steps of the VSS in the Input Stage (cf. Section 3.9.2):
   a) P_i selects at random a polynomial $p^{(i)}(x, y)$ of degree δ in both variables, where $p^{(i)}(0,0) = s_i$, and sends the polynomials $h_j^{(i)}(x) = p^{(i)}(x, \alpha_j)$ and $\widetilde{h}_j^{(i)}(y) = p^{(i)}(\alpha_j, y)$ to processor P_j (for $j = 1, \ldots, n'$). This implicitly defines the polynomial $h^{(i)}(x) = p^{(i)}(0, x)$.

   b) Each pair of processors P_j, P_k (for $1 \leq j, k \leq n'$) verifies the equality of their common shares. For this, P_j sends $h_j^{(i)}(\alpha_k)$ to P_k, who then checks whether the received value is equal to $\widetilde{h}_k^{(i)}(\alpha_j)$.

2. PROVING CORRECTNESS. Every processor P_i proves that $h^{(i)}(0) = f_i(0)$ by showing that the free coefficient of the polynomial $h^{(i)}(x) - f_i(x)$ is equal zero. This is done in two steps:

   a) Let $\mu = \max(\gamma, \delta)$. P_i computes the polynomial $g^{(i)}(x) := (h^{(i)}(x) - f_i(x))/x$ (whose degree is at most $\mu - 1$), and distributes the shares on $g^{(i)}$ among the processors. For this purpose the non-robust VSS protocol from Step 1 is used, where the corresponding bivariate polynomial, say $q^{(i)}(x, y)$, is chosen randomly so that $q^i(0, x) = g^{(i)}(x)$.

   b) Every processor P_k checks whether $\alpha_k g^{(i)}(\alpha_k) = h^{(i)}(\alpha_k) - f_i(\alpha_k)$.

3. WEAK FAULT DETECTION. Every processor sends to every other processor one bit indicating whether or not any of his consistency checks in Steps 1, 2a and 2b, have failed.
4. LAGRANGE INTERPOLATION. Every processor P_i, who has neither detected nor was informed about any inconsistencies, computes his degree-δ share of s as a linear combination of his shares of $s_1, \ldots, s_{n'}$.

It is easy to see (using basic algebra), that if no processor has reported inconsistencies during the weak fault detection, then the result of re-sharing is a proper δ-sharing of s. Otherwise, if at least one (honest) processor has sent or received a bit indicating inconsistencies, it will be possible to identify a $(1, 2)$-localization (see below).

3.9.3.3 Fault Detection

At the end of the segment, every processor P_i *broadcasts* one bit indicating whether or not an inconsistency was observed by or reported to P_i in one of the re-sharing protocols in the segment. If all processors broadcast a confirmation, then the computation of the segment is completed and the next segment can be started. If at least one processor broadcasts a complaint, then fault localization is invoked.

3.9.3.4 Fault Localization

The goal of fault-localization is to identify a $(1,2)$-localization $\mathcal{D}$, i.e., a set $\mathcal{D} \subset \mathcal{P}$ containing two processors, at least one of them being corrupted. These processors will then be eliminated from the protocol, and hence fault localization is invoked at most t times.

The two processors to be eliminated are selected from the processors involved in the first fault that occurred in the current segment.[8] In order to determine the first fault, every processor who complained during fault detection broadcasts the index (relative to the segment) of the re-sharing protocol, in which for the first time an inconsistency occurred, together with a number denoting the step of the re-sharing protocol in which the fault was detected (Step 1, 2a or 2b), or reported (Step 3). From the broadcast indices (gate-number and step-number), the smallest one is selected. Let P_k denote the processor who complained about the selected re-sharing protocol (if there are several such processors, we select the one with the smallest index k). The method of determining the $(1,2)$-localization $\mathcal{D}$ depends on the step of the re-sharing protocol in which the first fault appeared:

(i) The first fault is in Step 1, i.e., for some i and j, the value $h_j^{(i)}(\alpha_k)$ sent by P_j differs from $\widetilde{h}_k^{(i)}(\alpha_j)$:
P_k broadcasts i, j, and $\widetilde{h}_k^{(i)}(\alpha_j)$. On this request, P_j broadcasts $\widetilde{h}_j^{(i)}(\alpha_k)$, and P_i broadcasts $p^{(i)}(\alpha_k, \alpha_j)$. Given these three values, the set $\mathcal{D}$ is determined as follows:

- If $\widetilde{h}_k^{(i)}(\alpha_j) = h_j^{(i)}(\alpha_k)$, then $\mathcal{D} := \{P_j, P_k\}$, else
- if $p^{(i)}(\alpha_k, \alpha_j) \neq \widetilde{h}_k^{(i)}(\alpha_j)$, then $\mathcal{D} := \{P_i, P_k\}$, else
- if $p^{(i)}(\alpha_k, \alpha_j) \neq h_j^{(i)}(\alpha_k)$, then $\mathcal{D} := \{P_i, P_j\}$.

(ii) The first fault is in Step 2a: analogously to the case (i).

(iii) The first fault is in Step 2b, i.e., for some i the check $\alpha_k g^{(i)}(\alpha_k) \stackrel{?}{=} h^{(i)}(\alpha_k) - f_i(\alpha_k)$ failed:
According to P_k, processor P_i is cheating, so P_k broadcasts the index i, and $\mathcal{D}$ is set to $\{P_i, P_k\}$.

[8] Only the first occurred fault must be caused by a corrupted processor. Later faults can be aftereffects of the first fault.

(iv) The first fault is in Step 3, i.e., P_k claims that in Step 3 some processor reported a fault to him:
Since no processor admits the discovery of an inconsistency (as follows from the rule for choosing P_k), obviously either P_k is lying or the processor who reported the fault to him was malicious. P_k broadcasts the index i of the processor P_i who in Step 3 reported the fault to him, and $\mathcal{D}$ is set to $\{P_i, P_k\}$.

It is obvious that all processors find the same set $\mathcal{D}$, and that in each case at least one processor in $\mathcal{D}$ is corrupted, hence $\mathcal{D}$ is a $(1,2)$-localization.

3.9.3.5 Processor Elimination

Processor elimination is trivial. All processors set $\mathcal{P}'$ to $\mathcal{P}' \setminus \mathcal{D}$, and reduce n' to $n' - 2$ and t' to $t' - 1$.

3.9.3.6 Fault Correction

Fault correction is achieved by repeating the failed segment. Since at each failure at least one malicious processor is eliminated, there will be at most t segment-repetitions in a complete protocol run.

3.9.4 Output Stage

Let P be the designated processor supposed to receive a value s that is t-shared among the processors in $\mathcal{P}'$ with the polynomials $f(x)$ and $f_1(x), \ldots, f_{n'}(x)$. First, every processor $P_i \in \mathcal{P}'$ sends the polynomial $f_i(x)$ and the share-shares $s_{1i}, \ldots s_{n'i}$ to P. Then, P interpolates the secret s from the shares $s_i = f_i(0)$ for all i where $f_i(x)$ is consistent with all but (at most) t' share-shares s_{ij}. Note that this protocol needs neither error correction nor broadcast.

The privacy of this protocol is obvious. The correctness can be proven as follows: At most t' processors hand a bad polynomial $f_i'(x) \neq f_i(x)$, and they will be inconsistent with at least $n' - t - t' > t'$ share-shares. Hence, P will ignore bad polynomials and interpolate the correct secret s.

3.9.5 Complexity Analysis

In this section we analyze the communication complexity of the proposed multi-party computation protocol (for secure function evaluation) and compare it with the most efficient protocols known before. We focus on the case when an adversary is present and neglect the efficiency gain that some protocols achieve when no fault at all occurs. The computational complexity is not analyzed because it is only on the order of the communication complexity, which is clearly the bottleneck of such a multi-party computation.[9]

The communication complexity of a protocol is characterized by two quantities: the bit complexity (BC, the total number of bits transmitted by all processors during the protocol), and the *round complexity* (RC, the number of communication rounds of the protocol).

3.9.5.1 Broadcast Simulation

The considered multi-party computation (MPC) protocols make extensive use of a broadcast primitive and hence their efficiency depends heavily on the communication complexity of the applied broadcast subprotocol. Therefore, in a first step we will count broadcast messages separately, namely in terms of the *broadcast complexity* (BCC, the total number of bits broadcast by all processors during the protocol) and the *broadcast round complexity* (BCRC, the total number of broadcast rounds). In a second step, we determine the communication complexity of the protocol where the broadcast primitive is realized with an efficient broadcast protocol.

The broadcast protocols with optimal bit complexity, namely $\mathcal{O}(n^2)$ bits, require $\mathcal{O}(n)$ rounds of communication [BGP89, CW89, DR85, HH91]. Broadcast protocols requiring only a constant number of communication rounds are known [FM88, BPW91], but they communicate at least $\mathcal{O}(n^4)$ bits and have a non-zero probability of error. A protocol with both constant rounds and $\mathcal{O}(n^2)$ bit complexity is not known, and it is not clear whether such a protocol exists. However, the algorithm

[9] This is true unless the number of addition (or linear) gates (which require no communication) in the circuit is by orders of magnitude greater than the number of multiplication gates. It is an interesting research problem to minimize the number of multiplication gates in an arithmetic circuit for a given function, without restriction on the number of addition (or linear) gates [BPP98].

of Turpin and Coan can reduce the complexity of broadcasting long messages: For a message of k bits, this algorithm (applied to [FM88]) communicates $\mathcal{O}(kn^2 + n^5)$ bits in $\mathcal{O}(1)$ rounds. For some circuits (particularly for circuits with small depth), this algorithm improves the communication complexity of the protocol. However, for many circuits (with great depth), the algorithm even increases the complexity. In the cryptographic setting there exist a broadcast protocol with optimal bit complexity, i.e., $\mathcal{O}(n^2)$ bits, with a constant number of rounds, e.g. [CKS00]. However, this protocol requires a trusted dealer in the set-up phase. It is unclear whether such a protocol exists without additional assumptions on the model. There exist also various techniques which improve the efficiency of (stand-alone) protocols for Byzantine agreement, e.g. "early stopping" [DRS82]. However, they lead to "staggered termination" and hence are not directly applicable when such a Byzantine agreement is used as a sub-protocol of a synchronous multi-party protocol.

3.9.5.2 Complexity of the new protocol

We present a detailed analysis of the proposed protocol. Note that in the expressions describing the respective complexities only the leading terms are given while the terms of lower orders are neglected. Furthermore we assume that $n = 3t + 1$. If there are more processors (i.e., $n > 3t + 1$), one can eliminate all but $3t + 1$ of the processors before the computation starts and thereby reduce the total communication complexity of the protocol. Of course, the eliminated processors still provide input and receive output.

PROVIDING INPUT (cf. Section 3.9.2). For the distribution of polynomials $2nt$ field elements are sent. During the consistency checks every processor sends to every other processor one value from $\mathbb{F}$, yielding in total n^2 field elements. Afterwards n bits indicating the results of the checks are broadcast. If there was no cheating the protocol is finished, yielding in total BC $= 5/3n^2$ field elements, RC $= 2$, BCC $= n$ bits and BCRC $= 1$. Otherwise, if cheating occurs, the complaints and the dealer's replies require broadcasting of at most n^2 bits and nt field elements. Furthermore, during the two possible rounds of accusations (where the accusations of the second round require no reply) $2n$ bits and at most $2t^2$ field elements are broadcast. Hence, in the case with cheating BC and RC are the same as without cheating, but BCC $= n^2 + (5/9n^2) \log |\mathbb{F}|$ bits and BCRC $= 6$.

RECEIVING OUTPUT (cf. Section 3.9.4). Every processor sends to the designated processor P a polynomial of degree t and n share-shares, i.e.,

$(t+n)$ field elements, yielding in total BC $= 4/3n^2$ field elements and RC $= 1$. Broadcast is not used.

MULTIPLICATION (cf. Section 3.9.3.2). One multiplication involves three re-sharings. In each re-sharing, for every processor the following complexities arise: Re-sharing one's share plus the consistency checks: BC $= 2nt + n^2$ field elements; re-sharing the high-degree (at most $2t$) polynomial with zero-free coefficient: BC $= 4nt + n^2$ field elements. Additionally, one weak fault detection is performed: BC $= n^2$ bits. Altogether, the costs for one re-sharing are: BC $= n(6nt + 2n^2)\log|\mathbb{F}| + n^2$ bits in RC $= 3$ rounds. One multiplication involves three re-sharings, one for both inputs (in parallel) and one for the output. This sums up to: BC $= 12n^3\log|\mathbb{F}| + 3n^2$ and RC $= 6$.

SEGMENT WITH m_s MULTIPLICATIONS WITH DEPTH d_s (cf. Section 3.9.1). The computation of the m_s multiplications requires BC $= (12n^3\log|\mathbb{F}| + 3n^2)m_s$ bits in RC $= 6d_s$ rounds. At the end of the segment, during fault detection, n bits are broadcast, which yields BCC $= n$ bits in BCRC $= 1$ round.

OVERHEAD FOR FAILED SEGMENTS (cf. Section 3.9.3.4). In fault localization, every processor broadcasts the index of the multiplication gate with the first recognized inconsistency. This requires BCC $= n\log m_s$ bits. Then, at most three field elements are broadcast, and at most two processor indices. This gives the following overall complexities: BCC $= n\log m_s + 3\log|\mathbb{F}| + 2\log n$ bits in BCRC $= 3$ rounds.

FUNCTION WITH n_I INPUTS, n_O OUTPUTS, m MULTIPLICATIONS, AND DEPTH d. The protocol for providing input is performed n_I times in parallel. Let k denote the number of segments, each consisting of at most m_s multiplications with depth at most d_s. At most t segments may fail and require repetition. The protocol for computing output is performed n_O times in parallel. Since $t < n/3$ and $n < |\mathbb{F}|$, this yields in total BC $= \mathcal{O}(n_I n^2 + (m + nm_s)n^3 + n_O n^2)$ field elements, RC $= \mathcal{O}(d + nd_s)$, BCC $= \mathcal{O}(n_I n^2\log|\mathbb{F}| + (k+n)n + n^2\log m_s)$ bits, and BCRC $= \mathcal{O}(k+n)$. For $m_s = \lceil m/n\rceil$, $d_s = \lceil d/n\rceil$, and hence $k = 2n$, this gives BC $= \mathcal{O}(n_I n^2 + mn^3 + n_O n^2)$ field elements, RC $= \mathcal{O}(d+n)$, BCC $= \mathcal{O}(n_I n^2\log|\mathbb{F}| + n^2\log\frac{m}{n}))$ bits, and BCRC $= \mathcal{O}(n)$. Finally, we substitute the broadcast primitive by invocations of a broadcast protocol with optimal bit complexity [BGP89, CW89], and obtain a multiparty protocol with BC $= \mathcal{O}(n_I n^4 + mn^3 + n_O n^2)$ field elements[10] and

[10] plus $\mathcal{O}(n^4\log\frac{m}{n})$ bits, but this vanishes in mn^3 if $m \geq n$, and in $n_I n^4$ if $m < n$ and $n_I \geq 1$.

RC $= \mathcal{O}(d + n^2)$, as claimed in Theorem 8.

3.9.5.3 Comparison with other protocols

The complexity of the new protocol should be compared with the most efficient multi-party computation protocol for the unconditional model known before, namely the protocol of Beaver [Bea91a] (with [BGW88] as basic protocol), provided that the used broadcast primitive is replaced by a Byzantine agreement protocol with optimal bit complexity [BGP89, CW89]. The total bit complexity of this protocol is $\mathcal{O}(n_I n^4 + mn^6 + n_O n)$ field elements, and the total round complexity is $\mathcal{O}(d + n)$.[11]

For completeness, we also analyze the complexities of other well-known multi-party protocols. Most multi-party protocols invoke the broadcast primitive at least once in each (multiplication) gate, and hence the total round complexity of the multi-party protocol is at least d times the round complexity of the used broadcast protocol. For the sake of simplicity we assume in the sequel that the number of inputs is $n_I = \mathcal{O}(n)$, the number of outputs is $n_O = \mathcal{O}(n)$, and the number of multiplications in the circuit to be computed is at least $m = \Omega(n)$. The bit complexities (BC) are specified in field elements, not bits. A more detailed analysis of the complexities is given in Section 3.9.6.

Table 3.2 summarizes the complexities of unconditionally secure multi-party protocols. Note that the protocol of Franklin and Yung [FY92] improves the efficiency of [BGW88] only when no adversary is present. Gennaro, M. Rabin, and T. Rabin [GRR98] propose two improvements on [BGW88], an algebraic simplification and a new sub-protocol for proving that a shared value is the product of two shared factors. The latter sub-protocol is insecure (cf. [HMP00]) and hence ignored in our analysis, and the former alone does not improve the order of the complexity of the protocol. Moreover, the stated complexity of [CCD88] hides a large constant (depending polynomially on the security parameter) due to recursive application of cut-and-choose techniques.

Finally, we compare the complexity of the new protocol with the best protocol for the cryptographic model, in which up to $t < n/2$ of the processors can be corrupted, but the security of the protocol relies on unproven assumptions. The most efficient protocol in this setting is the

[11] The same bit complexity is achieved by [BGW88] with [BGP89, CW89] as broadcast protocol, but then the round complexity is $\mathcal{O}(dn)$.

MPC protocol	Broadcast protocol	BC	RC
this protocol	[BGP89, CW89]	$\mathcal{O}(mn^3)$	$\mathcal{O}(d+n^2)$
[BGW88]	[FM88]	$\mathcal{O}(mn^8)$	$\mathcal{O}(d)$
	[BGP89, CW89]	$\mathcal{O}(mn^6)$	$\mathcal{O}(dn)$
[CCD88]	[FM88]	$\mathcal{O}(mn^9)$	$\mathcal{O}(dn)$
	[BGP89, CW89]	$\mathcal{O}(mn^7)$	$\mathcal{O}(dn^2)$
[Bea91a]	[BGP89, CW89]	$\mathcal{O}(mn^6)$	$\mathcal{O}(d)$
[FY92]	[FM88]	$\mathcal{O}(mn^8)$	$\mathcal{O}(d)$
	[BGP89, CW89]	$\mathcal{O}(mn^6)$	$\mathcal{O}(dn)$
[BGW88, GRR98]	[FM88]	$\mathcal{O}(mn^8)$	$\mathcal{O}(d)$
	[BGP89, CW89]	$\mathcal{O}(mn^6)$	$\mathcal{O}(dn)$

Table 3.2: Communication complexities of unconditional MPC protocols.

protocol of [GRR98]. Also this protocol makes extensive use of a broadcast primitive (with $\mathcal{O}(n^2)$ invocations in each multiplication gate), and hence depends strongly on the efficiency of the used broadcast protocol. Using a protocol for Byzantine agreement with optimal bit complexity yields the overall complexities presented in Table 3.3. Note that the recommended broadcast protocol [CKS00] provides cryptographic security, but it assumes a trusted dealer in the set-up phase. If this is not acceptable, then another bit-optimal broadcast protocol can be used, but the round complexity of the resulting protocol will be super-linear.

MPC protocol	Broadcast protocol	BC	RC
[GRR98]	[CKS00]	$\mathcal{O}(mn^4)$	$\mathcal{O}(d)$
	[BGP89, CW89]	$\mathcal{O}(mn^4)$	$\mathcal{O}(dn)$
	[FM88]	$\mathcal{O}(mn^6)$	$\mathcal{O}(d)$

Table 3.3: Communication complexities of cryptographic MPC protocols.

3.9.6 Complexity Analysis of Known Protocols

3.9.6.1 Analysis of [BGW88]

Providing one input requires one VSS of [BGW88], hence the corresponding complexities here are the same as in our protocol. Receiving output requires in BC $= n$ field elements and RC $= 1$. For multiplication every processor shares $t+3$ values (shares of the factors, product of these shares, and t auxiliary values during the ABC-protocol). Hence for one multiplication $n(t+3)$ VSS operations are performed, yielding the following complexities: BC $= \mathcal{O}(n^4)$ field elements, RC $= \mathcal{O}(1)$, BCC $= \mathcal{O}(n^4)$ field elements and BCRC $= \mathcal{O}(1)$. When using [BGP89, CW89] as broadcast sub-protocol, this results in total complexities for m multiplication gates with depth d of BC $= \mathcal{O}(mn^6)$ field elements and RC $= \mathcal{O}(dn)$. When instead the broadcast protocol of [FM88] is used, then in total BC $= \mathcal{O}(mn^8)$ field elements and RC $= \mathcal{O}(d)$.

3.9.6.2 Analysis of [CCD88]

The protocol of [CCD88] makes extensive use of cut-and-choose techniques. For simplicity, we neglect the factors (depending on the security parameter) resulting from these techniques, and also omit the BC and RC complexities of the sub-protocols, while concentrating on the use of broadcast. A basic tool of the protocol is a double-blob, whose complexities are BCC $= \mathcal{O}(n^3)$ field elements and BCRC $= \mathcal{O}(n)$. For providing one input a *robust* double-blob is used, which requires $\mathcal{O}(n)$ double-blobs (performed possibly in parallel), hence BCC $= \mathcal{O}(n^4)$ field elements and BCRC $= \mathcal{O}(n)$. Receiving output requires BCC $= \mathcal{O}(n^2)$ field elements and BCRC $= \mathcal{O}(1)$. One multiplication gate uses $\mathcal{O}(n)$ robust double-blobs (which also may be performed in parallel), resulting BCC $= \mathcal{O}(n^5)$ field elements and BCRC $= \mathcal{O}(n)$. When using [BGP89, CW89] as broadcast sub-protocol, this results in total complexities for m multiplication gates of BC $= \mathcal{O}(mn^7)$ field elements and RC $= \mathcal{O}(dn^2)$. When instead the broadcast protocol of [FM88] is used, then in total BC $= \mathcal{O}(mn^9)$ field elements and RC $= \mathcal{O}(dn)$.

3.9.6.3 Analysis of [Bea91a]

The protocol of [Bea91a] is essentially the same as [BGW88] except for the multiplication sub-protocols being performed in the pre-computation

stage (all in parallel). In addition to [BGW88], $2m$ random values must be generated (for the circuit randomization), which costs additional $2mn$ VSS operations. This results in the following complexities for m multiplication gates: BC $= \mathcal{O}(mn^4)$ field elements, RC $= \mathcal{O}(d)$, BCC $= \mathcal{O}(mn^4)$ field elements and BCRC $= \mathcal{O}(1)$. When using [BGP89, CW89] as broadcast sub-protocol, this results in total complexities for m multiplication gates of BC $= \mathcal{O}(mn^6)$ field elements and RC $= \mathcal{O}(d+n)$. There is no advantage in using [FM88] as broadcast protocol.

3.9.6.4 Analysis of [FY92]

In the protocol of [FY92] first an efficient, fault-detecting but non-resilient protocol is performed. If an adversary is present (even if only a single processor is corrupted) and a fault is detected, the entire computation is restarted using the resilient protocol of [BGW88]. Hence the protocol of [FY92] has essentially the same complexity as the protocol of [BGW88].

3.9.6.5 Analysis of [BGW88, GRR98]

In [GRR98], two simplifications to the protocol of [BGW88] are proposed. An algebraic simplification of the multiplication sub-protocol (using Lagrange interpolation), and a new sub-protocol for proving that a shared value is the product of two shared factors. The latter was shown to be insecure (cf. [HMP00]), and the efficiency improvement of the former alone does not affect the complexity order of the resulting protocol. Hence the complexity of this protocol is equal to the complexity of [BGW88].

3.9.6.6 Analysis of [GRR98], cryptographic security

In the protocol of [GRR98], providing one input requires one VSS of [GRR98], which is clearly dominated by broadcast: BCC $= \mathcal{O}(n)$ field elements and BCRC $= \mathcal{O}(1)$. The sub-protocol for receiving output has BC $= \mathcal{O}(n)$ field elements and RC $= \mathcal{O}(1)$. For multiplication, $\mathcal{O}(n)$ VSS operations are performed, yielding the following complexities: BCC $= \mathcal{O}(n^2)$ field elements and BCRC $= \mathcal{O}(1)$. When using [CKS00] as a broadcast sub-protocol, this results in total complexities for m multiplication gates of BC $= \mathcal{O}(mn^4)$ field elements and RC $= \mathcal{O}(d)$.

3.9.6.7 Analysis of [Bea91a] with [GRR98] multiplication, cryptographic security

Per multiplication, $\mathcal{O}(n)$ VSS operations are performed, and for the preparation of the random values, $2mn$ VSS operations are necessary. Hence in total there are $\mathcal{O}(mn)$ VSS operations, which results in a total complexity of BCC $= \mathcal{O}(mn^2)$ field elements and BCRC $= \mathcal{O}(1)$. When using [BGP89, CW89] as broadcast sub-protocol, this results in total complexities for m multiplication gates of BC $= \mathcal{O}(mn^4)$ field elements and RC $= \mathcal{O}(d)$. There is no advantage in using [FM88] as broadcast protocol.

Chapter 4

General Adversaries in MPC

For many applications, threshold-security as provided by classical multi-party protocols is not sufficient. More generally, security is defined with respect to an *adversary structure*, a set of adversary classes, where each class specifies which processors may be corrupted in which mode. We consider the passive model, where an adversary class is specified by a subset of the processor set, indicating which processors may be passively corrupted, the active model, where still a class is a subset of the processors set, but this set may be actively corrupted, and the mixed model, where a class is a pair of subsets of the processor set, the first subset indicating which processors may be actively corrupted, and the second indicating which processors may be passively corrupted. Fail-corruption is not considered in this chapter.

The main technique to construct multi-party protocols secure against a general adversary is *processor simulation*: We start with a multi-party protocol among three or four *virtual processors*, with security against an adversary corrupting one of these processors. Then, each of these virtual processors is simulated by a multi-party protocol among a set of new (virtual) processors, and so on. We formally define what it means to simulate a processor by a multi-party protocol among a set of (new) processors, and we derive the tolerated adversary structure of the new protocol as a function of the adversary structures of the original protocol and the protocols used for the simulation. This technique of simulating

processors will be of central importance when constructing protocols for general adversary structures. It turns out that it is sufficient to consider univariate (i.e., passive or active, but not mixed) adversary structures. The main result about processor simulation, which will be rigorously proven in Section 4.1, is the following: Consider a multi-party computation protocol, in which some of the (virtual) processors are simulated using other multi-party computations. Then, the resulting protocol tolerates a specific adversary if every corrupted non-simulated processor is tolerated in the original protocol and, in addition, for every simulated processor, either the adversary is tolerated in the corresponding simulation sub-protocol, or this processor is tolerated to be (additionally) corrupted in the original protocol. The intuition behind this is that if the adversary is tolerated in the simulation sub-protocol, then the simulated virtual processor behaves honestly, and follows the protocol. If, however, the adversary is not tolerated in the simulation sub-protocol, then the simulated processor might misbehave, and this must be tolerated in the main protocol.

For constructing protocols secure against general adversaries, we will not use the full power of processor simulation. The simulation trees will be simple and "clean". But we think that processor simulation is a generic tool that might be useful for other purposes as well, and therefore introduce the general simulation techniques and prove security for arbitrary processor simulation hierarchies.

With these foundations, we can combine given protocol generators and recursively construct new multi-party computation protocols. By clever combining processor-simulation trees, protocols for any admissible adversary structure can be constructed.

We first focus on the passive model, and then on the active model, once without assuming broadcast channels, and once with assuming broadcast channels. We then generalize these results by considering the mixed model with simultaneous active and passive corruptions, once with perfect security (with or without broadcast channels), once with unconditional security with assuming broadcast channels, and finally, with unconditional security without assuming broadcast.

In order to state the main results on the feasibility of multi-party computation in the considered models, we need to define some predicates on adversary structures. We first define two predicates on univariate struc-

tures:

$$Q^{(2)}(\mathcal{P}, \mathcal{Z}) \iff \forall Z_1, Z_2 \in \mathcal{Z} : Z_1 \cup Z_2 \neq \mathcal{P},$$
$$Q^{(3)}(\mathcal{P}, \mathcal{Z}) \iff \forall Z_1, Z_2, Z_3 \in \mathcal{Z} : Z_1 \cup Z_2 \cup Z_3 \neq \mathcal{P}.$$

Furthermore, we define three predicates on "bivariate" structures for the mixed model:

$$Q^{(2,2)}(\mathcal{P}, \mathcal{Z}) \iff \forall (D_1, E_1), (D_2, E_2) \in \mathcal{Z} : D_1 \cup E_1 \cup D_2 \cup E_2 \neq \mathcal{P},$$
$$Q^{(3,2)}(\mathcal{P}, \mathcal{Z}) \iff \forall (D_1, E_1), (D_2, E_2), (D_3, E_3) \in \mathcal{Z} : D_1 \cup E_1 \cup D_2 \cup E_2 \cup D_3 \neq \mathcal{P},$$
$$Q^{(3,0)}(\mathcal{P}, \mathcal{Z}) \iff \forall (D_1, E_2), (D_2, E_2), (D_3, E_3) \in \mathcal{Z} : D_1 \cup D_2 \cup D_3 \neq \mathcal{P}.$$

In the passive model, a passive adversary structure can be tolerated if and only if no two sets in the structure cover the full processor set (i.e., if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This condition corresponds to (and strictly generalizes) the necessary condition for the passive threshold model, namely that $t_p < n/2$. Formally stating and proving this result is the goal of Section 4.2.

Analogously, in the active model, an active adversary structure can be tolerated if no three sets cover the full processor set (i.e., if $Q^{(3)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This condition immediately corresponds to (and strictly generalizes) the condition in the active threshold model, namely that $t_a < n/3$. We will state and prove this result in Section 4.3.

In the active model with broadcast, the necessary and sufficient condition for unconditionally secure multi-party computation is that no two sets in the adversary structure cover the full processor set (i.e., if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This condition corresponds to (and strictly generalizes) the condition in the active threshold model, namely that $t_a < n/2$. This model is considered in Section 4.4.

In the mixed model, where passive and active corruption occur simultaneously, we consider bivariate adversary structures. The necessary and sufficient condition for perfectly secure multi-party computation to exist for every specification is that the union of two adversary classes and the active set of another class in the adversary structure do not cover the full processor set, i.e., $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. Also this condition strictly generalizes the condition for the threshold mixed model with active and

passive corruptions, namely that $3t_a + 2t_p < n$. We will prove this result in Section 4.5.

When given broadcast channels, and accepting a negligible error probability, then the above condition can be weakened: It is sufficient (and necessary) that no two adversary classes cover the full processor set, i.e., $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ must be satisfied. Also this condition strictly generalizes the corresponding condition for the threshold mixed model, namely that $2t_a + 2t_p < n$. We consider this model and prove the necessity and sufficiency of the condition in Section 4.6

Finally, if no broadcast channels are available, but still some error probability is tolerated, then the additional requirement that the active sets of any three classes in the adversary structure do not cover the processor set (i.e., $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$), is required for simulating the broadcast channels. This corresponds to the additional condition that $3t_a < n$ in the mixed model without broadcast. This model will be discussed in Section 4.7.

The efficiency of the proposed protocols is polynomial in the size of the basis of the adversary structure to be tolerated. It is an open problem to find other general descriptions of structures for which polynomial (in the number of players) protocols can be found (for a possible approach and some new results see [CDM00]). A further open problem is to give general conditions on adversary structures such that polynomial protocols exist. However, the number of maximal bases of structures satisfying the $Q^{(2)}$ or the $Q^{(3)}$ condition are doubly-exponential in the number of processors, and therefore a construction of polynomial protocols can be found at most for some particular classes of structures (cf. Section 4.8).

4.1 Processor Simulation

The goal of secure multi-party computation is to transform a given protocol involving a trusted party into a protocol without need for the trusted party, by *simulating* the party among the processors. Indeed, by the same means, one can simulate an arbitrary processor in any given protocol.[12] We formally define what it means to simulate a processor by a multi-party protocol among a set of (new) processors, and we derive the resilience of the new protocol as a function of the resiliences of the origi-

[12]The idea of simulating a single processor by a subprotocol was used in [Cha89] for a different purpose.

nal protocol and the protocol used for the simulation. This technique of simulating processors will be of central importance when constructing protocols for general adversary structures.

In this section, we only consider a model with a single corruption type, either passive or active. Hence, an adversary structure $\mathcal{Z}$ is a set of subsets of the processor set, i.e., $\mathcal{Z} \subseteq 2^{\mathcal{P}}$.

As an example of processor simulation, consider the set $\mathcal{P} = \{P_1, \ldots, P_6\}$ of processors, and the 4-party protocol of [BGW88] (for the active case) in which P_1 and P_3 play for one processor each and the other two processors are simulated by 4-party protocols of the same type, one among the processors P_1, P_2, P_3, and P_4, and the other among the processors P_1, P_2, P_5, and P_6 (see Figure 4.1).

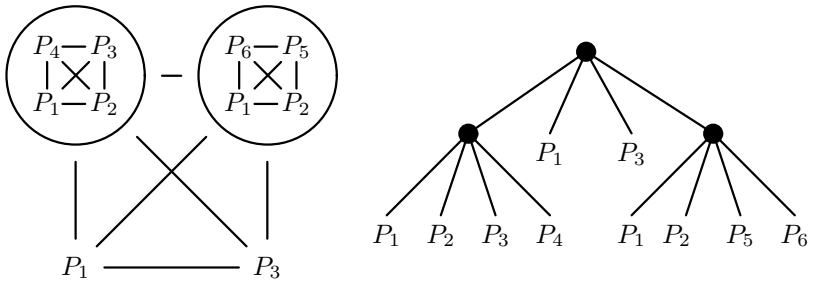


Figure 4.1: Example of a processor simulation.

This protocol tolerates the following adversary structure:

$$\mathcal{Z} = \left\langle \left\{ \{P_1\}, \{P_2, P_4\}, \{P_3, P_5\}, \{P_3, P_6\}, \{P_2, P_5, P_6\}, \{P_4, P_5, P_6\} \right\} \right\rangle .$$

For each set in $\mathcal{Z}$, one can easily verify that the set is tolerated: for example, the set $\{P_2, P_5, P_6\}$ is tolerated because only one processor is corrupted in the simulating protocol among P_1, P_2, P_3, and P_4 (thus this protocol simulates an honest processor for the main protocol), and hence three of the four processors in the main protocol play honestly. The fact that there are too many corrupted processors in the subprotocol among P_1, P_2, P_5, and P_6 does not matter.

The tolerated sets can easily be derived by representing the simulation hierarchy as a tree (see right hand side of Figure 4.1). For a specific adversary, to every leaf the value 1 is assigned if the corresponding processor is non-corrupted, and 0 is assigned if the corresponding processor

is corrupted. To every inner node, 1 is assigned if and only if more than $2/3$ of its children have 1 associated (more than $1/2$ in the passive model). The considered adversary is tolerated exactly if this procedure assigns 1 to the root node. More formally, the tree corresponds to a circuit with threshold gates, and an adversary is tolerated exactly if the corresponding input vector evaluates to 1.

In this example, we have considered a particular simulation tree, and derived the tolerated adversary structure. Deriving and rigorously proving the tolerated adversary structure of a simulation is the major goal of this section. In a first step (Section 4.1.2), virtual processors are simply *renamed* using a *processor mapping*, i.e., one (virtual) processor plays for one or several virtual processors. In a second step (Section 4.1.3), virtual processors are *simulated* by a set of (virtual) processors, i.e., the simulating processors perform all operations of the simulated virtual processor by a multi-party computation.

4.1.1 Definitions

Let $\mathcal{P}$ and $\mathcal{P}'$ be sets of processors. A *processor mapping* σ,

$$\sigma : \mathcal{P} \to \mathcal{P}' ,$$

is a surjective function from $\mathcal{P}$ onto $\mathcal{P}'$.[13] The definition of a processor mapping σ is extended to the following domains: For a protocol π, the mapped protocol $\sigma(\pi)$ (or, equivalently, $\sigma\pi$) is the same protocol, where in each statement all involved processors are replaced by the corresponding mapped processors (if processors that are not in $\mathcal{P}$ are involved in a statement, then these processors are not replaced). For a specification (π_0, τ) with $\tau \notin \mathcal{P}$, the mapped specification is the specification with the mapped protocol, i.e., $\sigma(\pi_0, \tau) = (\sigma\pi_0, \tau)$. Note that $\sigma(\pi_0, \tau)$ stands for $\sigma\big((\pi_0, \tau)\big)$.

The *inverse processor mapping* σ^{-1} of a processor mapping σ is defined by

$$\sigma^{-1} : \ \mathcal{P}' \to 2^{\mathcal{P}}, \ P' \mapsto \{P \in \mathcal{P} \colon \sigma(P) = P'\} \ .$$

[13] In the application of processor mappings, parentheses may be omitted whenever the precedence rules allow it. As usual, function application (in particular a processor mapping) is right-associative and has higher precedence than any two-adic operator. For any two processor mappings σ_1 and σ_2, for an arbitrary two-adic operator $\diamond$, and for any x_1 and x_2 we have $\sigma_1\sigma_2 x_1 \diamond x_2 = \sigma_1(\sigma_2(x_1)) \diamond x_2$.

If the processor mapping σ is bijective, then the function value of the inverse processor mapping σ^{-1} is sometimes interpreted as a single processor (instead of a set that contains a single processor).[14] Also, we define the mapping of a set of processors to be the set of the mapped processors, and the inverse mapping of a set B of processors to be the union of the sets of the inverse mappings applied to the processors in B (i.e., $\sigma(B) = \bigcup_{P \in B} \{\sigma(P)\}$ and $\sigma^{-1}(B) = \bigcup_{P \in B} \sigma^{-1}(P)$).

In the following, we give definitions for applying processor mappings to adversary structures and to protocol generators. These definitions are appropriate in the sense that if a protocol (generator) tolerates an adversary structure, then the mapped protocol (generator) will tolerate the mapped adversary structure. This will be proven in the next section.

For a structure $\mathcal{Z}$ for the set $\mathcal{P}$ of processors and a processor mapping $\sigma : \mathcal{P} \to \mathcal{P}'$, the mapped structure is

$$\sigma(\mathcal{Z}) = \left\{ Z \subseteq \mathcal{P}' : \sigma^{-1}(Z) \in \mathcal{Z} \right\} ,$$

i.e., a set Z is in $\sigma(\mathcal{Z})$ if the set of all processors mapped to a processor in Z is in $\mathcal{Z}$.

For a protocol generator G for the set $\mathcal{P}$ of processors and a processor mapping $\sigma : \mathcal{P} \to \mathcal{P}'$, the mapped protocol generator $\sigma(G)$ is a protocol generator that, applied to a specification (π_0, τ), simulates the trusted party τ by the processors in $\mathcal{P}'$ (instead of $\mathcal{P}$). In order to prevent syntactical collisions with the names of the processors in $\mathcal{P}$ and of those appearing in π_0, we first rename the processors appearing in π_0 to some new processor names[15], then apply the original protocol generator G, then apply the processor mapping σ, and finally rename the previously renamed processors back to their original names. More formally, when given (π_0, τ) where π_0 involves the set $\mathcal{P}_0$ of processors, $\sigma(G)$ first applies an arbitrary bijective processor mapping [16] $\rho : (\mathcal{P}_0 \setminus \tau) \to \overline{\mathcal{P}}$, where $\overline{\mathcal{P}}$ is a set of new processor names, to the specification, then applies the protocol generator G to this modified protocol specification, further applies

[14] Generally, the inverse of a processor mapping is not a processor mapping. However, the inverse of a *bijective* processor mapping can be considered (and will be considered) as a processor mapping.

[15] That is, processor names that did not yet appear anywhere, neither in the protocol nor in the protocol generator nor in the mapping.

[16] This corresponds to alpha renaming in the context of lambda calculus and is a purely technical step. Note that the name of the trusted party must not be mapped.

the original processor mapping σ and finally applies the inverse processor mapping ρ^{-1} to the resulting protocol. Formally,

$$\sigma G = \sigma(G) = \Big((\pi_0, \tau) \mapsto \rho^{-1}\big(\sigma(G(\rho\pi_0, \tau))\big) \Big)$$

for an appropriate bijective processor mapping ρ. Note that σG does not depend on the choice of ρ.

Consider a multi-party protocol π among the set $\mathcal{P}$ of processors and a protocol generator G for the set $\mathcal{P}_G$ of processors. To *simulate* a virtual processor $P \in \mathcal{P}$ in π applying the protocol generator G means to consider this processor P as a trusted party and to have this party simulated by a subprotocol among the processors in $\mathcal{P}_G$, according to G. More precisely, the specification (π, P) is used as input for the protocol generator G. To *simultaneously simulate* the processors $P_{r_1}, \ldots, P_{r_k} \in \mathcal{P}$ in π using the protocol generators $G_1, \ldots, G_k$ for the processor sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$, respectively, is defined as follows: First consider k arbitrary bijective processor mappings $\sigma_i : \mathcal{P}_i \to \overline{\mathcal{P}}_i$ (for $i = 1, \ldots, k$), where $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_k$ are pairwise disjoint sets of new processor names. Then, the resulting protocol is

$$\sigma_k^{-1} \cdots \sigma_1^{-1} \Big((\sigma_k G_k) \big(\cdots (\sigma_2 G_2) \big((\sigma_1 G_1)(\pi, P_{r_1}), P_{r_2} \big) \cdots, P_{r_k} \big) \Big) ,$$

and does not depend on the choices for $\sigma_1, \ldots, \sigma_k$, nor on the order of the protocol generators $G_1, \ldots, G_k$.

4.1.2 Renaming Processors

It is trivial that by renaming processors in a protocol, the tolerated adversary structure is the same with the identically renamed processors. More precisely, security of a protocol is defined with respect to a specification, and the security of the renamed protocol is with respect with the renamed specification. Furthermore, when several processors are renamed to the same processor P (i.e., P "plays" the role of several processors), then a subset Z of the processors that contains P is tolerated in the renamed protocol if and only if the set of all the renamed processors and all the processors in $Z \backslash \{P\}$ is tolerated in the original protocol. A subset Z with $P \notin Z$ is tolerated if Z is tolerated in the original protocol. This naturally extends the "partition lemma" of [CK89].

Lemma 1 *Given a protocol π for the set $\mathcal{P}$ of processors that $\mathcal{Z}$-securely computes the specification (π_0, τ), and some processor mapping σ, then $\sigma(\pi)$ is a protocol for the set $\sigma(\mathcal{P})$ of processors that $\sigma(\mathcal{Z})$-securely computes the specification $\sigma(\pi_0, \tau)$.*

Proof: We have to show that for every adversary A' for the mapped protocol $\sigma(\pi)$, with $Z_{A'} \in \sigma(\mathcal{Z})$, the protocol $\sigma(\pi)$ A'-securely computes the mapped specification $\sigma(\pi_0, \tau)$. Figure 4.2 illustrates the procedure for constructing an ideal adversary A'_0 for the mapped specification $\sigma(\pi_0, \tau)$ from a given adversary A' for $\sigma\pi$. We begin with the adversary A' for the mapped protocol $\sigma\pi$, construct an adversary A for the original protocol π, and show that A is tolerated in the protocol π. Thus, by the definition of security of a protocol, there exists an ideal adversary A_0 for the protocol π_0 of the specification. Then we use A_0 to construct an adversary A'_0 for the mapped specification $\sigma(\pi_0, \tau)$, and we prove that this adversary is an ideal adversary of the original adversary A'.

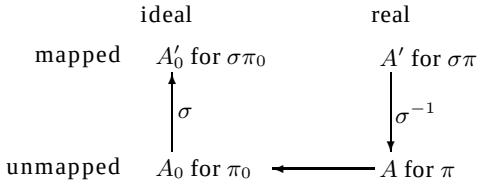


Figure 4.2: Construction of the ideal adversary A'_0 for a given adversary A'.

Consider an arbitrary adversary A' for $\sigma\pi$ with $Z_{A'} \in \sigma(\mathcal{Z})$. We define A to be the adversary for π with $Z_A = \sigma^{-1}(Z_{A'})$ and with the same strategy as A', except that whenever A' reads from, or writes to, the tape of a corrupted processor $P' \in Z_{A'}$, then A accesses the tape of the corresponding[17] processor $P \in \sigma^{-1}(P')$ in the same manner as A'. By the definition of processor mappings for structures, $Z_{A'} \in \sigma(\mathcal{Z})$ implies that $Z_A \in \mathcal{Z}$. Hence, by what it means for a protocol to be A-secure, there exists a statement index function $f_\pi : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\}$ and an adversary A_0 for π_0 with $Z_{A_0} = Z_A\big|_{\mathcal{P}_0 \setminus \{\tau\}}$ (where $\mathcal{P}_0$ is the set of processors of the ideal protocol π_0) such that for every $i = 1, \ldots, |\pi_0| + 1$ the joint distribution of the view of the adversary A_0 and the views $\nu_i(P)$

[17] If the mapping is not bijective, then for constructing the adversary A one must consult the unmapped protocol π to determine which processor's tape needs to be accessed.

of all non-corrupted processors $P \in \big(\mathcal{P}_0 \setminus \{\tau\} \setminus Z_{A_0}\big)$ before the i-th statement of the ideal protocol π_0 (with the adversary A_0 present) is equal to the joint distribution of the view of the adversary A and the views $\nu_i(P)$ of all non-corrupted processors $P \in \big(\mathcal{P}_0 \setminus \{\tau\} \setminus Z_{A_0}\big)$ before the $f_\pi(i)$-th statement of the real protocol π (with the adversary A present). Let the statement index function $f_{\sigma\pi}$ for the mapped protocol be equal to that of the unmapped protocol, i.e., $f_{\sigma\pi} = f_\pi$, and let A'_0 be the adversary for the mapped ideal protocol $\sigma\pi_0$ with $Z_{A'_0} = \sigma(Z_{A_0}) = \sigma\big(Z_A\big|_{\mathcal{P}_0\setminus\{\tau\}}\big) = \sigma\big(\sigma^{-1}(Z_{A'})\big|_{\mathcal{P}_0\setminus\{\tau\}}\big) = Z_{A'}\big|_{\sigma(\mathcal{P}_0)\setminus\{\tau\}}$. The strategy of A'_0 is the same as the strategy of the adversary A_0, except that whenever A_0 reads from, or writes to, the tape of a corrupted processor $P \in Z_{A_0}$, then A'_0 accesses the tape of the processor $\sigma(P)$.

Clearly, for each $i = 1, \ldots, |\sigma\pi_0| + 1$, the joint distribution of the view of the adversary A'_0 and the views $\nu_i(P)$ of all non-corrupted processors $P \in \big(\sigma(\mathcal{P}_0)\setminus\{\tau\}\setminus Z_{A'_0}\big)$ before the i-th statement of the mapped ideal protocol $\sigma\pi_0$ (with the adversary A'_0 present) is equal to the joint distribution of the view of the adversary A' and the views $\nu_{f_{\sigma\pi}(i)}(P)$ of all non-corrupted processors $P \in \big(\sigma(\mathcal{P}_0) \setminus \{\tau\} \setminus Z_{A'_0}\big)$ before the $f_{\sigma\pi}(i)$-th statement of the mapped real protocol $\sigma\pi$ (with the adversary A' present). ■

If the structure $\mathcal{Z}$ is maximal for the protocol π (i.e., for every $Z \subseteq \mathcal{P}$ with $Z \notin \mathcal{Z}$ there exists an adversary A with $Z_A \subseteq Z$, such that π does not A-securely compute (π_0, τ)), then $\sigma(\mathcal{Z})$ is also maximal for $\sigma(\pi)$.

The corollary below follows immediately from Lemma 1 and from the definition of processor mappings for protocol generators.

Corollary 1 *Given a $\mathcal{Z}$-secure protocol generator G for the set $\mathcal{P}_G$ of processors, and given some processor mapping σ, then $\sigma(G)$ is a $\sigma(\mathcal{Z})$-secure protocol generator for the set $\sigma(\mathcal{P}_G)$ of processors.*

4.1.3 Simulating a Single Processor

Consider a protocol π for the set $\mathcal{P}$ of processors, a processor $P \in \mathcal{P}$, and a protocol generator G' for the set $\mathcal{P}'$ of processors, and assume that π is $\mathcal{Z}$-secure for an adversary structure $\mathcal{Z}$ and G' is $\mathcal{Z}'$-secure for an appropriate adversary structure $\mathcal{Z}'$. Let P be simulated in π by G', and let π^* denote the resulting protocol. A set Z of processors is tolerated in

π^* if the set Z is tolerated in π (i.e., $Z\big|_{\mathcal{P}} \in \mathcal{Z}$) and Z is tolerated by G' (i.e $Z\big|_{\mathcal{P}'} \in \mathcal{Z}'$). Even if Z is not tolerated by G', but instead π tolerates that P is corrupted in addition to the processors in Z (i.e., $\left(Z\big|_{\mathcal{P}} \cup \{P\}\right) \in \mathcal{Z}$), then Z is nevertheless tolerated in the resulting protocol π^*. This is formally stated and proved below.

Lemma 2 *Consider a specification (π_0, τ) for the set $\mathcal{P}_0$ of processors, a protocol π for the set $\mathcal{P}$ of processors that $\mathcal{Z}$-securely computes (π_0, τ) for some adversary structure $\mathcal{Z} \subseteq 2^{\mathcal{P}}$, and a natural protocol generator G' for the set $\mathcal{P}'$ of processors (where $\mathcal{P}' \cap \mathcal{P} = \emptyset$) that is $\mathcal{Z}'$-secure for some adversary structure $\mathcal{Z}' \subseteq 2^{\mathcal{P}'}$. Simulating a processor $P \in \left(\mathcal{P} \setminus \{\tau\} \setminus \mathcal{P}_0\right)$ in π by applying the protocol generator G' results in a protocol π^* for the set $\mathcal{P}^*$ of processors that $\mathcal{Z}^*$-securely computes the specification (π_0, τ) where*

$$\mathcal{P}^* = \left(\mathcal{P} \setminus \{P\}\right) \cup \mathcal{P}'$$
$$\mathcal{Z}^* = \left\{ Z \subseteq \mathcal{P}^* \colon \left(\left(Z\big|_{\mathcal{P}} \cup \{P\}\right) \in \mathcal{Z}\right) \vee \left(Z\big|_{\mathcal{P}} \in \mathcal{Z} \wedge Z\big|_{\mathcal{P}'} \in \mathcal{Z}'\right) \right\}.$$

Proof: Consider an arbitrary adversary A^* for the protocol π^* with $Z_{A^*} \in \mathcal{Z}^*$, and a statement index function $f' : \{1, \ldots, |\pi|+1\} \to \{1, \ldots, |\pi^*|+1\}$ for G'. We construct a statement index function $f^* : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi^*| + 1\}$ and an ideal adversary A_0 of the adversary A^* where $Z_{A_0} = Z_{A^*}\big|_{\mathcal{P}_0 \setminus \{\tau\}}$. We distinguish between two cases: In the first case, we assume that $\left(Z_{A^*}\big|_{\mathcal{P}} \cup \{P\}\right) \in \mathcal{Z}$, and in the second case, we assume that $Z_{A^*}\big|_{\mathcal{P}} \in \mathcal{Z} \wedge Z_{A^*}\big|_{\mathcal{P}'} \in \mathcal{Z}'$.

First, assume that $\left(Z_{A^*}\big|_{\mathcal{P}} \cup \{P\}\right) \in \mathcal{Z}$, i.e., the adversary may corrupt the simulated processor in the main protocol. We define A to be the adversary for the protocol π with $Z_A = Z_{A^*}\big|_{\mathcal{P}} \cup \{P\}$ with the following strategy: Without loss of generality, let $\mathcal{P}' = \{P_1, \ldots, P_m\}$. First, A locally initializes $|\mathcal{P}'| + 1$ virtual processors $\widetilde{P}_0, \widetilde{P}_1, \ldots, \widetilde{P}_m$. These virtual processors will be used to simulate the statements that involve processors in $\mathcal{P}'$. More precisely, $\widetilde{P}_0$ will be used to simulate the behavior of the uncorrupted sender / receiver in a *transmit*-statement to / from P (the real sender / receiver just transmits a value to / from P, according to the statement in π, so A has to simulate the corresponding statement sequence in π^* with $\widetilde{P}_0$ as sender / receiver), and the other virtual processors will be used to simulate the (in π not existing) processors in P'. Note that here we must assume that the protocol generator is natural, i.e., a *transmit*-statement only involves the value to be transmitted (plus maybe some

auxiliary values). For example, when in π a value is sent from an uncorrupted processor to P, then P (resp. A) receives this value, and simulates the corresponding statement sequence in π^*, where as sender, $\widetilde{P}_0$ is used, and the receiver is simulated among $\widetilde{P}_1, \ldots, \widetilde{P}_m$. Hence, it is necessary that the statement sequence in π^* only involves the value that the uncorrupted processor sends to P, such that A can perform the necessary simulation of π^*.

For every $i = 1, \ldots, |\pi| + 1$, A performs the following steps for the i-th statement d_i of π:

- If the statement d_i does not involve P, A performs the same steps that A^* would perform.
- If d_i is a *comp*-statement for P, then A executes the sequence $f'(i), \ldots, \big(f'(i+1) - 1\big)$ of statements of π^*, where each processor $P_k \in \mathcal{P}'$ is relabeled to the virtual processor $\widetilde{P}_k$.[18] After each statement of this sequence, A performs the same steps that A^* would perform (modified such that it accesses the tapes of $\widetilde{P}_k$ instead of P_k).
- If d_i is a *transmit*(P, P_j, x)-statement, then A executes the sequence $f'(i), \ldots, f'(i+1) - 1$ of statements of π^*, where each processor $P_k \in \mathcal{P}'$ is relabeled to the virtual processor $\widetilde{P}_k$, and the receiving processor P_j is relabeled to $\widetilde{P}_0$. After each statement of this sequence, A performs the same steps that A^* would perform (where A accesses the tapes of $\widetilde{P}_k$ instead of P_k, and of $\widetilde{P}_0$ instead of P_j). At the end of the sequence, A reads the value of the variable x in the view of $\widetilde{P}_0$ and sends this value to P_j. If the adversary A^* is passive (and hence also A is passive), then this value corresponds to the value that P would send to P_j.
- If d_i is a *transmit*(P_j, P, x)-statement, then A first reads the value of x and puts this value into the view of $\widetilde{P}_0$, then executes the sequence $f'(i), \ldots, f'(i+1) - 1$ of statements of π^*, where each processor $P_k \in \mathcal{P}'$ is first relabeled to the virtual processor $\widetilde{P}_k$, and the sending processor P_j is renamed to $\widetilde{P}_0$. After each statement of this sequence, A performs the same steps that A^* would perform (modified as above).

[18] More precisely, a processor mapping σ that maps $P_k \mapsto \widetilde{P}_k$ $\big(1 \leq k \leq m\big)$ is applied to the sequence of statements.

The described adversary A is tolerated in π because $Z_A \in \mathcal{Z}$. Thus there exists an ideal adversary A_0 of A with $Z_{A_0} = Z_A\big|_{\mathcal{P}_0\setminus\{\tau\}}$. Clearly, this is also an ideal adversary for A'.

Second, assume that $Z_{A^*}\big|_{\mathcal{P}} \in \mathcal{Z} \wedge Z_{A^*}\big|_{\mathcal{P}'} \in \mathcal{Z}'$. A^* is tolerated by the protocol generator G' (because $Z_{A^*}\big|_{\mathcal{P}'} \in \mathcal{Z}'$); thus by considering (π, P) as the specification of π^*, there exists an ideal adversary A of A^* for the protocol π with $Z_A = Z_{A^*}\big|_{\mathcal{P}}$. Because $Z_A \in \mathcal{Z}$ there exists an ideal adversary A_0 of A for the ideal protocol π_0 with $Z_{A_0} = Z_A\big|_{\mathcal{P}_0\setminus\{\tau\}}$, and this A_0 is also an ideal adversary of A^*. ■

4.1.4 General Simulation of Processors

In this section, we consider the simultaneous simulation of several processors with completely general (possibly overlapping) sets of simulating processors. The goal of this section is to prove the following theorem:

Theorem 9 *Let π be a protocol among the set $\mathcal{P}$ of processors that $\mathcal{Z}$-securely compute a specification (π_0, τ), and let $G_1, \ldots, G_k$ be $\mathcal{Z}_1$-, $\ldots$,$\mathcal{Z}_k$-secure natural protocol generators for the processor sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$, respectively. Assume that in π the k virtual processors $P_{r_1}, \ldots, P_{r_k} \in \mathcal{P}$ are simultaneously simulated by subprotocols applying the protocol generators $G_1, \ldots, G_k$, respectively. Then the resulting multi-party protocol π^* is for the set $\mathcal{P}^*$ of processors and $\mathcal{Z}^*$-securely computes the specification (π_0, τ), where*

$$\mathcal{P}^* = \big(\mathcal{P}\setminus R\big) \cup \bigcup_{i=1}^{k} \mathcal{P}_i \text{ , and}$$

$$\mathcal{Z}^* = \left\{ Z \subseteq \mathcal{P}^* \colon \left(Z\big|_{\mathcal{P}\setminus R} \cup \left\{ P_{r_i} \in R \colon Z\big|_{\mathcal{P}_i} \notin \mathcal{Z}_i \right\} \right) \in \mathcal{Z} \right\},$$

and $R = \{P_{r_1}, \ldots, P_{r_k}\}$ is the set of replaced processors.

Proof: According to the definition of simultaneous simulation,

$$\pi^* = \sigma_k^{-1} \cdots \sigma_1^{-1} \Big(\big(\sigma_k G_k\big)\big(\cdots(\sigma_2 G_2)((\sigma_1 G_1)(\pi, P_{r_1}), P_{r_2}) \cdots, P_{r_k}\big) \Big)$$

for some bijective processor mappings $\sigma_1 : \mathcal{P}_1 \to \overline{\mathcal{P}}_1, \ldots, \sigma_k : \mathcal{P}_k \to \overline{\mathcal{P}}_k$, where $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_k$ are pairwise disjoint sets of new processor names. According to Corollary 1, $\sigma_1 G_1, \ldots, \sigma_k G_k$ are natural protocol generators

for the sets $\sigma_1\mathcal{P}_1, \ldots, \sigma_k\mathcal{P}_k$ of processors that are $\sigma_1\mathcal{Z}_1, \ldots, \sigma_k\mathcal{Z}_k$ secure, respectively.

These protocol generators are applied subsequently, where after applying the i-th generator the set of processors is denoted by $\mathcal{P}^{(i)}$ and the tolerated adversary structure is denoted by $\mathcal{Z}^{(i)}$. In the following, some technical transformations of $\mathcal{Z}^{(i)}$ may at first glance appear to be unmotivated.

Applying Lemma 2, $(\sigma_1 G_1)(\pi, P_{r_1})$ is a protocol for the set $\mathcal{P}^{(1)}$ of processors tolerating $\mathcal{Z}^{(1)}$, where

$$\begin{aligned}
\mathcal{P}^{(1)} &= \big(\mathcal{P} \backslash \{P_{r_1}\}\big) \cup \sigma_1\mathcal{P}_1 \\
\mathcal{Z}^{(1)} &= \left\{ Z \subseteq \mathcal{P}^{(1)} \colon \begin{array}{l} \Big(\big(Z\big|_{\mathcal{P}} \cup \{P_{r_1}\}\big) \in \mathcal{Z}\Big) \ \bigvee \\ \Big(Z\big|_{\mathcal{P}} \in \mathcal{Z} \wedge Z\big|_{\sigma_1\mathcal{P}_1} \in \sigma_1\mathcal{Z}_1\Big) \end{array} \right\} \\
&= \left\{ Z \subseteq \mathcal{P}^{(1)} \colon \left(Z\big|_{\mathcal{P}} \cup \left\{P_{r_i} \in \{P_{r_1}\} \colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\right\}\right) \in \mathcal{Z}\right\} .
\end{aligned} \tag{1}$$

Furthermore, $\big(\sigma_2 G_2\big)\big((\sigma_1 G_1)(\pi, P_{r_1}), P_{r_2}\big)$ is a protocol for the set $\mathcal{P}^{(2)}$ of processors tolerating $\mathcal{Z}^{(2)}$, where

$$\begin{aligned}
\mathcal{P}^{(2)} &= \big(\mathcal{P}^{(1)} \backslash \{P_{r_2}\}\big) \cup \sigma_2\mathcal{P}_2 = \big(\mathcal{P} \backslash \{P_{r_1} \cup P_{r_2}\}\big) \cup \sigma_1\mathcal{P}_1 \cup \sigma_2\mathcal{P}_2 \\
\mathcal{Z}^{(2)} &= \left\{ Z \subseteq \mathcal{P}^{(2)} \colon \begin{array}{l} \Big(\big(Z\big|_{\mathcal{P}^{(1)}} \cup \{P_{r_2}\}\big) \in \mathcal{Z}^{(1)}\Big) \ \bigvee \\ \Big(Z\big|_{\mathcal{P}^{(1)}} \in \mathcal{Z}^{(1)} \wedge Z\big|_{\sigma_2\mathcal{P}_2} \in \sigma_2\mathcal{Z}_2\Big) \end{array} \right\} \\
&= \Big\{ Z \subseteq \mathcal{P}^{(2)} \colon \underbrace{\Big(Z\big|_{\mathcal{P}^{(1)}} \cup \Big\{P_{r_i} \in \{P_{r_2}\} \colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big)}_{T} \in \mathcal{Z}^{(1)} \Big\} .
\end{aligned}$$

We now replace $\mathcal{Z}^{(1)}$ in the above equation by using (1). Let T be the under-braced term.

$$\mathcal{Z}^{(2)} = \left\{ Z \subseteq \mathcal{P}^{(2)} \colon \left(T\big|_{\mathcal{P}} \ \cup \left\{P_{r_i} \in \{P_{r_1}\} \colon T\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\right\}\right) \in \mathcal{Z}\right\} .$$

We have

$$\begin{aligned}
T\big|_{\mathcal{P}} &= \Big(Z\big|_{\mathcal{P}^{(1)}} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big)\Big|_{\mathcal{P}} \\
&= \big(Z\big|_{\mathcal{P}^{(1)}}\big)\big|_{\mathcal{P}} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big|_{\mathcal{P}} \\
&= Z\big|_{\mathcal{P}} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}
\end{aligned}$$

and

$$\begin{aligned}
T\big|_{\sigma_i\mathcal{P}_i} &= \Big(Z\big|_{\mathcal{P}^{(1)}} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big)\Big|_{\sigma_i\mathcal{P}_i} \\
&= \big(Z\big|_{\mathcal{P}^{(1)}}\big)\big|_{\sigma_i\mathcal{P}_i} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big|_{\sigma_i\mathcal{P}_i} \\
&= Z\big|_{\sigma_i\mathcal{P}_i} \cup \emptyset = Z\big|_{\sigma_i\mathcal{P}_i}\,.
\end{aligned}$$

This gives

$$\begin{aligned}
\mathcal{Z}^{(2)} &= \left\{Z \subseteq \mathcal{P}^{(2)}\colon \left(\begin{array}{l}\Big(Z\big|_{\mathcal{P}} \cup \Big\{P_{r_i} \in \{P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big) \cup \\ \Big\{P_{r_i} \in \{P_{r_1}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\end{array}\right) \in \mathcal{Z}\right\} \\
&= \Big\{Z \subseteq \mathcal{P}^{(2)}\colon \Big(Z\big|_{\mathcal{P}} \cup \Big\{P_{r_i} \in \{P_{r_1}, P_{r_2}\}\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big) \in \mathcal{Z}\Big\}\,.
\end{aligned}$$

Repeating this step k times yields the set $\mathcal{P}^{(k)}$ of processors and the tolerated structure $\mathcal{Z}^{(k)}$ of the protocol $(\sigma_k G_k)(\cdots(\sigma_2 G_2)((\sigma_1 G_1)(\pi, P_{r_1}), P_{r_2})\cdots, P_{r_k})$:

$$\begin{aligned}
\mathcal{P}^{(k)} &= \big(\mathcal{P}\setminus R\big) \cup \sigma_1\mathcal{P}_1 \cup \ldots \cup \sigma_k\mathcal{P}_k \\
\mathcal{Z}^{(k)} &= \Big\{Z \subseteq \mathcal{P}^{(k)}\colon \Big(Z\big|_{\mathcal{P}} \cup \Big\{P_{r_i} \in R\colon Z\big|_{\sigma_i\mathcal{P}_i} \notin \sigma_i\mathcal{Z}_i\Big\}\Big) \in \mathcal{Z}\Big\}\,.
\end{aligned}$$

Finally, we apply the inverse processor mappings $\sigma_1^{-1}, \ldots, \sigma_k^{-1}$. Let $\vartheta = \sigma_k^{-1} \cdots \sigma_1^{-1}$. Because $\sigma_1^{-1}, \ldots, \sigma_k^{-1}$ are bijective and have pairwise disjoint domains, all function values of ϑ are sets with a single processor and are considered as those processors (rather than as sets). Also, ϑ must be extended to be the identity function for the processors in $\mathcal{P} \backslash R$, since it will be applied to the previously constructed protocol among the set $\mathcal{P}^{(k)}$ of processors. The resulting protocol π^* for the set $\mathcal{P}^*$ of processors tolerates the structure $\mathcal{Z}^*$, where

$$
\begin{aligned}
\mathcal{P}^* &= \vartheta \mathcal{P}^{(k)} \\
&= \vartheta\Big((\mathcal{P} \backslash R) \cup \sigma_1 \mathcal{P}_1 \cup \ldots \cup \sigma_k \mathcal{P}_k \Big) \\
&= (\mathcal{P} \backslash R) \cup \sigma_1^{-1}\sigma_1 \mathcal{P}_1 \cup \ldots \cup \sigma_k^{-1}\sigma_k \mathcal{P}_k \\
&= (\mathcal{P} \backslash R) \cup \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_k \\
&= (\mathcal{P} \backslash R) \cup \bigcup_{i=1}^{k} \mathcal{P}_i \\
\mathcal{Z}^* &= \vartheta \mathcal{Z}^{(k)} \\
&= \Big\{ Z \subseteq \vartheta \mathcal{P}^{(k)} \colon \vartheta^{-1}(Z) \in \mathcal{Z}^{(k)} \Big\} \\
&= \Big\{ Z \subseteq \mathcal{P}^* \colon \Big(\vartheta^{-1}(Z)\big|_{\mathcal{P}} \cup \Big\{ P_{r_i} \in R \colon \vartheta^{-1}(Z)\big|_{\sigma_i \mathcal{P}_i} \notin \sigma_i \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} \ .
\end{aligned}
$$

Due to the definition of ϑ, we have $\vartheta^{-1}(Z)\big|_{\mathcal{P}} = Z\big|_{\mathcal{P} \backslash R}$. Since the sets $\overline{\mathcal{P}}_i$ are pairwise disjoint we have $\vartheta^{-1}(Z)\big|_{\sigma_i \mathcal{P}_i} = \sigma_i(Z)\big|_{\sigma_i \mathcal{P}_i}$ and, because σ_i is bijective, also $\sigma_i(Z)\big|_{\sigma_i \mathcal{P}_i} = \sigma_i\big(Z\big|_{\mathcal{P}_i}\big)$. Again using that σ_i is bijective implies that $\sigma_i\big(Z\big|_{\mathcal{P}_i}\big) \in \sigma_i \mathcal{P}_i$ if and only if $Z\big|_{\mathcal{P}_i} \in \mathcal{P}_i$. This results in the claimed adversary structure

$$
\mathcal{Z}^* = \Big\{ Z \subseteq \mathcal{P}^* \colon \Big(Z\big|_{\mathcal{P} \backslash R} \cup \Big\{ P_{r_i} \in R \colon Z\big|_{\mathcal{P}_i} \notin \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \Big\} \ .
$$

■

4.2 Passive Model

In this section, we prove the following theorem:

Theorem 10 *In the passive model, a set $\mathcal{P}$ of processors can compute every specification (perfectly) $\mathcal{Z}$-securely if no two sets in the adversary structure $\mathcal{Z}$ cover $\mathcal{P}$ (i.e., if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This bound is tight: if two sets cover $\mathcal{P}$, then there exist functions that cannot be computed $\mathcal{Z}$-securely. The computation is polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure and linear in the length of the specification.*

The construction of the protocols is based on a basic multi-party protocol for the passive model with threshold security. As basic protocol, we take the protocol of Section 3.1 [BGW88]. G^{p3} denotes the three-party protocol generator for the set $\mathcal{P}_{G^{\mathsf{p3}}} = \{P_1, P_2, P_3\}$ of processors for the passive model, tolerating all passive adversaries that may corrupt one single processor. We assume that G^{p3} is $\mathcal{Z}$-secure under the definitions of Section 2 for $\mathcal{Z} = \big\{\{\}, \{P_1\}, \{P_2\}, \{P_3\}\big\}$.

We first show that a set $\mathcal{P}$ of processors can compute every specification $\mathcal{Z}$-securely if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. We then show that this protocol is indeed polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure. And finally, we will prove the necessity of this condition for passively secure multi-party protocols to exist for every specification.

4.2.1 Construction

Consider a set $\mathcal{P}$ of processors and a structure $\mathcal{Z}$ for this set $\mathcal{P}$ such that $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. We construct a $\mathcal{Z}$-secure protocol generator G for the set $\mathcal{P}$ of processors, i.e., G takes as input an arbitrary specification (π_0, τ) for the set $\mathcal{P}_0$ of processors and outputs a protocol π for the set $\big(\mathcal{P}_0 \setminus \{\tau\}\big) \cup \mathcal{P}$ of processors that A-securely computes the specification (π_0, τ) for every adversary A with $Z_A\big|_{\mathcal{P}} \in \mathcal{Z}$.

If some processor $P \in \mathcal{P}$ does not occur in any set of $\mathcal{Z}$ (i.e., $\mathcal{Z}\big|_{\{P\}} = \{\emptyset\}$) then G simply replaces the trusted party τ in the specification by this processor. More precisely, let ρ_τ be the processor mapping that maps τ to P (and is the identity function for all other processors), then $G = ((\pi_0, \tau) \mapsto \rho_\tau(\pi_0))$.

Consider the case where every processor in $\mathcal{P}$ occurs in at least one set in $\mathcal{Z}$. The following construction is based on ideas of [AR63, pp. 22–24]

and [Fit96]. We select some three-partition of $\overline{\mathcal{Z}}$ where the size of each set of the partition is at most $\lceil|\overline{\mathcal{Z}}|/3\rceil$. Let $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$ be the union of the first two, the first and the third, and the last two sets of the partition, respectively, each completed such that it is monotone. Assume that protocol generators G_1, G_2, and G_3, each among the set $\mathcal{P}$ of processors, tolerating $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$, respectively, have been constructed (by recursion). The protocol generator G that tolerates $\mathcal{Z}$ can be constructed as follows: Remember that G^{p3} is the natural protocol generator of [BGW88] for the passive model for the set $\mathcal{P}_{G^{\mathsf{p3}}} = \{P_1, P_2, P_3\}$ of processors, tolerating the adversary structure $\mathcal{Z}_{G^{\mathsf{p3}}} = \{\{P_1\}, \{P_2\}, \{P_3\}\}$. Let σ be a bijective processor mapping $\sigma : \mathcal{P}_{G^{\mathsf{p3}}} \to \overline{\mathcal{P}}$, where $\overline{\mathcal{P}}$ is a set of new processor names. First, the protocol generator G applies $\sigma(G^{\mathsf{p3}})$ to the specification (π_0, τ). $\sigma(G^{\mathsf{p3}})$ is a protocol generator that tolerates $\sigma(\mathcal{Z}_{G^{\mathsf{p3}}})$ (Corollary 1), thus the resulting protocol tolerates all adversaries A with $|Z_A|_{\overline{\mathcal{P}}}| \leq 1$. Then G simultaneously simulates all three processors in $\overline{\mathcal{P}}$ by subprotocols, applying the protocol generators G_1, G_2, and G_3. This results in a protocol π^* for the set $\mathcal{P}^*$ of processors that $\mathcal{Z}^*$-securely computes the specification (π_0, τ), where according[19] to Theorem 9 the set of processors is

$$\mathcal{P}^* = \Big(\big((\mathcal{P}_0 \setminus \{\tau\}) \cup \overline{\mathcal{P}}\big) \setminus \overline{\mathcal{P}}\Big) \cup \mathcal{P} = \Big(\mathcal{P}_0 \setminus \{\tau\}\Big) \cup \mathcal{P}$$

and the tolerated adversary structure is

$$\begin{aligned}
\mathcal{Z}^* &= \left\{ Z \subseteq \mathcal{P}^* \colon \left(Z\Big|_{\left((\mathcal{P}_0 \setminus \{\tau\}) \cup \overline{\mathcal{P}}\right) \setminus \overline{\mathcal{P}}} \cup \left\{ P_{r_i} \in \overline{\mathcal{P}} \colon Z|_{\mathcal{P}} \notin \mathcal{Z}_i \right\} \right) \in \sigma \mathcal{Z}_{G^{\mathsf{p3}}} \right\} \\
&= \left\{ Z \subseteq \mathcal{P}^* \colon \left| \left(Z|_{\mathcal{P}_0 \setminus \{\tau\}} \cup \left\{ P_{r_i} \in \overline{\mathcal{P}} \colon Z|_{\mathcal{P}} \notin \mathcal{Z}_i \right\} \right)\Big|_{\overline{\mathcal{P}}} \right| \leq 1 \right\} \\
&= \left\{ Z \subseteq \mathcal{P}^* \colon \left| \left\{ P_{r_i} \in \overline{\mathcal{P}} \colon Z|_{\mathcal{P}} \notin \mathcal{Z}_i \right\} \right| \leq 1 \right\} .
\end{aligned}$$

Every set $Z \in \mathcal{Z}$ is in two of the structures $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, thus every adversary A with $Z_A|_{\mathcal{P}} \in \mathcal{Z}$ is tolerated in π^*. As claimed, the constructed protocol generator G is for the set $\mathcal{P}$ of processors and is $\mathcal{Z}$-secure.

The suitability of this construction can be proved by induction. First, consider an adversary structure $\mathcal{Z}$ satisfying $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ with $|\overline{\mathcal{Z}}| \leq 2$.

[19] Note that in Theorem 9 the protocol generators for the simulation are assumed to be natural protocol generators. The protocol generators $G_1, \ldots, G_3$ of this proof are recursively constructed protocol generators, which means that in fact only natural protocol generators (alternated with processor mappings) are applied.

Since the (at most) two sets in $\overline{\mathcal{Z}}$ do not cover $\mathcal{P}$, and all other sets in $\mathcal{Z}$ are subsets of one of the sets in the basis, there is a processor $P \in \mathcal{P}$ that does not occur in any set in $\mathcal{Z}$ (induction basis). Now assume that we can construct a protocol generator for every adversary structure which contains $2m$ of the sets in $\overline{\mathcal{Z}}$ (induction hypothesis). Then the above construction yields a protocol generator for an arbitrary adversary structure with up to $3m$ of the sets in $\overline{\mathcal{Z}}$ (induction step).

Let t_i be defined as the basis size guaranteed to be achievable with recursion of depth i. We have $t_0 = 2$, $t_1 = 3$, and $t_{i+1} = t_i + \lfloor t_i/2 \rfloor$. One can easily verify that $(3/2)^i \leq t_i \leq (3/2)^{i+2}$. Thus, in order to construct a protocol that tolerates the adversary structure $\mathcal{Z}$, the recursion depth is at most $\lceil \log_{\frac{3}{2}} |\overline{\mathcal{Z}}| \rceil$.

4.2.2 Efficiency

The protocol generator G^{p3} applied to a specification (π, P) translates every statement in π that involves P into a statement sequence of length at most b, where b is a constant parameter of G^{p3}. Considering all simultaneous simulations at a given level i of the recursion, every statement is affected by the application of at most two natural protocol generators (because every statement involves at most two processors). Hence the total blow-up due to a given level of the recursion is at most b^2. The total length of the constructed protocol tolerating $\mathcal{Z}$ is thus at most $|\pi_0| \cdot \left(b^2\right)^{\lceil \log_{\frac{3}{2}} |\overline{\mathcal{Z}}| \rceil} = |\pi_0| \cdot |\overline{\mathcal{Z}}|^{O(1)}$, which is polynomial in $|\overline{\mathcal{Z}}|$.

4.2.3 Tightness

In order to prove the necessity of the condition $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ for the existence of $\mathcal{Z}$-secure protocols, suppose there is a protocol that tolerates an adversary structure not satisfying $Q^{(2)}$, i.e., there are two potential sets Z_1 and Z_2 with $Z_1 \cup Z_2 = \mathcal{P}$. Without loss of generality we assume $Z_1 \cap Z_2 = \emptyset$. Then we can construct a protocol with two processors P_1 and P_2, where P_1 simulates all processors in Z_1 and P_2 simulates all processors in Z_2 (i.e., we apply a mapping to the given protocol), and we obtain a protocol for two processors that tolerates both sets with a single adverse processor. Such a protocol for secure function evaluation does not exist for most functions (for example for the binary OR-function), as

stated in [BGW88], thus resulting in a contradiction. A more careful analysis of the class of functions that are not securely computable if $Q^{(2)}$ is not satisfied is given in [CK89, Kus89, Bea89].

4.2.4 Example

We illustrate the construction with an example. The goal is to construct a protocol generator G for the passive model among the set

$$\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5, P_6\}$$

of processors, tolerating the adversary structure $\mathcal{Z}$ with the basis

$$\overline{\mathcal{Z}} = \Big\{\{P_1, P_4, P_6\}, \{P_2, P_3, P_6\}, \{P_1, P_2, P_6\}, \{P_1, P_2, P_5\}, \\ \{P_2, P_4, P_5\}, \{P_1, P_3, P_5\}, \{P_1, P_2, P_3, P_4\}\Big\}.$$

It is easy to verify that $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied.

As a short notation, we write $[P_i, P_j, P_k]$ for the (mapped) protocol generator G^{p3} with the three processors P_i, P_j, and P_k, and $[P_i, P_j, [P_k, P_l, P_m]]$ for the protocol generator among the processors P_i, P_j and a virtual processor simulated by a protocol generated by the protocol generator G^{p3} among the processors P_k, P_l, and P_m (i.e., a mapped protocol generator). As a special case, $[P]$ refers to the protocol generator that simply replaces the name of the trusted party in the multi-party computation specification by P. Whenever a structure is partitioned, this partition is not made explicit, but can easily be derived from the three resulting structures (we write ";" instead of "," at partition boundaries).

Figure 4.3 illustrates this protocol generator. We denote virtual processors by P_i with $i < 0$.

Step 1: Divide $\overline{\mathcal{Z}}$ into three partitions and set
$\overline{\mathcal{Z}}_1 =$
$\big\{\{P_1, P_4, P_6\}, \{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_2, P_5\}, \{P_2, P_4, P_5\}\big\}$,
$\overline{\mathcal{Z}}_2 =$
$\big\{\{P_1, P_4, P_6\}, \{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_3, P_5\}, \{P_1, P_2, P_3, P_4\}\big\}$,
$\overline{\mathcal{Z}}_3 = \big\{\{P_1, P_2, P_5\}, \{P_2, P_4, P_5\};\ \{P_1, P_3, P_5\}, \{P_1, P_2, P_3, P_4\}\big\}$.
Step 2: Construct G_1 tolerating $\overline{\mathcal{Z}}_1$.

Step 2.1: Divide $\overline{\mathcal{Z}}_1$ into three partitions and set
$\overline{\mathcal{Z}}_{11} = \big\{\{P_1, P_4, P_6\};\ \{P_2, P_3, P_6\}, \{P_1, P_2, P_6\}\big\}$,
$\overline{\mathcal{Z}}_{12} = \big\{\{P_1, P_4, P_6\};\ \{P_1, P_2, P_5\}, \{P_2, P_4, P_5\}\big\}$,
$\overline{\mathcal{Z}}_{13} = \big\{\{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_2, P_5\}, \{P_2, P_4, P_5\}\big\}$.

Step 2.2: Construct G_{11} tolerating $\overline{\mathcal{Z}}_{11}$. This is achieved by $[P_5]$.

Step 2.3: Construct G_{12} tolerating $\overline{\mathcal{Z}}_{12}$. This is achieved by $[P_3]$.

Step 2.4: Construct G_{13} tolerating $\overline{\mathcal{Z}}_{13}$.

Step 2.4.1: Divide $\overline{\mathcal{Z}}_{13}$ into three partitions and set
$\overline{\mathcal{Z}}_{131} = \big\{\{P_2, P_3, P_6\};\ \{P_1, P_2, P_6\}, \{P_1, P_2, P_5\}\big\}$,
$\overline{\mathcal{Z}}_{132} = \big\{\{P_2, P_3, P_6\};\ \{P_2, P_4, P_5\}\big\}$,
$\overline{\mathcal{Z}}_{133} = \big\{\{P_1, P_2, P_6\}, \{P_1, P_2, P_5\},\ \{P_2, P_4, P_5\}\big\}$.

Step 2.4.2: Construct G_{131} tolerating $\overline{\mathcal{Z}}_{131}$. This is achieved by $[P_4]$.

Step 2.4.3: Construct G_{132} tolerating $\overline{\mathcal{Z}}_{132}$. This is achieved by $[P_1]$.

Step 2.4.4: Construct G_{133} tolerating $\overline{\mathcal{Z}}_{133}$. This is achieved by $[P_3]$.

Step 2.4.5: $G_{13} = [P_4, P_1, P_3]$ is $\overline{\mathcal{Z}}_{13}$-secure.

Step 2.5: $G_1 = \big[P_5, P_3, [P_4, P_1, P_3]\big]$ is $\overline{\mathcal{Z}}_1$-secure.

Step 3: Construct G_2 tolerating $\overline{\mathcal{Z}}_2$.

Step 3.1: Divide $\overline{\mathcal{Z}}_2$ into three partitions and set
$\overline{\mathcal{Z}}_{21} = \big\{\{P_1, P_4, P_6\};\ \{P_2, P_3, P_6\}, \{P_1, P_2, P_6\}\big\}$,
$\overline{\mathcal{Z}}_{22} = \big\{\{P_1, P_4, P_6\};\ \{P_1, P_3, P_5\}, \{P_1, P_2, P_3, P_4\}\big\}$,
$\overline{\mathcal{Z}}_{23} = \big\{\{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_3, P_5\}, \{P_1, P_2, P_3, P_4\}\big\}$.

Step 3.2: Construct G_{21} tolerating $\overline{\mathcal{Z}}_{21}$. This is achieved by $[P_5]$.

Step 3.3: Construct G_{22} tolerating $\overline{\mathcal{Z}}_{22}$.

Step 3.3.1: Divide $\overline{\mathcal{Z}}_{22}$ into three partitions and set
$\overline{\mathcal{Z}}_{221} = \big\{\{P_1, P_4, P_6\};\ \{P_1, P_3, P_5\}\big\}$,
$\overline{\mathcal{Z}}_{222} = \big\{\{P_1, P_4, P_6\};\ \{P_1, P_2, P_3, P_4\}\big\}$,
$\overline{\mathcal{Z}}_{223} = \big\{\{P_1, P_3, P_5\};\ \{P_1, P_2, P_3, P_4\}\big\}$.

Step 3.3.2: Construct G_{221} tolerating $\overline{\mathcal{Z}}_{221}$. This is achieved by $[P_2]$.

Step 3.3.3: Construct G_{222} tolerating $\overline{\mathcal{Z}}_{222}$. This is achieved by $[P_5]$.

Step 3.3.4: Construct G_{223} tolerating $\overline{\mathcal{Z}}_{223}$. This is achieved by $[P_6]$.

Step 3.3.5: $G_{22} = [P_2, P_5, P_6]$ is $\overline{\mathcal{Z}}_{22}$-secure.

Step 3.4 Construct G_{23} tolerating $\overline{\mathcal{Z}}_{23}$.

Step 3.4.1: Divide $\overline{\mathcal{Z}}_{23}$ into three partitions and set
$\overline{\mathcal{Z}}_{231} = \big\{\{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_3, P_5\}\big\}$,
$\overline{\mathcal{Z}}_{232} = \big\{\{P_2, P_3, P_6\}, \{P_1, P_2, P_6\};\ \{P_1, P_2, P_3, P_4\}\big\}$,
$\overline{\mathcal{Z}}_{233} = \big\{\{P_1, P_3, P_5\};\ \{P_1, P_2, P_3, P_4\}\big\}$.

Step 3.4.2: Construct G_{231} tolerating $\overline{\mathcal{Z}}_{231}$. This is achieved by $[P_4]$.
Step 3.4.3: Construct G_{232} tolerating $\overline{\mathcal{Z}}_{232}$. This is achieved by $[P_5]$.
Step 3.4.4: Construct G_{233} tolerating $\overline{\mathcal{Z}}_{233}$. This is achieved by $[P_6]$.
Step 3.4.5: $G_{23} = [P_4, P_5, P_6]$ is $\overline{\mathcal{Z}}_{23}$-secure.

Step 3.5: $G_2 = \big[P_5, [P_2, P_5, P_6], [P_4, P_5, P_6]\big]$ is $\overline{\mathcal{Z}}_2$-secure.

Step 4: Construct G_3 tolerating $\overline{\mathcal{Z}}_3$. This is achieved by $[P_6]$.
Step 5: The protocol generator

$$G = \Big[\big[P_5, P_3, [P_4, P_1, P_3]\big], \big[P_5, [P_2, P_5, P_6], [P_4, P_5, P_6]\big], P_6\Big]$$

is $\mathcal{Z}$-secure.

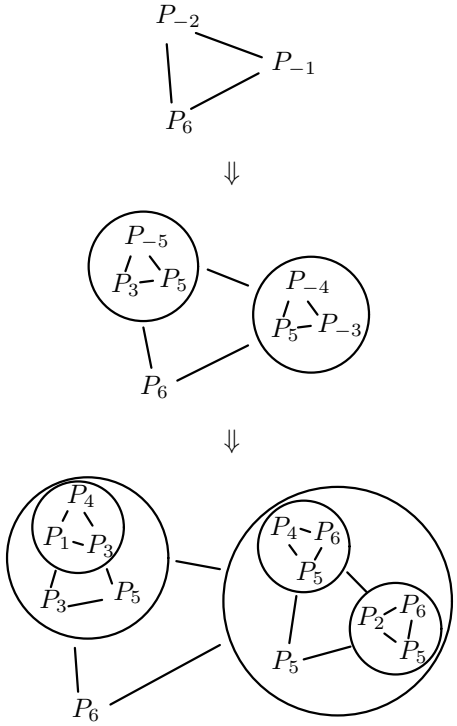


Figure 4.3: An example of recursive processor simulation.

4.3 Active Model

We prove the following theorem:

Theorem 11 *In the active model, a set $\mathcal{P}$ of processors can compute every specification (perfectly) $\mathcal{Z}$-securely if no three sets in the adversary structure $\mathcal{Z}$ cover $\mathcal{P}$ (i.e., if $Q^{(3)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This bound is tight: if three sets cover the full set of processors, there are functions that cannot be computed $\mathcal{Z}$-securely. The computation is polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure and linear in the length of the specification.*

The construction of the protocols is based on a basic multi-party protocol for the active model with threshold security. As basic protocol, we take the protocol of Section 3.2 [BGW88]. G^{a4} denotes the four-party protocol generator for the set $\mathcal{P}_{G^{\mathsf{a4}}} = \{P_1, P_2, P_3, P_4\}$ of processors for the active model, tolerating all active adversaries that may corrupt one single processor. We assume that G^{a4} is $\mathcal{Z}$-secure for $\mathcal{Z} = \{\{\}, \{P_1\}, \{P_2\}, \{P_3\}, \{P_4\}\}$.

We first show that a set $\mathcal{P}$ of processors can compute every specification $\mathcal{Z}$-securely if $Q^{(3)}(\mathcal{P}, \mathcal{Z})$ is satisfied. We then show that this protocol is indeed polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure. And finally, we will prove the necessity of this condition for actively secure multi-party protocols to exist for every specification.

4.3.1 Construction

A four-partition of the adversary structure $\overline{\mathcal{Z}}$ is selected where the size of each set of the partition is at most $\lceil |\overline{\mathcal{Z}}|/4 \rceil$. By recursion, a protocol is constructed for each of the four unions of three set of the partition. First, the protocol generator applies G^{a4} in order to substitute the trusted party τ in the specification by a protocol among four virtual processors, then simultaneously replaces the four virtual processors by applying the recursively constructed protocol generators. Applying Theorem 9 shows that the resulting protocol tolerates the adversary structure $\mathcal{Z}$.

The induction basis (there is a processor $P \in \mathcal{P}$ that does not occur in $\mathcal{Z}$) holds for any structure $\mathcal{Z}$ with $|\overline{\mathcal{Z}}| \leq 3$, and the induction step constructs a protocol generator that tolerates $4m$ of the sets in $\mathcal{Z}$ by assuming protocol generators that tolerate $3m$ of the sets.

4.3.2 Efficiency

Let b be the constant "blow-up factor" of G^{a4}, and let u_i be defined as the minimal size of the basis of the adversary structures guaranteed to be achievable with recursion of depth i. The sequence u_i is hence given by $u_0 = 3$, $u_1 = 4$, and $u_{i+1} = u_i + \lfloor u_i/3 \rfloor$. One can easily verify that $(4/3)^i \leq u_i \leq (4/3)^{i+3}$. Thus, in order to construct a protocol that tolerates the adversary structure $\mathcal{Z}$, the recursion depth is at most $\lceil \log_{\frac{4}{3}} |\overline{\mathcal{Z}}| \rceil$, and the total length of the constructed protocol tolerating $\mathcal{Z}$ is at most $|\pi_0| \cdot \left(b^2\right)^{\lceil \log_{\frac{4}{3}} |\overline{\mathcal{Z}}| \rceil} = |\pi_0| \cdot |\overline{\mathcal{Z}}|^{O(1)}$, which is polynomial in $|\overline{\mathcal{Z}}|$.

4.3.3 Tightness

In order to prove the necessity of condition $Q^{(3)}(\mathcal{P}, \mathcal{Z})$, suppose that there exists a protocol generator for an adversary structure not satisfying $Q^{(3)}$, i.e., there are three potential adversaries that cover the full set of processors. Then we can construct a protocol among three processors, where each of them simulates the processors of one adversary, and we obtain a protocol among three processors, perfectly tolerating active cheating of one of them. Such a protocol for secure function evaluation does not exist for most functions (for example for the broadcast function, as proved in [PSL80, LSP82]), thus resulting in a contradiction.

4.4 Active Model with Broadcast

When broadcast channels are available, the condition for secure MPC protocols to exist can be weakened:

Theorem 12 *In the active model with broadcast, a set $\mathcal{P}$ of processors can compute every specification (unconditionally) $\mathcal{Z}$-securely if no two sets in the adversary structure $\mathcal{Z}$ cover $\mathcal{P}$ (i.e., if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied). This bound is tight: if two sets cover the full set of processors, there are functions that cannot be computed $\mathcal{Z}$-securely. The computation is polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure and linear in the length of the specification.*

Protocols with unconditional security are constructed along the lines of the construction of perfectly secure protocols for the active model, except that on the lowest level of the recursion, threshold protocols tolerating a faulty minority are employed (Section 3.3). We denote this protocol

generator for the set $\mathcal{P}_{G^{\mathsf{a3b}}} = \{P_1, P_2, P_3\}$ with G^{a3b}, and assume that it is (actively) $\mathcal{Z}$-secure for $\mathcal{Z} = \big\{\{\}, \{P_1\}, \{P_2\}, \{P_3\}\big\}$.

4.4.1 Construction

Consider a set $\mathcal{P}$ of processors and a structure $\mathcal{Z}$ for the set $\mathcal{P}$ such that $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ is satisfied, and an arbitrary specification (π_0, τ). We have to construct a $\mathcal{Z}$-secure protocol π for the set $\mathcal{P}$ of processors.

The case $\big|\overline{\mathcal{Z}}\big| \leq 3$ is simple. Since we have $Q^{(2)}(\mathcal{P}, \mathcal{Z})$, we have three processors P_1, P_2, and P_3, where P_i may occur in the i-th set of $\mathcal{Z}$, but does not occur in the other sets, hence we can apply G^{a3b} to obtain an unconditionally $\mathcal{Z}$-secure protocol for $\mathcal{P}$.

The case of a basis with at least four classes is treated along the lines of the construction in Section 4.3.1: First we select some four-partition of $\overline{\mathcal{Z}}$ and, by recursion, a protocol is constructed for each of the four unions of three subsets of the partition. Then, these four protocols are composed to a four-party protocol with using G^{a4}.

4.4.2 Efficiency

The efficiency of this protocol can be analyzed along the lines of the analysis given in Section 4.3.2. However, as the protocol of Section 3.3 which is used in the lowest level of the substitution tree provides some (negligible) error probability, special care is required in the analysis (cf. [HM97]). It follows immediately from the analysis in Section 4.3.2 that the protocol which results after applying all substitutions except for those on the lowest level, has polynomial complexity. But every statement of this protocol is expanded at most twice by all the remaining substitutions (once per involved processor), and each blow-up is polynomial, and hence the final protocol is also polynomial in the number $|\overline{\mathcal{Z}}|$ of maximal sets in the adversary structure.

Note that the construction for the active model with broadcast given in [HM97] have super-polynomial complexity.

4.4.3 Tightness

The necessity of the condition $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ follows immediately from the necessity of this same condition in the passive model (Section 4.2.3).

4.5 Mixed Model with Perfect Security

When perfect security without any probability of failure is required, then broadcast channels do not help. The following theorem states the necessary (even with broadcast channels) and sufficient (even without broadcast channels) conditions on the adversary structure for secure MPC protocols to exist:

Theorem 13 *A set $\mathcal{P}$ of processors can compute every specification (perfectly) $\mathcal{Z}$-securely if $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. This bound is tight: if $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is not satisfied, then there exist functions that cannot be computed perfectly $\mathcal{Z}$-securely, even if a broadcast channel is available.*[20] *The communication and computation complexities are polynomial in the size $|\overline{\mathcal{Z}}|$ of the basis of the adversary structure and linear in the length of the specification.*

The construction of protocols for the mixed model is very similar to the construction in the univariate models. Indeed, only in the lowest level of the recursion, we take advantage of the fact that some of the corrupted processors are only passively corrupted. On the higher levels, where subprotocols are combined with each other, only active corruption is considered. Surprisingly, this simple approach yields secure multi-party protocols for all adversary structures for which such protocols exist.

As usual, we first show that a set $\mathcal{P}$ of processors can compute every specification $\mathcal{Z}$-securely if $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. We then show that this protocol is indeed polynomial in the size of the basis $|\overline{\mathcal{Z}}|$ of the adversary structure. And finally, we will prove the necessity of this condition for perfectly secure multi-party protocols to exist for every specification in the mixed model, even when broadcast channels are available.

4.5.1 Construction

The construction of perfectly secure protocols for the mixed model is along the lines of the constructions in the active model, but using another basic protocol. We first show how for every admissible mixed adversary structure with only three classes in the basis such a basic protocol can be constructed. Then, we will use such basic protocols as ingredients of the recursive construction.

[20] Indeed, almost every non-trivial function cannot be computed perfectly $\mathcal{Z}$-securely.

Lemma 3 *A set $\mathcal{P}$ of processors can compute every specification perfectly $\mathcal{Z}$-securely if $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ and $|\overline{\mathcal{Z}}| \leq 3$. The computation and communication complexities are linear in the size of the specification.*

Proof: Consider an arbitrary adversary structure $\mathcal{Z}$ with $|\overline{\mathcal{Z}}| \leq 3$ that satisfies $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$, and a specification (π_0, τ). We show that there exists a subset of the processors that can compute the specification in a secure way. If $|\overline{\mathcal{Z}}| < 3$, then the condition $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ immediately implies that there is a processor $P \in \mathcal{P}$ that is not contained in any class of $\overline{\mathcal{Z}}$ (i.e., $\mathcal{Z}\big|_{\{P\}} = \{(\emptyset, \emptyset)\}$). Hence this processor cannot be corrupted by any admissible adversary, and one can simply replace the occurrence of the trusted party τ in the protocol π_0 of the specification by the name of this processor. Thus assume that $|\overline{\mathcal{Z}}| = 3$ and $\overline{\mathcal{Z}} = \{(D_1, E_1), (D_2, E_2), (D_3, E_3)\}$. Condition $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ implies that there exists a processor $P_3 \in \mathcal{P}$ with $P_3 \notin D_1 \cup E_1 \cup D_2 \cup E_2 \cup D_3$ (but potentially $P_3 \in E_3$). Hence this processor remains uncorrupted by an adversary of the first or the second class, and is (at most) passively corrupted by an adversary of the third class. By symmetry reasons, there exist processors P_1 and P_2 which can be corrupted at most passively and only by an adversary of the first or the second class, respectively. This means that every admissible adversary may corrupt none of the processors P_1, P_2, or P_3 actively and only at most one of them passively. Hence, these three processors can simulate the trusted party of the specification by using the protocol generator for the passive model, G^{p3}, for three processors (cf. Section 4.2). The other processors (if any) are not involved in the simulation of the trusted party. ∎

Consider a set $\mathcal{P}$ of processors and a structure $\mathcal{Z}$ for this set $\mathcal{P}$ such that $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied, and an arbitrary specification (π_0, τ). We recursively construct a $\mathcal{Z}$-secure protocol π:

The case $|\overline{\mathcal{Z}}| \leq 3$ was treated in Lemma 3 (induction basis). Thus assume that $|\overline{\mathcal{Z}}| \geq 4$, and that for all adversary structures with basis size strictly less than k there exists a secure protocol (induction hypothesis). We select some four-partition of $\overline{\mathcal{Z}}$ where the size of each set of the partition is at least $\lfloor |\overline{\mathcal{Z}}|/4 \rfloor$. Let $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, and $\mathcal{Z}_4$ be the four unions of three distinct sets of the partition, each of them completed such that it is monotone. Since $|\overline{\mathcal{Z}}| \geq 4$, the size $|\overline{\mathcal{Z}_i}|$ of the basis of each such structure is strictly smaller than the size $|\overline{\mathcal{Z}}|$ of the current structure basis, i.e., $|\overline{\mathcal{Z}_i}| < |\overline{\mathcal{Z}}|$ $(1 \leq i \leq 4)$, and one can recursively construct protocols π_1,

π_2, π_3, and π_4, each among the set $\mathcal{P}$ of processors, tolerating $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, and $\mathcal{Z}_4$, respectively (hypothesis). The protocol π that tolerates $\mathcal{Z}$ can be constructed as follows:

First, one uses G^{a4} constructs a protocol among four "virtual" processors that computes the specification (π_0, τ), tolerating an adversary that actively corrupts a single processor (Section 4.3). Then one simulates the four virtual processors by the recursively constructed protocols $\pi_1, \ldots, \pi_4$, respectively. Since every adversary class is tolerated by at least three of the protocols π_1, π_2, π_3, and π_4 (thus only one of the virtual processors in the main protocol is misbehaving), the resulting protocol tolerates all adversary classes in the adversary structure and, as claimed, the constructed protocol π is $\mathcal{Z}$-secure.

4.5.2 Efficiency

In order to analyze the efficiency of the protocols, we need the help of the following observation: The protocol generators G^{p3} and G^{a4} (Sections 4.2 and 4.3) have a constant "blow-up factor" b_p and b_a, respectively, i.e., for any specification of length l, the length of the protocol computing this specification is bounded by $b_p \cdot l$ in the passive model and by $b_a \cdot l$ in the active model.

In the construction given above, on each recursion level all involved processors are simulated by using protocols for the active model (Section 4.3), except for the lowest level, where passively secure protocols (Section 4.2) are used. The simulations on each level can be performed independently, and every statement in the current protocol is affected by at most two such simulations (as at most two processors occur in one statement). Hence, the total blow-up of all simulations on a given level is bounded by b_a^2 (b_p^2 on the lowest level), and as the recursion depth of the construction is logarithmic in the number $|\overline{\mathcal{Z}}|$ of maximal sets in the adversary structure, the total blow-up is polynomial in $|\overline{\mathcal{Z}}|$.

4.5.3 Tightness

In order to prove the tightness of the theorem, assume an adversary structure $\mathcal{Z}$ for which every function can be computed perfectly $\mathcal{Z}$-securely and suppose $Q^{(3,2)}(\mathcal{P}, \mathcal{Z})$ is not satisfied. Then there exist three classes $(D_1, E_1), (D_2, E_2), (D_3, E_3) \in \mathcal{Z}$ with $D_1 \cup E_1 \cup D_2 \cup E_2 \cup D_3 = \mathcal{P}$, and

(due to the monotonicity of $\mathcal{Z}$) with the sets D_1, E_1, D_2, E_2 and D_3 being pairwise disjoint.

One can construct a protocol for three processors $\hat{P}_1$, $\hat{P}_2$, and $\hat{P}_3$, where $\hat{P}_1$ plays for all the processors in $D_1 \cup E_1$, $\hat{P}_2$ plays for those in $D_2 \cup E_2$, and $\hat{P}_3$ plays for those in D_3. This new protocol is secure with respect to an adversary that passively corrupts either $\hat{P}_1$ or $\hat{P}_2$, or actively corrupts $\hat{P}_3$. It was proven in Section 3.5.4 that such a protocol does not exist for many specifications.

4.6 Mixed Model with Unconditional Security with Broadcast

In the mixed model with unconditional security and with assuming broadcast channels, it turns out that the necessary conditions for secure MPC implies the sufficient condition for the active model with broadcast. In other words, though some of the processors are corrupted only passively, the adversary cannot be allowed to corrupt any more of them. Hence, constructing protocols for the mixed model with broadcast is based on the protocols for the active model with broadcast. Still, the tightness of the condition for the mixed model with broadcast must be proven.

Theorem 14 *If a broadcast channel is available, a set $\mathcal{P}$ of processors can compute every specification unconditionally $\mathcal{Z}$-securely if $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. This bound is tight: if $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is not satisfied, then there exist functions that cannot be computed unconditionally $\mathcal{Z}$-securely. The communication and computation complexities of the protocol are polynomial in the size $|\overline{\mathcal{Z}}|$ of the basis of the adversary structure and linear in the length of the specification.*

4.6.1 Construction

Consider a set $\mathcal{P}$ of processors and a structure $\mathcal{Z}$ for this set $\mathcal{P}$ such that $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied. We define the adversary structure $\mathcal{Z}'$ for the *active* model such that it contains all set of processors that can be corrupted (actively or passively) at once, i.e., $\mathcal{Z}' = \{D \cup E \colon (D, E) \in \mathcal{Z}\}$. Every mixed adversary tolerated by $\mathcal{Z}$ is also tolerated by $\mathcal{Z}'$, and as $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ implies $Q^{(3)}(\mathcal{P}, \mathcal{Z}')$, we can use the protocol of Section 4.4 to tolerate $\mathcal{Z}$.

4.6.2 Tightness

We prove that for every adversary structure $\mathcal{Z}$ for a processor set $\mathcal{P}$ not satisfying $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$, there exist specifications that cannot be computed unconditionally $\mathcal{Z}$-securely, even when broadcast channels are available. For the sake of contradiction, assume that for an adversary structure $\mathcal{Z}$ for which $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is not satisfied, there exists an unconditional $\mathcal{Z}$-secure protocol for every function. Clearly, this protocol is *passively* $\mathcal{Z}'$-secure for $\mathcal{Z}' = \{D \cup E \colon (D, E) \in \mathcal{Z}\}$, though $Q^{(2)}(\mathcal{P}, \mathcal{Z}')$ is not satisfied, contradicting Theorem 10.

4.7 Mixed Model with Unconditional Security without Broadcast

When no broadcast channels are available, then the broadcast primitive is simulated by a sub-protocol for Byzantine agreement. This yields an additional condition, namely $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$, for the existence of secure MPC protocols.

Theorem 15 *A set $\mathcal{P}$ of processors can compute every specification unconditionally $\mathcal{Z}$-securely if $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ and $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$ are satisfied. This bound is tight: if $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ or $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$ is not satisfied, then there exist functions that cannot be computed unconditionally $\mathcal{Z}$-securely. The communication and computation complexities of the protocol are polynomial in the size $|\overline{\mathcal{Z}}|$ of the basis of the adversary structure and linear in the length of the specification.*

4.7.1 Construction

Consider a set $\mathcal{P}$ of processors and a structure $\mathcal{Z}$ for this set $\mathcal{P}$ such that $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ and $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$ are satisfied. The condition $Q^{(3,0)}$ implies the existence of an efficient secure protocol for broadcast [FM98], and hence the construction of Section 4.6.1 (with replacing the invocations to the broadcast primitive by broadcast sub-protocols) yields a $\mathcal{Z}$-secure protocol.

4.7.2 Efficiency

The efficiency of this construction follows immediately from the efficiency of the protocol for the mixed model with broadcast channels and the efficiency of the broadcast simulation protocol.

4.7.3 Tightness

The necessity of $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is already proven (Section 4.6.2). Thus assume that $Q^{(2,2)}(\mathcal{P}, \mathcal{Z})$ is satisfied but not $Q^{(3,0)}(\mathcal{P}, \mathcal{Z})$, i.e., there exist three classes $(D_1, E_1), (D_2, E_2), (D_3, E_3) \in \mathcal{Z}$ with $D_1 \cup D_2 \cup D_3 = \mathcal{P}$ (and D_1, D_2, and D_3 pairwise disjoint). For the sake of contradiction, assume that for every function a $\mathcal{Z}$-secure multi-party protocol exists, hence in particular for the broadcast function. One can hence construct a broadcast protocol for the three processors $\hat{P}_1$, $\hat{P}_2$, and $\hat{P}_3$ (where each processor $\hat{P}_1$, $\hat{P}_2$, and $\hat{P}_3$ "plays" for the processors in one of the sets D_1, D_2, and D_3, respectively), where the adversary is allowed to actively corrupt one of them, contradicting the result that broadcast for three processors is not possible if the adversary may actively corrupt one of the processors, even if a negligible error probability is tolerated [LSP82, KY].

4.8 Counting Adversary Structures

The goal of this section is, informally, to prove that there exists a family of adversary structures and specifications, such that the length of every resilient protocol computing this specification grows exponentially in the number of processors. This means that every resilient protocol has exponential communication and / or computation complexity.

For a specification (π_0, τ), a set $\mathcal{P}$ of processors, and an adversary structure $\mathcal{Z}$, let $\varphi\big((\pi_0, \tau), \mathcal{P}, \mathcal{Z}\big)$ denote the length of the shortest protocol π for $\mathcal{P}$ that $\mathcal{Z}$-securely computes (π_0, τ). Furthermore, let (π_*, τ) denote the specification for the processors P_1 and P_2 that reads one input of both processors, computes the product and hands it to P_1. Finally, let $\mathcal{P}_n$ denote the set $\{P_1, \ldots, P_n\}$ of processors.

Theorem 16 *For any of the considered models there exists a family $\mathcal{Z}_2, \mathcal{Z}_3, \ldots$ of adversary structures for the sets $\mathcal{P}_2, \mathcal{P}_3, \ldots$ of processors, respectively, such that the length $\varphi\big((\pi_*, \tau), \mathcal{P}_n, \mathcal{Z}_n\big)$ of the shortest $\mathcal{Z}_n$-secure protocol for (π_*, τ) grows exponentially in n.*

Note that if this theorem hold for the passive and for the active model, then trivially it also holds for all mixed models; hence it is sufficient to prove it for the passive and for the active model. In order to prove the theorem we need an additional definition: An admissible adversary structure $\mathcal{Z}$ for the set $\mathcal{P}$ of processors is *maximal* if $Q^{(2)}(\mathcal{P}, \mathcal{Z})$ (in the passive model) or $Q^{(3)}(\mathcal{P}, \mathcal{Z})$ (in the active model) is satisfied, but any adversary structure $\mathcal{Z}'$ with $\mathcal{Z} \subset \mathcal{Z}'$ (and $\mathcal{Z} \neq \mathcal{Z}'$) violates $Q^{(2)}(\mathcal{P}, \mathcal{Z}')$, or $Q^{(3)}(\mathcal{P}, \mathcal{Z}')$, respectively.

Proof: The proof proceeds in three steps: First we prove that in both models, the number of maximal admissible adversary structures grows doubly-exponentially in the number n of processors. In the second step, we show that for the given specification (π_*, τ), for every maximal admissible adversary structure a different protocol is required. Finally, we conclude that for some adversary structures the length of every secure protocol is exponential in the number of processors.

1. First consider the passive model. Without loss of generality, assume that $n = |\mathcal{P}|$ is odd, and let $m = (n+1)/2$. Fix a processor $P \in \mathcal{P}$, and consider the set $\mathcal{B}$ that contains all subsets of $\mathcal{P} \setminus \{P\}$ with exactly m processors, i.e., $\mathcal{B} = \{Z \subseteq (\mathcal{P} \setminus \{P\}) \colon |Z| = m\}$. For each subset $\mathcal{B}' \subseteq \mathcal{B}$, we define $\mathcal{Z}_{\mathcal{B}'}$ to be the adversary structure that contains all sets in $\mathcal{B}'$, plus all sets $Z \subseteq \mathcal{P}$ with $|Z| < n/2$ and $(\mathcal{P} \setminus Z) \notin \mathcal{B}'$. One can easily verify that $\mathcal{Z}_{\mathcal{B}'}$ is admissible and maximal, and that for two different subsets $\mathcal{B}', \mathcal{B}'' \subseteq \mathcal{B}$, the structures $\mathcal{Z}_{\mathcal{B}'}$ and $\mathcal{Z}_{\mathcal{B}''}$ are different. The size of $\mathcal{B}$ is $|\mathcal{B}| = \binom{n-1}{m} = 2^{\Omega(n)}$, hence there are $2^{2^{\Omega(n)}}$ different subsets $\mathcal{B}'$ of $\mathcal{B}$, and thus doubly-exponentially many different maximal admissible adversary structures for the passive model.

   For the active model, fix an arbitrary processor P', and consider the set of all maximal admissible adversary structures for $\mathcal{P} \setminus \{P'\}$ for the passive model. Each such structure $\mathcal{Z}$ satisfies $Q^{(2)}(\mathcal{P} \setminus \{P'\}, \mathcal{Z})$, and hence the extended adversary structure $\mathcal{Z}' = \mathcal{Z} \cup \{\{P'\}\}$ satisfies $Q^{(3)}(\mathcal{P}, \mathcal{Z}')$. There exists at least one maximal adversary structure $\widehat{\mathcal{Z}} \supseteq \mathcal{Z}'$ for $\mathcal{P}$, and for two different adversary structures $\mathcal{Z}_1'$ and $\mathcal{Z}_2'$, also $\widehat{\mathcal{Z}_1}$ and $\widehat{\mathcal{Z}_2}$ are different (given $\widehat{\mathcal{Z}}$, one can easily compute $\mathcal{Z}'$ by deleting all sets containing P' in $\widehat{\mathcal{Z}}$). Hence, the number of maximal admissible adversary structures for the active model with

n processors is at least as large as for the passive model with $n-1$ processors, thus doubly-exponential in n.

2. Let $\mathcal{Z}$ be a maximal admissible adversary structure, and let π be a protocol that $\mathcal{Z}$-securely computes (π_*, τ). For the sake of contradiction, assume that for some other maximal admissible adversary structure $\mathcal{Z}'$ (where $\mathcal{Z}' \neq \mathcal{Z}$), the same protocol π $\mathcal{Z}'$-securely computes (π_*, τ). Then π would $(\mathcal{Z} \cup \mathcal{Z}')$-securely compute (π_*, τ). However, since both $\mathcal{Z}$ and $\mathcal{Z}'$ are maximal admissible, $(\mathcal{Z} \cup \mathcal{Z}')$ is not admissible, and hence no such protocol exists (see Theorem 10 and 11). Hence, for each maximal admissible adversary structure $\mathcal{Z}$ a different protocol π is required for securely computing (π_*, τ).

3. There are doubly-exponentially many maximal admissible adversary structures, and for each of them, a different protocol is required, hence there are doubly-exponentially many different protocols. This implies that some of these protocols have exponential length. ■

Chapter 5

Receipt-Free Secret-Ballot Voting

5.1 Introduction

In this section, we consider the problem of receipt-freeness in secret-ballot voting. Classical voting schemes guarantee the correctness of the output while preserving the secrecy of each vote. In a model with vote-buyers and coercers, it is not sufficient that an attacker cannot determine the vote of any voter, but additionally one must require that even the voter himself cannot convince the attacker about the cast vote. Schemes which disable the voter from proving his own vote are called *receipt-free*.

We propose two novel receipt-free voting protocols based on homomorphic encryption. Both protocols involve a set of N authorities. The security of the protocols is specified with respect to a fixed parameter t: The correctness of the computed tally is guaranteed as long as at least t authorities remain honest during the whole protocol execution, and the secrecy of each vote is guaranteed as long as no t authorities maliciously collaborate with each other.[21] Vote-buying and blackmailing is disabled under the assumption that untappable channels are available. This is the weakest assumption under which receipt-free voting protocols are known to exist.

[21] In contrast to the context of multi-party protocols, where t usually denotes the upper bound on the number of dishonest parties, here t denotes the number of *honest* authorities required for tallying (as well as for opening any single ballot).

The proposed protocols capture K-out-of-L votes, where every voter may vote for any K candidates out of a list of L candidates. Clearly, Yes / No-votes and single-choice candidate elections are special cases of K-out-of-L votings (with $K = 1$).

The first proposed protocol is a variation of the receipt-free voting protocol of Hirt and Sako [HS00]. Compared to the original protocol, the variation prevents the heavy computation load for decrypting the tally when the number of candidates L is large, and it makes the protocol resistant against the randomization attack [Sch99].

The second protocol borrows an idea from the protocol by Lee and Kim [LK00]: Each voter privately sends an encryption of his vote to a *randomizer*, who changes the randomness in the encryption and casts this new ballot as the voter's ballot. Furthermore (and in contrast to [LK00]), the voter and the randomizer jointly generate a proof of validity for the new ballot. Hence, the trustworthiness of the randomizer is required for receipt-freeness, but not for privacy or correctness.

Finally, we analyze the security of the protocols of [BT94] and [LK00] and show that they are not receipt-free, in contrast to what is claimed in the papers.

5.1.1 Receipt-Freeness

In a model with vote-buyers and coercers, a voting protocol must not only ensure that each voter *can* keep his vote private, but rather that each voter *must* keep it private. Still, a voter can be coerced or payed for casting a particular vote, but as the voter is not able to verifiably reveal the cast vote, the coercer or vote-buyer cannot verify whether the voter followed his instructions. The inability of constructing a verifiable receipt of the cast vote is equivalent to the ability of constructing fake "receipts". Whatever receipt the voter could construct when casting the requested vote, he can create an indistinguishable receipt even when casting any other vote.

There is a subtle difference in the requirements for preventing blackmailing on the one hand and for preventing vote-buying on the other hand: In the context of blackmailing, any (non-negligible) probability of the voter being caught when presenting a fake receipt to the coercer is unacceptable. However, in the context of vote-buying, it might be sufficient that a vote-buyer cannot distinguish fake receipts from real receipts with reasonable probability.

Receipt-freeness cannot be achieved without some (physical) assumptions on the communication channels. If the coercer can read and write all communication channels from and to the voter, then obviously the voter can give him all his secrets, and then the coercer can perfectly impersonate the voter and cast the requested vote. The weakest assumption for which receipt-free voting schemes are known to exist are one-way untappable channels from the authorities to the voters. The secrecy in these channels is unconditional: Even the voter cannot convince any other party what particular string he has received through these channels. In the scheme of this chapter, these channels need not be authentic.

Another requirement for achieving receipt-freeness is that in general, authorities cannot be allowed to collude with a coercer or a vote-buyer. The joint information of all communication channels, together with all secrets of the voter, uniquely determine the voter's vote. Hence, a voter who wants to present a fake receipt must lie for the communication transcript with at least one authority. If this authority colludes with the coercer, then the voter is caught lying. In the context of vote-buying, it may be acceptable that the voter selects one authority at random and lies about this transcript, and will be successful with probability linear in the number of colluding authorities. In the context of black-mailing, a linear probability of being caught is usually not acceptable.

5.1.2 Entities and Network

We consider a model with N authorities $A_1, \ldots, A_N$ and M voters. For authentication purposes, to each voter a secret key and a public key is associated, where the public key must be publicly known and the secret key must be kept private. We stress that in order to achieve receipt-freeness it must be guaranteed that each voter knows the secret key corresponding to his known public key (but the voter is allowed to reveal the secret-key to the coercer). For the case that this is not guaranteed by the underlying public-key infrastructure, we provide a protocol for ensuring that a voter knows his own secret key. Furthermore, we assume that there exists a proof protocol for proving knowledge of one's secret key. This protocol must be three-move honest-verifier zero-knowledge (see below).

Communication takes place by means of a bulletin board which is publicly readable, and which every participant can write to (into his own section), but nobody can delete from. The bulletin board can be considered as an authenticated public channel with memory. Furthermore, we

assume the existence of untappable channels. The privacy of an untappable channel must be physical, in such a way that even the recipient cannot prove what was received from the channel (of course, the recipient can record all received data, but he must not be able to *prove* to a third party that he received a particular string). The untappable channels need not to be authenticated on the sender's side.

5.2 Preliminaries

5.2.1 Homomorphic Encryption Scheme

We consider a probabilistic public-key encryption function

$$E_Z : \mathbb{V} \times \mathbb{R} \to \mathbb{E}, \quad (v, \alpha) \mapsto e,$$

where Z denotes the public key, $\mathbb{V}$ denotes a set of votes, $\mathbb{R}$ denotes the set of random strings, and $\mathbb{E}$ denotes the set of encryptions. We write E instead of E_Z for shorthand. For simplicity, we assume that E is surjective. The decryption function is

$$D_z : \mathbb{E} \to \mathbb{V}, \quad e \mapsto v,$$

where z denotes the secret key. Again, we write D instead of D_z.

An encryption function is semantically secure if for any given encryption e and any two candidate votes v_1 and v_2, where one of them is a decryption of e, it is infeasible to determine which vote is contained in e with probability significantly higher than 0.5, unless the secret key z is known [GM84].

A group homomorphism f is a function that maps elements from a group $(\mathbb{A}, \oplus)$ onto a group $(\mathbb{B}, \otimes)$, such that the group structure is preserved, i.e., $f(x) \otimes f(y) = f(x \oplus y)$ for any $x, y \in \mathbb{A}$. An encryption scheme for which the encryption function $E : \mathbb{V} \times \mathbb{R} \to \mathbb{E}$ is a group homomorphism (where $\mathbb{A} = \mathbb{V} \times \mathbb{R}$ and $\mathbb{B} = \mathbb{E}$) is called a *homomorphic encryption scheme*. The group operation in $\mathbb{V}$ is denoted by $+$,[22] the one in $\mathbb{R}$ by $\boxplus$, and the one in $\mathbb{E}$ by $\oplus$. All groups are Abelian, and the group operations are efficient, as is computing inverses. The groups are written

[22] Usually, one will require that the group $(\mathbb{V}, +)$ is equal to $\mathbb{Z}_{|\mathbb{V}|}$ (respectively isomorphic with an efficiently computable and invertible isomorphism), such that the tally indeed represents the sum of the cast votes. However, this is not a formal requirement.

as additive groups, e.g., ke for $k \in \mathbb{Z}$ and $e \in \mathbb{E}$ denotes the sum $e \oplus \ldots \oplus e$ (k times). The encryption function is homomorphic means that for any two elements $v_1, v_2 \in \mathbb{V}$ and $\alpha_1, \alpha_2 \in \mathbb{R}$,

$$E(v_1, \alpha_1) \oplus E(v_2, \alpha_2) = E(v_1 + v_2, \alpha_1 \boxplus \alpha_2).$$

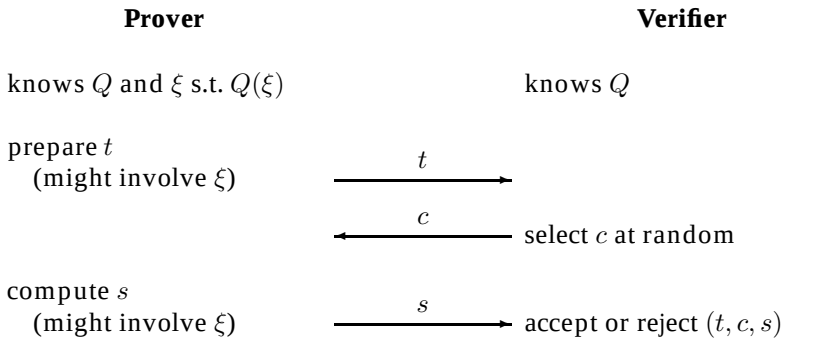
We say that an encryption function E is *q-invertible* for a given $q \in \mathbb{Z}$ exactly if for every encryption e, the decryption v and the randomness α of qe can be efficiently computed, i.e., the function $D_q : e \mapsto (v, \alpha)$ such that $qe = E(v, \alpha)$ is efficient (given Z). In contrast to normal decryption where only v must be computed such that there exists an α with $e = E(v, \alpha)$, for q-invertibility we require that both v and α can be computed such that $qe = E(v, \alpha)$. This notion of q-invertibility is inspired by the notion of q-one-way group-homomorphism of Cramer [Cra96, CD98].

Finally, a threshold encryption scheme is an encryption scheme with distributed protocols for key generation and for decryption. In the key generation protocol, the secret key is (verifiably) shared among a set of parties, and the corresponding public key is published. In the decryption protocol, any arbitrary encryption can be verifiably decrypted, as long as enough of the parties cooperate. The decryption protocol must not reveal any information that could compromise the secrecy of other encryptions. Note that every encryption scheme can be turned into a threshold variant by applying techniques of general multi-party computations, but such an approach would be rather inefficient.

5.2.2 Σ-Proofs

A *Σ-proof* is a special kind of protocol between a prover and a verifier, allowing the prover to prove knowledge of a witness satisfying some public predicate, without giving away the witness. The notion of Σ-proofs originates from the notion of a Σ-protocol, as introduced by Cramer [Cra96].

More specifically, a Σ-proof for a public predicate Q is a three-move protocol (where the prover sends the first message), with the property that whenever the verifier accepts the outcome of the protocol, then indeed the prover "knows" a witness ξ satisfying Q, and whenever the verifier selects the challenge c independently from the first message of the prover, then he does not learn anything from the protocol execution that he could not have simulated on his own.



At the end of the protocol, the verifier accepts or rejects the proof. Intuitively, the verifier should accept the proof if and only if the prover knows a witness satisfying the public predicate Q, but the protocol should not give any information about the witness to the verifier. For soundness we assume that the verifier selects the challenge c uniformly at random out of some specified space, and the property that the verifier does not learn anything about the witness is only required to be satisfied for verifiers that select c independently of t. For simplicity, we assume that c is chosen from $\mathbb{Z}_u$ for some integer u.

More formally, the following properties must be satisfied:

Completeness. When both the prover and the verifier are honest, and the prover knows a witness ξ satisfying Q, then the verifier will always accept the conversation.

(Special) soundness. Whenever an honest verifier accepts a conversation, then with overwhelming probability, the (honest or malicious) prover "knows" a witness ξ' satisfying Q. *Special soundness* means that if a prover can reply on two different challenges for any first message t, then he can compute ξ. More formally, we require the existence of an efficient algorithm X (called *knowledge extractor*) that computes a witness ξ' satisfying Q for any given two accepting conversations (t_1, c_1, s_1) and (t_2, c_2, s_2) with $t_1 = t_2$ and $c_1 \neq c_2$.

Special honest-verifier zero knowledge. Zero-knowledge means that the verifier learns nothing from the protocol execution what he could not generate on his own. Honest-verifier zero-knowledge means zero-knowledge with an honest verifier (who selects the

challenge c independently from the first message t), but potentially leaking information to a malicious verifier. Formally, a protocol is special honest-verifier zero-knowledge if there exists a *simulator* S which on input *any* challenge c outputs an accepting conversation (t, c, s), where the conversations are simulated with the same probability distribution as the conversations of the honest verifier with the honest prover.

Completeness implies that a conversation between an honest prover and an honest verifier is always accepted. Special soundness implies that when the challenge space $\mathbb{Z}_u$ is large enough (i.e., $1/u$ is negligible), then a malicious prover (not knowing a witness satisfying Q) cannot convince an honest verifier with non-negligible probability.

Formally, a Σ-proof is represented as a tuple (π_t, u, π_s, ϕ), where π_t is an efficient algorithm that computes the first message t of the protocol for any witness ξ and random string r, u specifies the space $\mathbb{Z}_u$ from which the verifier (uniformly) selects the challenge c, π_s is an algorithm that computes the third message s of the protocol for any given witness ξ, challenge c, and randomness r, and ϕ is the predicate that decides whether a conversation (t, c, s) is to be accepted (i.e., $\phi(t, c, s)$ is satisfied) or to be rejected. A (correct) protocol execution hence looks as follows:

Prover		**Verifier**
knows Q and ξ s.t. $Q(\xi)$		knows Q
select r at random, $t \leftarrow \pi_t(\xi, r)$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_u$
$s \leftarrow \pi_s(\xi, c, r)$	$\xrightarrow{s}$	$\phi(t, c, s)$?

A tuple (π_t, u, π_s, ϕ) is a Σ-proof for predicate Q if there exists an (efficient) knowledge extractor X and an (efficient) simulator S such that the following holds:

- (Completeness) For every witness ξ with $Q(\xi)$, for every random string r, and for every challenge $c \in \mathbb{Z}_u$: $\phi(\pi_t(\xi, r), c, \pi_s(\xi, c, r))$.

- (Special soundness) For any two conversations (t_1, c_1, s_1) and (t_2, c_2, s_2) with $\phi(t_1, c_1, s_1)$, $\phi(t_2, c_2, s_2)$, $t_1 = t_2$, and $c_1 \neq c_2$, the predicate $Q\big(X((t_1, c_1, s_1), (t_2, c_2, s_2))\big)$ is satisfied.
- (Special honest-verifier zero-knowledge) For every witness ξ with $Q(\xi)$ and for every challenge $c \in \mathbb{Z}_u$, the probability distribution of the real conversations $(\pi_t(\xi, r), c, \pi_s(\xi, c, r))$ for uniformly chosen r is the same as the probability distribution of $S(c, r')$ for uniformly chosen r'.

5.2.3 Σ-Transforms

We are mainly interested in two transformations on Σ-proofs, namely AND- and OR-combinations. The AND-combination of two Σ-proofs is a Σ-proof of knowledge of two (usually different) witnesses, satisfying the predicate of the first, respectively of the second, Σ-proof. The OR-combination of two Σ-proofs is a Σ-proof of knowledge of one witness, satisfying the predicate of the first or the predicate of the second Σ-proof. General techniques for combining zero-knowledge protocols can be found in [CDS94].

More formally, assume that we are given a Σ-proof of a witness ξ satisfying Q, and a Σ-proof of a witness ξ' satisfying Q'. The AND-combination is a Σ-proof of a witness (ξ, ξ') satisfying $Q(\xi) \wedge Q'(\xi')$, and the OR-combination is a Σ-proof of a witness ξ^* satisfying $Q(\xi^*) \vee Q'(\xi^*)$.

5.2.3.1 Reduction of the challenge space

As an auxiliary transformation, we need to reduce the challenge space of a given Σ-proof: Consider a Σ-proof (π_t, u, π_s, ϕ) for predicate Q, then for any $u' < u$, the tuple (π_t, u', π_s, ϕ) is a Σ-proof of knowledge of a witness satisfying Q, where the error probability of the new proof is negligible if $1/u'$ is negligible. The security of the new proof (completeness, special soundness, special honest-verifier zero-knowledge) follow immediately from the security of the original proof.

5.2.3.2 AND-combinations

Given a Σ-proof (π_t, u, π_s, ϕ) of knowledge of a witness ξ satisfying the predicate Q, and a Σ-proof $(\pi'_t, u', \pi'_s, \phi')$ of knowledge of a witness ξ'

satisfying the predicate Q', a Σ-proof $(\pi_t^*, u^*, \pi_s^*, \phi^*)$ of a witness (ξ, ξ') satisfying the predicate Q^* with $Q^*(\xi, \xi') \Leftrightarrow Q(\xi) \wedge Q'(\xi')$ can be constructed as follows: First, we set $u^* = \min(u, u')$ and restrict the challenge space of the Σ-proofs for Q and Q' to $\mathbb{Z}_{u^*}$. Then, the protocol for proving knowledge of a witness satisfying Q^* consists of two parallel executions of a protocol, once for proving knowledge of a witness satisfying Q, and once for proving knowledge of a witness satisfying Q', where the same challenge is used for both protocols. The protocol is formally depicted below:

Prover		**Verifier**
knows Q, Q', ξ, ξ' with $Q(\xi) \wedge Q'(\xi')$		knows Q, Q'
select r, r' at random, $t \leftarrow \pi_t(\xi, r)$, $t' \leftarrow \pi_t'(\xi', r')$,	$\xrightarrow{t, t'}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_{u^*}$
$s \leftarrow \pi_s(\xi, c, r)$, $s' \leftarrow \pi_s'(\xi', c, r')$	$\xrightarrow{s, s'}$	$\phi(t, c, s) \wedge \phi'(t', c, s')$?

Both knowledge extractor and simulator for the new protocol can be easily constructed from the knowledge extractors and the simulators of the basic protocols.

5.2.3.3 OR-combinations

Finally, we consider the OR-combination of two Σ-protocols: Given a Σ-proof (π_t, u, π_s, ϕ) of knowledge of a witness ξ satisfying the predicate Q, and a Σ-proof $(\pi_t', u', \pi_s', \phi')$ of knowledge of a witness ξ' satisfying the predicate Q', a Σ-proof $(\pi_t^*, u^*, \pi_s^*, \phi^*)$ of knowledge of a witness ξ^* satisfying the predicate Q^* with $Q^*(\xi^*) \Leftrightarrow Q(\xi^*) \vee Q'(\xi^*)$ can be constructed as follows: First, we set $u^* = \min(u, u')$ and restrict the challenge space of the Σ-proofs for Q and Q' to $\mathbb{Z}_{u^*}$. Then, the protocol for proving knowledge of a witness satisfying Q^* consists of two parallel executions of a protocol, once for proving knowledge of a witness satisfying Q, and

once for proving knowledge of a witness satisfying Q', where only one challenge c^* is sent, and the prover is allowed to split c^* into two subchallenges c and c' with $c + c' = c^*$. This allows the prover to use a simulator for one of the protocols (for the predicate which he does not know a witness for), but not for both. Again we give the protocol formally. Without loss of generality, we assume that the prover' knows a witness ξ satisfying Q (but not necessarily a witness ξ' satisfying Q').

Prover		**Verifier**
knows Q, Q', ξ with $Q(\xi)$		knows Q, Q'
select r, r' at random, $c' \in_R Z_{u^*}$ $t \leftarrow \pi_t(\xi, r)$, $(t', c', s') \leftarrow S'(c', r')$,	$\xrightarrow{t, t'}$	
	$\xleftarrow{c^*}$	$c^* \in_R \mathbb{Z}_{u^*}$
$c = c^* - c' \pmod{u^*}$ $s \leftarrow \pi_s(\xi, c, r)$	$\xrightarrow{c, c', s, s'}$	$c^* \stackrel{?}{=} c + c' \pmod{u^*}$ $\phi(t, c, s)$? $\phi'(t', c', s')$?

Completeness of the above Σ-proof follows immediately from inspecting the protocol. In order to prove special soundness, a knowledge extractor X^* must be given. Such an extractor can be constructed easily: Obviously, when the challenge c^* is different for two accepting conversations, then also either c or c' (as sent in the third message) must differ, and we can apply the knowledge extractor of the corresponding Σ-proof to find a witness satisfying Q or Q'. Finally, for proving the special honest-verifier zero-knowledge property, we need to construct a special simulator S^*. Such a simulator can be constructed in a straight-forward manner: For any given challenge $c^* \in \mathbb{Z}_{u^*}$, c and c' are selected at random such that $c + c' = c^*$. Then, the special simulators S and S' are invoked to generate accepting conversations (t, c, s) and (t', c', s'), respectively, which immediately define an accepting conversation of the combined protocol.

5.2.4 Identification Scheme

We assume an identification scheme which allows the voters to prove their identity with a Σ-proof. We briefly review Schnorr's identification protocol [Sch91] as a Σ-proof and show that it satisfies this requirement: Let G be a group of order $|G| = q$, and let g be a generator of this group, i.e., $G = \langle g \rangle$. Each voter selects a secret key $z_v \in \mathbb{Z}_q$ at random, and computes the public key $Z_v = g^{z_v} \pmod{G}$. Knowledge of the secret key (i.e., identity) is proven with the following Σ-proof. Formally, the voter proves knowledge of a witness z satisfying the predicate $Q_{Z_v}(z) \Leftrightarrow (g^z \stackrel{?}{=} Z_v)$. The security of this protocol can be proven with standard techniques.

Prover		**Verifier**
knows Z_v, z_v, s.t. $Z_v = g^{z_v}$		knows Z_v
$r \in_R \mathbb{Z}_q, t = g^r$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$s = cz_v + r \pmod{q}$	$\xrightarrow{s}$	$g^s \stackrel{?}{=} Z_v^c \cdot t$

5.2.5 Ensuring Knowledge of the Secret Key

Furthermore, in a model providing receipt-freeness, it is essential that each voter knows his own secret key. We provide a protocol that ensures a voter's knowledge of his secret key for Schnorr's identification scheme (cf. Section 5.2.4). This protocol should be performed as part of the key registration (in the public-key infrastructure), or alternatively, as part of the voting protocol if the key infrastructure does not provide this property. This protocol requires secure (one-way) untappable channels from the authorities to the voters, as will also be used by the voting protocols themselves. The following protocol is based on Feldman's secret-sharing scheme [Fel87]. It establishes that a voter v knows the secret key z_v corresponding to his public key Z_v (where $g^{z_v} = Z_v$). We stress that this protocol should be performed as part of the key-infrastructure system, and not as part of the voting protocol.

- The voter shares his secret key z_v among the authorities by using Feldman's secret-sharing scheme [Fel87]: The voter v chooses a uniformly distributed random polynomial $f_v(x) = z_v + a_1 x + \ldots + a_{t-1}x^{t-1}$ of degree $t-1$, and secretly sends[23] the share $s_i = f_v(i)$ to authority A_i (for $i = 1, \ldots, N$). Furthermore, the voter commits to the coefficients of the polynomial by sending $c_i = g^{a_i}$ for $i = 1, \ldots, t-1$ to the bulletin board.

- Each authority A_i verifies with the following equation whether the received share s_i indeed lies on the committed polynomial $f_v(\cdot)$:

  $$g^{s_i} \stackrel{?}{=} Z_v \cdot c_1^i \cdot \ldots \cdot c_{t-1}^{i^{t-1}} \quad \left(= g^{z_v} \cdot g^{a_1 i} \cdot \ldots \cdot g^{a_{t-1} i^{t-1}} = g^{f_v(i)}\right).$$

  If an authority detects an error, she complains and the voter is requested to post the corresponding share to the bulletin board. If the posted share does not correspond to the commitments, the voter is disqualified.

- Finally, every authority (which did not complain in the previous stage) sends her share through the untappable channel to the voter, and the voter interpolates the secret key from those shares that are consistent with the committed coefficients.

In the above protocol, clearly after the second step, either the (honest) authorities will have consistent shares of the voter's secret key z_v, or the voter will be disqualified. However, so far it is not ensured that the voter indeed knows the secret key, as the shares could have been provided by the coercer. In any case, in the final step the voter learns z_v. There are at least t honest authorities who either complained (and thus their share is published), or who send their share to the voter, and hence the voter can interpolate the secret key z_v. The secrecy of the voter's secret key z_v is preserved under the assumption that at most $t-1$ authorities are dishonest.

5.2.6 Designated-Verifier Proofs

A designated-verifier proof is a proof which is convincing for one particular (designated) verifier, but completely useless when transferred from

[23] Either the voter encrypts the share with the authority's public key, or alternatively the authority first sends a one-time pad through the untappable channel, and the voter then encrypts with this pad.

this designated verifier to any other entity. The notion of designated-verifier proofs was introduced in [JSI96].

The key idea of designated-verifier proofs is to prove knowledge of either the witness in question, or of the secret key of the designated verifier. Such a proof does convince the designated verifier (who assumes that the prover does not know his secret key), but if the proof is transferred from the verifier to another entity, it loses its persuasiveness completely.

We only consider designated-verifier Σ-proofs. Such a proof is constructed as follows: Let (π_t, u, π_s, ϕ) be a (normal) Σ-proof for the predicate Q to be proven, and let $(\pi'_t, u', \pi'_s, \phi')$ be a Σ-proof for proving identity of the designated verifier, i.e., the Σ-proof of the identification scheme for the verifier in question (where Q' denotes the predicate that the verifier's secret key is to satisfy). Then, a designated-verifier proof for Q is constructed as Σ-proof for $Q \vee Q'$ with the techniques presented in Section 5.2.3.

5.2.7 Non-interactive Proofs

In practice, we will often use non-interactive variants of Σ-proofs. Such non-interactive proofs can be constructed by using the Fiat-Shamir heuristics [FS86], which essentially replaces the challenge of the verifier by a hash value of the first message of the prover. More precisely, for a Σ-proof (π_t, u, π_s, ϕ) for predicate Q, the non-interactive proof is an accepting conversation (t, c, s) satisfying ϕ, where c is computed as the hash value of t, i.e., $c = H(t)$ for an appropriate hash function H. This non-interactive proof is sound in the random oracle model [BR93].

5.3 The Encryption Function

Our voting schemes require a semantically-secure q-invertible homomorphic public-key encryption scheme with threshold key generation and threshold decryption, according to the definitions of Section 5.2.1. Let $E_Z : \mathbb{V} \times \mathbb{R} \to \mathbb{E}, (v, \alpha) \mapsto e$ denote the encryption function for public key Z, and $D_z : \mathbb{E} \to \mathbb{V}, e \mapsto v$ the decryption function for secret key z. The encryption function is required to be q-invertible for a large number q, and we require that there is a number $u \leq q$, large enough that $1/u$ is considered negligible, with the property that all integers smaller than u are co-prime with q, i.e., $\forall u' < u : \gcd(u', q) = 1$. This property will

be used in the knowledge extractors of the Σ-proofs.[24] Obviously, if q is prime, we can set $u = q$.

In the sequel, we give two possible realizations of encryption schemes satisfying the required properties.

5.3.1 ElGamal-Like Encryption

The ElGamal encryption function [ElG84], enhanced with a threshold setup protocol and a threshold group decryption [Ped91], and slightly manipulated to provide additive homomorphism [CGS97], satisfies all above properties. In the following, we give a brief description of this encryption function. More details can be found in [Ped91] and [CGS97]. Note that this encryption scheme provides efficient decryption only for small messages (more precisely, the complexity for decrypting an encryption $e = E(v, \alpha)$ is at most $\mathcal{O}(v)$, respectively $\mathcal{O}(\sqrt{v})$ when using baby-step giant-step). This will be no disadvantage in the voting scheme with ballot shuffling, but will disallow certain types of votes in the scheme with randomizers.

Let G be a commutative group of order $|G| = q$, where q is a large prime, and let g and γ be independently chosen generators of G, i.e., $G = \langle g \rangle = \langle \gamma \rangle$. G can be constructed as a subgroup of $\mathbb{Z}_p^*$, where p is a large prime and $q|p-1$, but can also be obtained from elliptic curves. The secret key z is a random element in $\mathbb{Z}_q$, and the public key is $Z = g^z$. The encryption function is defined as follows:

$$E_Z : \mathbb{Z}_q \times \mathbb{Z}_q \to G \times G, \quad (v, \alpha) \mapsto (g^\alpha, \gamma^v Z^\alpha),$$

i.e., $\mathbb{V} = \mathbb{Z}_q$, $\mathbb{R} = \mathbb{Z}_q$, and $\mathbb{E} = G \times G$. The group operations in $\mathbb{V}$ and in $\mathbb{R}$ are addition modulo q, and the group operation in $\mathbb{E}$ is component-wise multiplication in G. One can easily verify that indeed $(\mathbb{V}, +)$, $(\mathbb{R}, \boxplus)$, and $(\mathbb{E}, \oplus)$ are Abelian groups. Clearly, this encryption function is homomorphic with respect to the given group operations:

$$\begin{aligned} E(v_1, \alpha_1) \oplus E(v_2, \alpha_2) &= (g^{\alpha_1} \cdot g^{\alpha_2}, \gamma^{v_1} Z^{\alpha_1} \cdot \gamma^{v_2} Z^{\alpha_2}) \\ &= (g^{\alpha_1 + \alpha_2}, \gamma^{v_1 + v_2} Z^{\alpha_1 + \alpha_2}) \\ &= E(v_1 + v_2, \alpha_1 + \alpha_2). \end{aligned}$$

[24] More generally, it would be sufficient to assume that for a given large u, there exists an efficiently computable and invertible bijection from $\mathbb{Z}_u$ onto a subset of $\mathbb{Z}_q$, where each element in this subset is co-prime with q.

The semantic security of this encryption function is implied by the well-known decisional Diffie-Hellman assumption (DDHA) in G (see e.g., [TY98]). The DDHA says that for two triples (a_1, b_1, c_1) and (a_2, b_2, c_2) in G^3, where one triple is randomly chosen and the other triples is a Diffie-Hellman-triple (i.e., for some $i \in \{1, 2\}$ there exist $\alpha, \beta \in \mathbb{Z}_q$ such that $a_i = g^\alpha$, $b_i = g^\beta$, and $c_i = g^{\alpha\beta}$), it is impossible for any polynomial-time algorithm to determine which triple is the Diffie-Hellman triple with probability significantly more than 0.5. This implies that Z^α (in the encryption function) could be replaced by a random value without changing significantly the attacker's probability of success. However, replacing Z^α by a random value hides the vote v information-theoretically.

Finally, the q-invertibility is obvious, because $|G| = q$: Any element in G (except the neutral element) has order q, and hence qe is an encryption of 0 with randomness 0 for any $e \in \mathbb{E}$. As q is prime, we can set $u = q$, and the condition $\forall u' < u : \gcd(u', q) = 1$ is satisfied.

A threshold variant of this cryptosystem can be achieved as follows [Ped91, CGS97]: In the setup-protocol, the key pair (z, Z) is constructed in a way that each authority obtains a share z_i of z in a (t, N)-threshold (Shamir) secret-sharing scheme and is publicly committed to this share by $Z_i = g^{z_i}$ [Ped91, CGS97]. In order to decrypt the tally T from the sum encryption $e = (x, y)$ the authorities first jointly compute and reveal $\hat{x} = x^z$ and prove its correctness. This can be achieved by having every authority A_i compute $\hat{x}_i = x^{z_i}$, where z_i is A_i's share of the secret key z, and then compute $\hat{x}$ from $\hat{x}_i$. This is possible if at least t authorities reveal the correct $\hat{x}_i$. More details can be found in [Ped91, CGS97]. Once $\hat{x}$ is known, one can compute

$$\frac{y}{\hat{x}} = \frac{\gamma^T \cdot Z^\alpha}{(g^\alpha)^z} = \gamma^T \ .$$

Then, the authorities must find T. The computation complexity of this task depends on the encoding of the votes and the number of voters (respectively on the size of T), and is discussed later.

5.3.2 Paillier Encryption

The probabilistic encryption function of Paillier [Pai99] is homomorphic with respect to addition. A threshold variant was proposed in [FPS00]

and [DJ01]. In the following we sketch the suitability of this encryption function. Details can be found in [FPS00] and [DJ01].

Let $n = pq$ be an RSA modulus, and let $g \in \mathbb{Z}^*_{n^2}$ be an element with order a multiple of n. The secret key of this scheme is (p, q), and the public key is (n, g). The encryption function is defined as follows:

$$E : \mathbb{Z}_n \times \mathbb{Z}^*_n \to \mathbb{Z}^*_{n^2}, \quad (v, \alpha) \mapsto (g^v \cdot \alpha^n) \pmod{n^2},$$

i.e., $\mathbb{V} = \mathbb{Z}_n$, $\mathbb{R} = \mathbb{Z}^*_n$, and $\mathbb{E} = \mathbb{Z}^*_{n^2}$. The group operation in $\mathbb{V}$ is addition modulo n, the operation in $\mathbb{R}$ is multiplication modulo n, and that in $\mathbb{E}$ is multiplication modulo n^2. The homomorphic property of the scheme can be verified easily:

$$\begin{aligned} E(v_1, \alpha_1) \oplus E(v_2, \alpha_2) &= g^{v_1}\alpha_1^n \cdot g^{v_2}\alpha_2^n \\ &= g^{v_1+v_2} \cdot (\alpha_1\alpha_2)^n \\ &= E(v_1 + v_2, \alpha_1\alpha_2). \end{aligned}$$

This encryption function is semantically secure if (and only if) the Composite Residuosity Assumption holds, i.e., if it is computationally hard to distinguish higher-degree residues. Details can be found in [Pai99].

Furthermore, this encryption function is n-invertible: For any encryption $e = g^v\alpha^n$, the decryption of ne is 0 with randomness e:

$$ne = nE(v, \alpha) = (g^v\alpha^n)^n = g^0 \cdot (g^v\alpha^n)^n = E(0, g^v\alpha^n) = E(0, e).$$

In order to give a u with $\forall u' < u : \gcd(u', n) = 1$, we specify a lower bound on the size of each prime factor of n, and let u be below this bound. For example, we let n be the product of two primes of 512 bits each, and set $u = 2^{511}$.

A threshold-secure setup protocol and a group-decryption protocol along the lines of [Sho00] were proposed in [FPS00] and [DJ01]. These protocols are rather complex (though polynomial in the number of authorities) and are out of the scope of this thesis.

5.4 Re-encrypting and Proving Re-encryptions

A random re-encryption e' of a given encryption e is an encryption with the same vote v, but a new (independently chosen) randomness α. Such

a re-encryption can be computed by adding a random encryption of 0 to e. Formally, a witness $\xi \in_R \mathbb{R}$ is chosen at random, and $e' = e \oplus E(0, \xi)$, i.e.,

$$e' = R(e, \xi) = e \oplus E(0, \xi).$$

Due to the homomorphic property of E, the randomness in e' is uniformly distributed over $\mathbb{R}$ for a uniformly chosen $\xi \in_R \mathbb{R}$.

Proving that a given e' is indeed a re-encryption of e can easily be done by proving that $e' \ominus e$ is an encryption of 0. We present a slightly more general protocol, namely a Σ-proof which allows the prover to prove knowledge of a witness ξ such that $e = E(v, \xi)$ for any given encryption e and vote v. The challenge for the protocol is uniformly selected from $\mathbb{Z}_u$, and the soundness of the protocol is proven under the assumption that E is q-invertible and that $\forall u' < u : \gcd(u', q) = 1$.

Prover		**Verifier**
knows e, v, ξ, s.t. $e = E(v, \xi)$		knows e, v
$\alpha \in_R \mathbb{R}$, $e' = E(0, \alpha)$	$\xrightarrow{e'}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_u$
$\beta = c\xi \boxplus \alpha$	$\xrightarrow{\beta}$	$E(cv, \beta) \stackrel{?}{=} ce \oplus e'$

Completeness of the protocol is obvious by inspection. We next show that the protocol satisfies special soundness, by showing that if for any e' the prover can reply to two different challenges $c_1 \neq c_2$, then he can compute a witness ξ with $e = E(v, \xi)$. So assume that for two different challenges c_1 and c_2, the prover can answer with β_1 and β_2, respectively, such that both conversations (e', c_1, β_1) and (e', c_2, β_2) are accepting, i.e., $E(c_1 v, \beta_1) = c_1 e \oplus e'$ and $E(c_2 v, \beta_2) = c_2 e \oplus e'$, and hence $E\big((c_1 - c_2)v, \beta_1 \boxminus \beta_2\big) = (c_1 - c_2)e$. Without loss of generality assume that $c_1 > c_2$, hence $0 < c_1 - c_2 < u$, and $\gcd(c_1 - c_2, q) = 1$. Hence we can apply the extended Euclidean algorithm to find two integers a and b such that $a(c_1 - c_2) + bq = 1$. Then, due to the q-invertibility of the encryption function, we can

compute (v_q, α_q) such that $eq = E(v_q, \alpha_q)$. This results in

$$\begin{aligned} e &= \big(a(c_1 - c_2) + bq\big)e \\ &= a(c_1 - c_2)e \oplus bqe \\ &= aE\big((c_1 - c_2)v, \beta_1 \boxminus \beta_2\big) \oplus bE(v_q, \alpha_q) \\ &= E\big(a(c_1 - c_2)v + bv_q, a(\beta_1 \boxminus \beta_2) \boxplus b\alpha_q\big), \end{aligned}$$

and due to the homomorphic property of E we have $qv = v_q$ in $\mathbb{V}$, and hence

$$\begin{aligned} e &= E\big(a(c_1 - c_2)v + bqv, a(\beta_1 \boxminus \beta_2) \boxplus b\alpha_q\big) \\ &= E\big(v, a(\beta_1 \boxminus \beta_2) \boxplus b\alpha_q\big) \end{aligned}$$

This concludes that indeed e encrypts v with witness $\xi = a(\beta_1 \boxminus \beta_2) \boxplus b\alpha_q$.

Finally, we show that the protocol is special honest-verifier zero-knowledge by constructing a simulator. The simulator is constructed as follows: For any given $c \in \mathbb{Z}_u$, we select β from $\mathbb{R}$ at random, and set $e' = E(cv, \beta) \ominus ce$. Obviously, the probability distribution of β is the same as the distribution of a real conversation in which α is chosen uniformly distributed (for the same challenge c).

It is important to note that the simulator can also be applied with parameters e and v where e is not an encryption of v, and the simulated conversation is computationally indistinguishable from a conversation with compatible e and v (an efficient distinguisher of these conversations would contradict the semantic security of the encryption function). This indistinguishability is important when several re-encryption proofs are OR-combined with the techniques of Section 5.2.3.

5.5 Voting Protocol based on Ballot Shuffling

In this section, we construct a voting protocol based on the assumed encryption function, where receipt-freeness is achieved with *ballot shuffling*. The idea behind ballot shuffling is that for each voter, the authorities generate in a mix-net-like construction an encrypted ballot of the desired vote, under the guidance of the voter. The voter can select the vote that will be encrypted in the final ballot, but has no influence on the randomness in the encryption.

5.5.1 Model

We assume (unauthenticated) untappable two-way channels between each authority and each voter. Furthermore, a bulletin board is assumed. All communication channels are synchronous.

A threshold t denotes the number of authorities that must remain honest during the whole vote. The outcome of the vote is guaranteed to be correct as long as at least t authorities honestly cooperate in computing the tally, and the privacy of every vote is ensured as long as no t bad authorities collaborate. Receipt-freeness is ensured under the assumption that every voter indeed knows the secret key corresponding to his public key, and that no authority collaborates with the coercer or the vote-buyer. In some cases, such a collaboration can be tolerated; see Section 5.5.7 for details.

5.5.2 Ballots

In this voting scheme, a ballot is represented as a list $[v_1, \ldots, v_L]$ of L sub-ballots (where L denotes the number of candidates), where the i-th sub-ballot is 1 if the i-th candidate is elected, and 0 otherwise.[25] A ballot is valid if all sub-ballots are either 0 or 1, and the sum of all of them is equal to K. Every voter casts one encrypted ballot $[e_1, \ldots, e_L]$, where the encryption is performed component-wise by applying the homomorphic encryption function to each sub-ballot, i.e., $[e_1, \ldots, e_L] = [E(v_1, \alpha_1), \ldots, E(v_L, \alpha_L)]$ for arbitrary random elements $\alpha_1, \ldots, \alpha_L$. Obviously, the required properties of the encryption function (in particular the homomorphic property and the semantic security) are also satisfied for this encryption function on ballots.

5.5.3 Set-up

In the set-up phase, the authorities jointly generate a uniformly distributed secret key and the corresponding public key for the encryption scheme, where the secret key is shared among the authorities, and the public key is publicly known. A protocol for (verifiably) generating a sharing of a randomly chosen secret key and a public key is a requirement on the encryption function.

[25] In fact, any group element can be used to represent a vote for a candidate. However, for unique decoding, the order of the element should be larger than the number of entitled voters.

5.5.4 Casting a Ballot

The basic idea of this protocol is illustrated in Fig. 5.1: First, an encrypted default ballot is constructed, where the first K sub-ballots are encryptions of 1, and the other sub-ballots are encryptions of 0. The encryptions are deterministic, e.g., the randomness in the encryption function is set to 0 (on the very left in the figure). Then, the voter sends a permutation to the first authority, who then picks this encrypted default ballot, re-encrypts each sub-ballot (illustrated by rotating the entries), reorders them according to the voter's permutation, and hands the resulting encrypted ballot to the second authority. Furthermore, the authority proves publicly that the new encrypted ballot indeed contains the same entries as the original ballot, and proves secretly in designated-verifier manner (through a secure untappable channel) to the voter that she used the requested permutation. Then, the voter sends a permutation to the second authority, who picks the encrypted ballot (from the first authority), shuffles it according to the voter's request, and provides the public and the private proofs. These steps are repeated t times (then, any less than t authorities cannot trace all permutations).

If an authority is malicious, she might fail the proof that she used the requested permutation. Because this proof is sent over an untappable channel, the voter cannot prove that the authority failed. Hence, we allow the voter reject up to $N - t$ of these proofs, i.e., the proofs of all but t authorities. The ballot is finished as soon as the voter has accepted t shufflings.

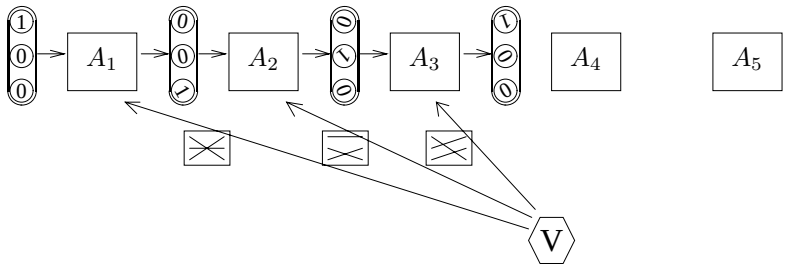


Figure 5.1: Constructing a vote in a 1-out-of-3 voting scheme with N=5 authorities and threshold $t = 3$. When there is no dispute between the voter and an authority, the ballot is finished after the third authority.

The protocol is shown more formally in the following. The essential sub-protocols, namely a (public) proof that a shuffling of an encrypted ballot indeed contains the same sub-ballots, and a (designated-verifier) proof that the shuffling is according to the requested permutation will be constructed afterwards.

We start with an initial ballot that contains encryptions of 1 in the first K sub-ballots, and encryptions of 0 in the other sub-ballots. All encryptions are deterministic (i.e., with randomness 0), and hence this initial ballot is implicitly defined and public:

$$[e_1^{(0)}, \ldots, e_L^{(0)}] = [\underbrace{E(1,0), \ldots, E(1,0)}_{K \text{ times}}, \underbrace{E(0,0), \ldots, E(0,0)}_{L-K \text{ times}}]$$

The voter selects t random permutations $\pi_i : \{1, \ldots, L\} \to \{1, \ldots, L\}$ (for $i = 1, \ldots, t$), such that $\pi_1 \circ \cdots \circ \pi_t$ results in the desired permutation. Then, the following steps are performed in turn for each authority A_k, till the voter has accepted t shufflings.

1. The voter picks the next uncrossed permutation π and sends it over an untappable channel to A_k. If the channels is not authentic, the voter must sign π.

2. A_k picks the encrypted ballot $[e_1^{(k-1)}, \ldots, e_L^{(k-1)}]$ (for the first authority A_1, this is the initial ballot with the 1-votes in the first K sub-ballots, and for all succeeding authorities, this is the ballot of the previous authority). Then the authority shuffles this encrypted ballot, and hands it to the next authority. To shuffle the ballot means to re-encrypt each encrypted sub-ballot and to permute the ordering of the sub-ballots according to the requested permutation. More precisely, the authority selects L random witnesses $\xi_1, \ldots, \xi_L \in_R \mathbb{R}^L$, and assigns $e_i^{(k)} = R(e_{\pi(i)}^{(k-1)}, \xi_i)$ for $i = 1, \ldots, L$. The new ballot $[e_1^{(k)}, \ldots, e_L^{(k)}]$ is posted to the bulletin board.

3. A_k constructs a (non-interactive) proof that the new ballot is valid, under the assumption that the previous ballot was valid, and posts it to the bulletin board. The construction of this proof is given in Section 5.5.4.1.

4. A_k secretly conveys to the voter a designated-verifier proof that she used the requested permutation π through the untappable channel. The construction of this proof is given in Section 5.5.4.2.

5. The voter publicly announces whether he accepts or rejects the proof. If he accepts, then he crosses the used permutation, and the protocol goes on with the next authority and the next permutation. If the voter rejects the proof, we set $[e_1^{(k)}, \ldots, e_L^{(k)}] = [e_1^{(k-1)}, \ldots, e_L^{(k-1)}]$, i.e., the shuffling of this authority is ignored, and the protocol goes on with the next authority, but the same permutation.

The ballot after the t-th accepted shuffling is the ballot of the voter. If the voter does not accept t proofs, then he must be dishonest, and he cannot cast a vote (see analysis in Section 5.5.7).

5.5.4.1 Proof of correct shuffling

We denote the permutation used for re-ordering the sub-ballots by π, and the witnesses used for re-encrypting the sub-ballots by $\xi_1, \ldots, \xi_L$, i.e., $e_i^{(k)} = R(e_{\pi(i)}^{(k-1)}, \xi_i)$ for $i = 1, \ldots, L$. The proof of correct shuffling consists of the following two steps:

1. Proof that for every $i = 1, \ldots, L$, $e_i^{(k)}$ is a re-encryption of *some* sub-ballot $e_1^{(k-1)}, \ldots, e_L^{(k-1)}$, and
2. Proof that the shuffled sum $e_\Sigma^{(k)} = \sum_{i=1}^{L} e_i^{(k)}$ is a re-encryption of the sum $e_\Sigma^{(k-1)} = \sum_{i=1}^{L} e_i^{(k-1)}$.

The first step ensures that the new ballot contains only encryptions of 0 or 1, and the second step ensures that the number of encryptions of 1 is the same in old and the new ballot. Both steps together ensure that the new ballot is a shuffling of the old ballot.

The first step of the proof can be formalized as follows: Let $Q_{ij}(\xi)$ denote the predicate that $e_i^{(k)}$ is a re-encryption of $e_j^{(k-1)}$ with witness ξ, i.e., $Q_{ij}(\xi) \Leftrightarrow \left(e_i^{(k)} \stackrel{?}{=} R(e_j^{(k-1)}, \xi)\right)$. Then the authority must prove knowledge of a witness $(\xi_1, \ldots, \xi_L)$ satisfying the predicate Q, where

$$Q(\xi_1, \ldots, \xi_L) \quad \Leftrightarrow \quad \big(Q_{11}(\xi_1) \vee Q_{12}(\xi_1) \vee \ldots \vee Q_{1L}(\xi_1)\big) \\ \wedge \ldots \wedge \big(Q_{L1}(\xi_L) \vee Q_{L2}(\xi_L) \vee \ldots \vee Q_{LL}(\xi_L)\big).$$

A Σ-proof for this predicate can be constructed in a straight-forward way with the techniques presented in Section 5.2.3, based on the re-encryption

Σ-proof of Section 5.4. For simplicity, we do not show the full Σ-proof, but only a Σ-proof for the first clause (i.e., for proving knowledge of a witness $(\xi_1, \ldots, \xi_L)$ satisfying $Q_{11}(\xi_1) \vee \ldots \vee Q_{1L}(\xi_1)$). A Σ-proof for the full predicate Q is constructed as L parallel instances of the shown partial proof, one for each clause.

Prover		**Verifier**
knows $e_1^{(k-1)}, \ldots, e_L^{(k-1)}, e_1^{(k)}$, i, ξ s.t. $e_1^{(k)} = e_i^{(k-1)} \oplus E(0, \xi)$		knows $e_1^{(k-1)}, \ldots, e_L^{(k-1)}, e_1^{(k)}$
For $j = 1, \ldots, L$: $c_j \in_R \mathbb{Z}_u$, $\beta_j \in_R \mathbb{R}$, $e'_j = E(0, \beta_j) \ominus$ $c_j(e_1^{(k)} \ominus e_j^{(k-1)})$ [Note: $e'_i = E(0, \beta_i - c_i\xi)$]	$\xrightarrow{e'_1, .., e'_L}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_u$
adjust c_i, β_i such that $c = \sum_{j=1}^{L} c_j \pmod u$, $e'_i = E(0, \beta_i - c_i\xi)$	$\xrightarrow{c_1, ., c_L, \beta_1, .., \beta_L}$	$c \stackrel{?}{=} \sum_{j=1}^{L} c_j \pmod u$ For $j = 1, \ldots, L$: $E(0, \beta_j) \stackrel{?}{=}$ $c_j(e_1^{(k)} \ominus e_j^{(k-1)}) \oplus e'_j$

This Σ-proof can easily be made non-interactive by applying the Fiat-Shamir heuristics (see Section 5.2.7). A non-interactive proof is a vector $[c_1, \ldots, c_L, \beta_1, \ldots, \beta_L]$ satisfying

$$c_1 + \ldots + c_L \stackrel{?}{=} H\Big(E(0, \beta_1) \ominus c_j(e_1^{(k)} \ominus e_1^{(k-1)}) \Big\| \ldots \\ \ldots \Big\| E(0, \beta_L) \ominus c_L(e_1^{(k)} \ominus e_L^{(k-1)})\Big),$$

where H denotes a hash function with image $\mathbb{Z}_u$. The full proof of the first step consists of L such vectors (one for each $e_1^{(k)}, \ldots, e_L^{(k)}$).

The purpose of the second step of the proof is to ensure that the sum of all sub-ballots is the same in the original ballot and in the new ballot (namely K). We need the following observation:

$$R(e_1^{(k-1)}, \xi_1) \oplus \ldots \oplus R(e_L^{(k-1)}, \xi_L) = R(e_1^{(k-1)} \oplus \ldots \oplus e_L^{(k-1)}, \xi_\Sigma),$$

where $\xi_\Sigma = \xi_1 \boxplus \ldots \boxplus = \xi_L$, and hence $e_\Sigma^{(k)}$ is a re-encryption of $e_\Sigma^{(k-1)}$ with witness ξ_Σ. A Σ-proof for proving re-encryptions was given in Section 5.4. The non-interactive proof is then the pair $[c, \beta]$ satisfying

$$c \stackrel{?}{=} H\Big(E(0,\beta) \ominus c\big(e_\Sigma^{(k)} \ominus e_\Sigma^{(k-1)}\big)\Big).$$

Altogether, the proof of correct shuffling consists of $(L^2+1)(\log|\mathbb{R}| + \log u)$ bits.

5.5.4.2 Designated-verifier re-encryption proof

The authority must proof to the voter that she permuted the sub-ballots according to the requested permutation π. This is achieved by proving that indeed $e_i^{(k)}$ is a re-encryption of $e_{\pi(i)}^{(k-1)}$ for $i = 1, \ldots, L$. This proof must be performed in designated-verifier manner, such that the voter cannot hand the proof to a vote-buyer or a coercer (see Section 5.2.6). We show the detailed proof with Schnorr's identification scheme, but we stress that the proof can be constructed for any identification scheme for which a Σ-proof for identification exists. As before, $Q_{ij}(\xi)$ denotes the predicate that $e_i^{(k)}$ is a re-encryption of $e_j^{(k-1)}$ with witness ξ, i.e., $Q_{ij}(\xi) \Leftrightarrow \big(e_i^{(k)} \stackrel{?}{=} R(e_1^{(k-1)}, \xi)\big)$. Furthermore, $Q_{Z_v}(z)$ denotes the predicate that z is the secret key that matches the voter's public key Z_v, i.e., $Q_{Z_v}(z) \Leftrightarrow \big(g^z \stackrel{?}{=} Z_v\big)$. We construct a Σ-proof of knowledge of a witness $(\xi_1, \ldots, \xi_L, z)$ satisfying the predicate $\big(Q_{1,\pi(1)}(\xi_1) \wedge Q_{2,\pi(2)}(\xi_2) \wedge \ldots \wedge Q_{L,\pi(L)}(\xi_L)\big) \vee Q_{Z_v}(z)$:

Prover		**Verifier**
knows $e_1^{(k-1)}, \ldots, e_L^{(k-1)}$, $e_1^{(k)}, \ldots, e_L^{(k)}, \pi, \xi_1, \ldots, \xi_L$, s.t. $\forall i : e_i^{(k)} = R(e_{\pi(i)}^{(k-1)}, \xi_i)$		knows $e_1^{(k-1)}, \ldots, e_L^{(k-1)}$, $e_1^{(k)}, \ldots, e_L^{(k)}, \pi$
For $j = 1, \ldots, L$: $\alpha_j \in_R \mathbb{R}, e'_j = E(0, \alpha_j)$ $c_2 \in_R \mathbb{Z}_u, s_2 \in_R \mathbb{Z}_q$, $t_2 = g^{s_2} Z_v^{-c_2}$	$\xrightarrow{e'_1, \ldots, e'_L, t_2}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_u$
$c_1 = c - c_2 \pmod u$ For $j = 1, \ldots, L$: $\beta_j = c_1 \xi_j \boxplus \alpha_j$	$\xrightarrow{c_1, c_2, \beta_1, ., \beta_L, s_2}$	$c \stackrel{?}{=} c_1 + c_2 \pmod u$ For $j = 1, \ldots, L$: $E(0, \beta_j) \stackrel{?}{=} c_1 \big(e_j^{(k)} \ominus e_{\pi(j)}^{(k-1)}\big) \oplus e'_j$ $g^{s_2} \stackrel{?}{=} Z_v^{c_2} \cdot t_2$

Also this Σ-proof is then turned into a non-interactive proof (see Section 5.2.7). A designated-verifier proof that the sent permutation π is correct is a vector $[c_1, c_2, \beta_1, \ldots, \beta_L, s_2]$ satisfying

$$c_1 + c_2 = H\Big(E(0, \beta_1) \ominus c_1\big(e_1^{(k)} \ominus e_{\pi(1)}^{(k-1)}\big) \Big\| \ldots \\ \ldots \Big\| E(0, \beta_L) \ominus c_1\big(e_L^{(k)} \ominus e_{\pi(L)}^{(k-1)}\big) \Big\| g^{s_2} Z_v^{-c_2}\Big).$$

5.5.5 Tallying

Tallying is achieved by adding up the cast ballots component-wise (by using the homomorphic property of the encryption function), and then jointly decrypting each sub-ballot of the sum. The protocol for joint decryption is a required property of the encryption function. Note that each sub-ballot contains a number in the range $[0, \ldots, M]$, and hence decryption of the sub-ballot is efficient even if the decryption function requires $\mathcal{O}(T)$ operations for decrypting the tally T.

5.5.6 Efficiency Analysis

We analyze the communication complexity of the K-out-of-L voting scheme. We assume that there are N authorities and M active (participating) voters. We denote the number of bits used to represent one group element with B, i.e., $|\mathbb{V}| \leq 2^B$, $|\mathbb{R}| \leq 2^B$, and $\mathbb{E} \leq 2^{2B}$.

The costs for initialization of the encryption function and for the computation of the tally are independent of the number M of voters, and are thus ignored in the analysis. The most relevant costs are those related with casting (and proving) votes. The representation of one ballot costs $2LB$ bits, and every authority must post one (shuffled) ballot to the bulletin board ($2LNB$ bits) per voter. Also, the authority must post a proof of correct shuffling, which consists of about $2L^2B$ bits. Additionally, the voter most send a permutation ($L \log L$ bits) and the authority must send a designated-verifier proof ($2LB$ bits) over the untappable channels. Finally, each voter posts the final permutation ($L \log L$ bits) to the bulletin board. This makes a total of about $2L^2MNB$ bits posted to the bulletin board and $3LMNB$ bits sent through the untappable channels.

5.5.7 Security Analysis

5.5.7.1 Correctness

The correctness is based on three facts: First, the ballot generated by the authorities is a valid ballot, i.e., all sub-ballots contain either a 0- or a 1-vote, and there are only K 1-votes. This is ensured by the public proofs of correct shuffling (Sect. 5.5.4.1). Second, the published tally is indeed the sum of the casted votes. This is ensured by the homomorphic property of the encryption function and the verifiable decryption of the encryption scheme. Finally, the voter is able to cast any valid vote, at his free will. This is ensured by the designated-verifier re-encryption proofs of the authorities.

If more than $N - t$ authorities are malicious, then some (or all) voters can be prevented from casting their vote (there might be no t correct proofs that the voter can accept). Furthermore, if more than $N - t$ authorities are malicious, they can reject computing and proving the tally. However, any number of malicious authorities cannot add invalid ballots to the vote, and cannot proof a wrong tally.

5.5.7.2 Privacy

The secrecy of each vote is guaranteed with respect to any set of less than t authorities. The ballot is shuffled t times, and as long as at least one of these authority is honest, the bad authorities cannot track the shuffling of the sub-ballots through the shuffling. A set of t or more authority can open any ballot by using their shares of the secret key, or by tracing the permutations.

5.5.7.3 Receipt-freeness

The voter cannot prove which permutations he sent through the untappable channels. Furthermore, the proofs he received through the untappable channels are designed-verifier, and the voter cannot convince the vote-buyer with these proofs (the voter can easily generate such proofs for any permutation). The only public communication of the voter is his confirmation or denial of the received shuffling proof. The vote-buyer can ask the voter to rejects certain (correct) proofs, but the only results would be that the voter cannot vote (something that anyway the vote-buyer can ask for).

In case that authorities collude with a vote-buyer or a coercer, then apparently receipt-freeness is still ensured as long as each voter knows at least one authority not colluding with the vote-buyer (then the voter can lie for the permutation π used with this authority A_k). If a voter does not know such an authority, he can select one authority at random and lie for this permutation. In the context of vote-buying this means that the voter can forge a receipt for a vote he did not cast, and the vote-buyer accepts such a forged receipt with probability linear in the number of authorities not colluding with him, which seems to be unacceptable for the vote-buyer. However, in the context of coercion, this means that the probability of a lying voter to be caught is linear in the number of authorities colluding with the coercer, and this seems to be unacceptable for the voter.

5.6 Voting Protocol based on Randomizers

In this section we present an alternative approach for receipt-free voting. The proposed protocol is very efficient. However, it relies on an additional assumption, namely the existence of a *randomizer*, whose honesty

is essential for the security of the protocol. The randomizer's task is to re-randomize encrypted ballots of the voters. More precisely, each voter constructs an encrypted ballot containing his vote and secretly sends it to the randomizer. The randomizer re-encrypts this ballot and posts it to the bulletin board. Furthermore, the randomizer proves to the voter (in designated-verifier manner) that indeed the new encrypted ballot contains the same vote, and the voter and the randomizer jointly generate a proof of validity for this new ballot.

5.6.1 Model

In this voting scheme, two-way untappable channels between every voter and the randomizer are assumed. Furthermore, we make use of a bulletin board. A threshold t denotes the number of authorities that is required for decrypting the tally, and which also is able to annihilate the secrecy of any vote. Collaboration of the randomizer with a voter-buyer or coercer cannot be tolerated. The randomizer does not learn the vote of any voter, but he can reject to re-encrypt the ballot of any voter and thereby prevent this voter from participating the vote. This problem can be overcome with using several randomizers; this will be discussed at the end of this section.

5.6.2 Ballots

In this scheme, a ballot is represented as a single vote v. In order to allow decoding of the sum, a special encoding of the votes is applied: A vote for the i-th candidate (where $1 \leq i \leq L$) is represented as M^{i-1}. More generally, the vote is set as follows:

$$v = \sum_{\text{vote for candidate } i} M^{i-1}.$$

This encoding allows easy decoding of the sum vote by simple remaindering. The set $\mathcal{V}$ of valid votes in a K-out-of-L scheme is defined as follows:

$$\mathcal{V} = \left\{ M^{I_1} + \ldots + M^{I_k} \;\middle\|\; k \leq K, 0 \leq I_1 < \ldots < I_k < L \right\}.$$

Note that this encoding requires that the decryption sub-protocol allows efficient decryption of arbitrary ciphertexts. The encryption function based on ElGamal (Section 5.3.1) does not satisfy this requirement.

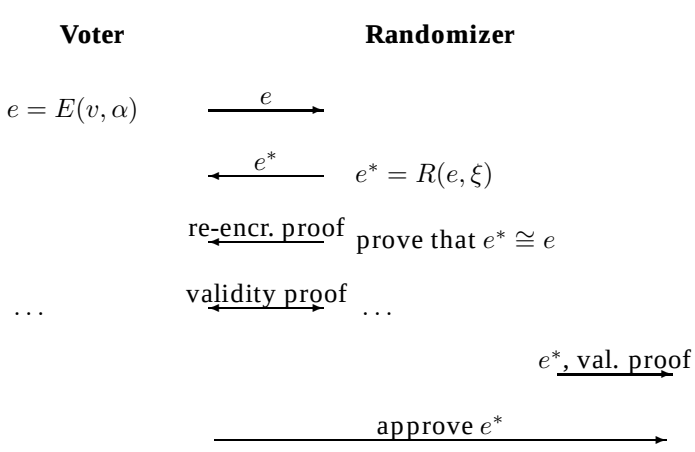
In order to decrypt a tally in a 1-out-of-L vote, computation complexity of $\mathcal{O}(\sqrt{M}^{L-1})$ is required [CGS97]. For large L, this effort may not be acceptable, and the alternative encryption scheme of Paillier (Section 5.3.2) is recommended.

5.6.3 Set-up

The set-up phase is identical to the set-up phase of the previous scheme.

5.6.4 Casting a Ballot

A ballot is cast as follows: The voter constructs a random encryption $e = E(v, \alpha)$ of the encoded vote v, and sends it through the untappable channel to the randomizer. The randomizer then computes a random re-encryption $e^* = R(e, \xi)$ of e, and proves to the voter in designated-verifier manner that indeed e^* is a re-encryption of e. Then, the voter and the randomizer jointly generate a validity proof for e^*, without the randomizer learning anything about the vote v, and without the voter learning anything about the re-encryption witness ξ. Finally, the randomizer posts the re-encrypted ballot e^* and the validity proof to the bulletin board, and the voter posts an approval message indicating that indeed e^* is his vote (i.e., he accepted the re-encryption proof of the randomizer).



5.6.4.1 Designated-verifier re-encryption proof

The purpose of this proof is to have the randomizer prove to the voter that the new encryption e^* is indeed a re-encryption of e. However, this proof must be non-transferable, such that the verifier cannot convince someone else that e^* is a re-encryption of e. The technique used to achieve this non-transferability is explained in Section 5.2.6: The randomizer proves knowledge of either a re-randomization witness ξ with $e^* = R(e, \xi)$, or of the voter's secret key. Obviously, this proof is convincing for the voter, but completely useless when transfered from the voter to a third party.

The proof is constructed as an OR-combination of the Σ-proof that the encryption $e^* \ominus e$ contains the vote 0, and the Σ-proof of the identification scheme. We show the proof for Schnorr's identification scheme. We denote the voter's secret key with z_v and the public key with $Z_v = g^{z_v}$. The designated-verifier re-encryption proof is constructed as follows:

Randomizer		**Voter**
knows $e, \xi, e^* = e + E(0, \xi)$		knows e, e^*
$\alpha \in_R \mathbb{R},\ e' = E(0, \alpha)$ $c_2 \in_R \mathbb{Z}_u,\ s_2 \in_R \mathbb{Z}_q,$ $t_2 = g^{s_2} Z_v^{-c_2}$	$\xrightarrow{e', t_2}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_u$
$c_1 = c - c_2 \pmod u$ $\beta = c_1 \xi \boxplus \alpha$	$\xrightarrow{c_1, c_2, \beta, s_2}$	$c \stackrel{?}{=} c_1 + c_2 \pmod u$ $E(0, \beta) \stackrel{?}{=} c_1(e^* \ominus e) \oplus e'$ $g^{s_2} \stackrel{?}{=} Z_v^{c_2} \cdot t_2$

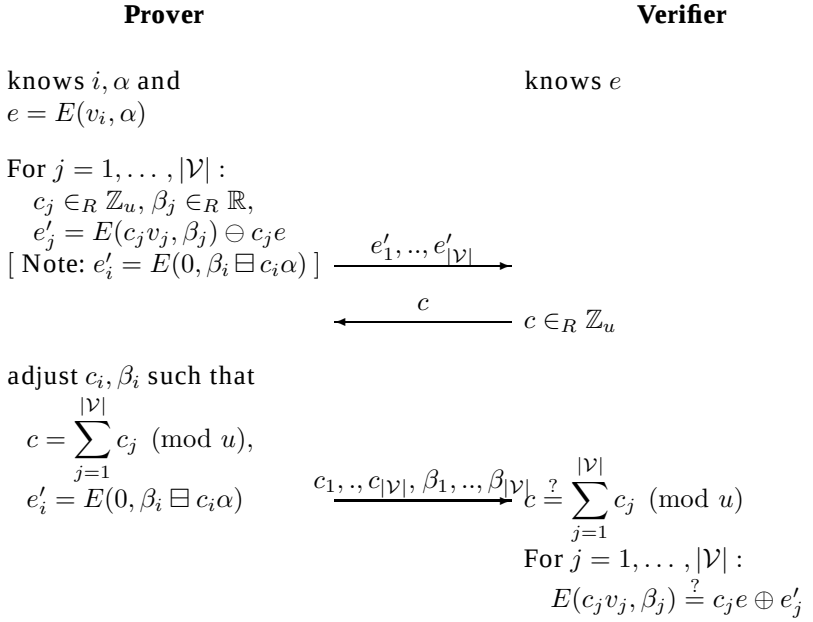
A non-interactive version of the proof is the vector $[c_1, c_2, \beta, s_2]$ satisfying the equation

$$c_1 + c_2 = H\Big(E(0, \beta) \ominus c_1\big(e^* \ominus e\big) \Big\| g^{s_2} Z_v^{-c_2}\Big).$$

5.6.4.2 Validity proof

The validity proof is a non-interactive proof that the randomized encryption e^* contains a vote from the set $\mathcal{V}$ of valid votes. The proof is generated in an interactive protocol between the voter and the randomizer. For didactic reasons, we first present a Σ-proof that a given encryption contains a vote from a given set, then we apply the Fiat-Shamir heuristics to this protocol and obtain a non-interactive variant. Finally, we present a protocol which allows the randomizer and the voter to generate such a non-interactive proof for a re-encrypted vote, where the randomizer learns nothing about the vote, and the voter learns nothing about the randomization witness.

The Σ-proof that a given encryption e contains some vote out of a set $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is an OR-combined proof of knowledge of a witness α such that either $e = E(v_1, \alpha)$ or $e = E(v_2, \alpha)$ or $\ldots$ (cf. Section 5.2.3).

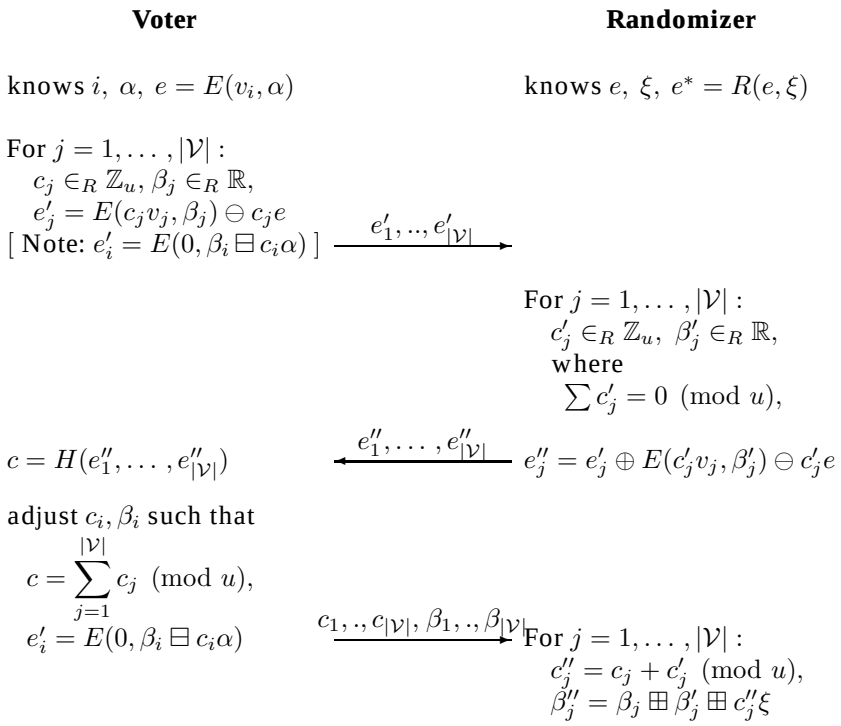


The non-interactive proof is obtained by applying the Fiat-Shamir heuristics to the above protocol. A non-interactive proof of vote validity

is a vector $[c_1, \ldots, c_{|\mathcal{V}|}, \beta_1, \ldots, \beta_{|\mathcal{V}|}]$ where the following equation holds:

$$c_1 + \ldots + c_{|\mathcal{V}|} \stackrel{?}{=} H\Big(E(c_1 v_1, \beta_1) \ominus c_1 e \Big\| \ldots \Big\| E(c_{|\mathcal{V}|} v_{|\mathcal{V}|}, \beta_{|\mathcal{V}|}) \ominus c_{|\mathcal{V}|} e\Big).$$

We observe that the above validity proof for encryption e can easily be converted into a validity proof for a re-encryption $e^* = R(e, \xi) = e + E(0, \xi)$: $[c_1, \ldots, c_{|\mathcal{V}|}, \beta_1 + c_1\xi, \ldots, \beta_{|\mathcal{V}|} + c_{|\mathcal{V}|}\xi]$. However, if the randomizer would publish this adjusted proof, then the voter could easily compute the re-encryption witness ξ and prove his vote. What we need is an *independent* proof of validity of e^*, i.e., the voter must not be able to proof that the resulting validity proof for e^* has any relation to the original vote e. Such an independent proof can be constructed interactively: The randomizer randomizes the first message of the voter, and asks as challenge the hash value of the randomized first message (such that he can construct a non-interactive proof). However, it would not be safe to let the randomizer ask any challenge (the protocol for proving validity of an encryption is only *honest-verifier* zero-knowledge, so the randomizer could potentially learn the vote). Therefore, we let the randomizer send the randomized first message as a whole to the voter, who then computes the hash value by himself. This challenge is "honest-verifier" in the random-oracle model. Finally, the voter sends back the correct third message, and the randomizer adjusts it such that it matches the randomized protocol.



Note that in this protocol, the randomizer does not need to verify the third message of the voter — if the voter sends a bad third message, the generated proof of validity of e^* will be bad, and the voter's vote is ignored in tallying. However, we need that all messages are authentic, and that the messages from the voter to the randomizer are untappable for the voter. This can be achieved by having the voter digitally sign the first and the third message. Note that these signatures give the power to the randomizer (but not to the voter) of proving the received messages. In our context, this doesn't matter, because the randomizer is not allowed to collaborate with a vote-buyer or a coercer.

The proof of validity of e^* is the vector $[c''_1, \ldots, c''_{|\mathcal{V}|}, \beta''_1, \ldots, \beta''_{|\mathcal{V}|}]$. We first argue that this protocol is correct (i.e., the generated vector is indeed a proof of validity of e^*), then show that the proof is zero-knowledge for the randomizer and receipt-free for the voter.

The final proof is accepted exactly if the following equation holds:

$$c''_1 + \ldots + c''_{|\mathcal{V}|} \stackrel{?}{=} H\Big(E(c''_1 v_1, \beta''_1) \ominus c''_j e^* \Big\| \ldots \Big\| E(c''_{|\mathcal{V}|} v_{|\mathcal{V}|}, \beta''_{|\mathcal{V}|}) \ominus c''_{|\mathcal{V}|} e^*\Big).$$

One can easily verify (by inspecting the protocol) that if both the voter and the randomizer are honest, then this equation will hold, and the final proof will be accepted.

In order to show that the protocol is zero-knowledge, we need the fact that the previous protocol (for proving that e is valid) is honest-verifier zero-knowledge. In the new protocol, the only difference for the prover is the second message, where he does not receive a challenge c, but rather a string of which the hash value is used as challenge. Obviously, this ensures that the protocol is zero-knowledge in the random-oracle model.

Finally, in order to show that the protocol is receipt-free, we must show that even a cheating voter cannot prove any correspondence between the proof of validity of e^* (i.e., the vector $[c''_1, \ldots, c''_{|\mathcal{V}|}, \beta''_1, \ldots, \beta''_{|\mathcal{V}|}]$) and the encrypted vote e. Obviously, the $\beta''_1, \ldots, \beta''_{|\mathcal{V}|}$ are completely random and independent from e (because the $\beta'_1, \ldots, \beta'_{|\mathcal{V}|}$ are random). The same holds for the $c''_1, \ldots, c''_{|\mathcal{V}|}$ with the exception that the sum is not random but equal to the sum of $c_1, \ldots, c_{|\mathcal{V}|}$ (because $c'_1 + \ldots + c'_{|\mathcal{V}|} = 0$). However, in the random-oracle model, this sum is assumed to be random and independent of any other values, and hence the vector $c''_1, \ldots, c''_{|\mathcal{V}|}$ is random. It is important to note that the proof $[c''_1, \ldots, c''_{|\mathcal{V}|}, \beta''_1, \ldots, \beta''_{|\mathcal{V}|}]$ is random and independent of the voter's behavior.

5.6.5 Tallying

Tallying is achieved by adding up the cast ballots (by using the homomorphic property of the encryption function), and then jointly decrypting the encrypted sum. The protocol for joint decryption is a required property of the encryption function. From the tally, the sub-tallies for each candidate can be computed by simple arithmetic.

Note that in this protocol, the final tally can be up to M^L, and we must require that the decryption is efficient, whereas the tallying protocol of the other voting scheme (Section 5.5) only requires decryption of small tallies (L partial tallies, each up to M). This makes this protocol inefficient for large L when used with the ElGamal encryption function, and Paillier's encryption function must be used instead.

5.6.6 Efficiency Analysis

We analyze the communication efficiency of this voting protocol for a K-out-of-L scheme. The number of bits used to store one group element is B. We assume that $B \geq L \log M$, i.e., we assume that one vote can be embedded in one encryption.

Again, we ignore the costs for initialization and decryption of the final tally — they are independent of the number M of voters. It remains to count the costs for casting and proving votes. In order to cast his vote, every voter sends the ballot to the randomizer ($2B$ bits), who sends a re-encryption and a re-encryption proof to the voter ($2B + 4B$ bits). Then, the voter and the randomizer run the interactive validity protocol ($6|\mathcal{V}|B$ bits), and the randomizer posts the randomized ballot and the non-interactive proof to the bulletin board ($(2|\mathcal{V}| + 2)B$ bits). This gives a total of $\left(6\binom{L}{K} + 8\right)MB$ bits sent through the untappable channels, and $\left(2\binom{L}{K} + 2\right)MB$ bits sent to the bulletin board. For $K = 1$ (i.e., for 1-out-of-L votes), there are $(6L + 8)MB$ bit sent through the untappable channels and $(2L + 2)MB$ bit posted to the bulletin board.

5.6.7 Security Analysis

The privacy of this scheme is guaranteed under the assumption that no t authorities maliciously pool their information. There are no additional assumption on the honesty of the randomizer. The tally is correct if the randomizer does not exclude entitled voters from participating the vote, and if at least t authorities honestly participate in the tally decryption. The scheme is receipt-free as long as the randomizer does not collaborate with a vote-buyer or a coercer, and as long as it is guaranteed that each voter indeed knows the secret key corresponding to his public key.

5.6.8 Variants

We present two variants of the proposed voting scheme. The goal of the first variant is to reduce the impact of the fact that the randomizer can disable any voter from participating in the vote. The second variant improves the efficiency of the scheme when K is large.

5.6.8.1 Multiple randomizers

The major drawback of this voting scheme is the power of the randomizer. He can disable any voter from casting his vote. One can reduce this problem by introducing several randomizers, where the voter can select one randomizer. In this solution, disabling a voter from participating requires collaboration of every randomizer. However, the cost is that we must assume that no randomizer collaborates with a vote-buyer or a coercer.

5.6.8.2 Alternative encoding of ballots

For large K, the proposed voting scheme becomes very inefficient. The reason for that is that the number of valid votes is $\binom{L}{K}$, which grows exponentially in K for $K < L/2$. For example, for $K = 2$, the number of valid votes is about $L^2/2$, and the size of the validity proof of the ballot is linear in the number of valid votes, hence quadratic in L. Similarly, the size of the validity proof grows cubic in L for $K = 3$.

This super-linear growth of the communication complexity can be avoided by using another encoding of the ballot, namely the encoding that was used in the voting protocol based on ballot shuffling (Section 5.5). There, a ballot is represented as a vector with L elements, where the i-th entry is an encryption of 1 if the i-th candidate is voted for, and 0 otherwise. This encoding makes the ballots bigger, but the validity proof becomes shorter. In order to prove that a ballot (a vector of sub-ballots) is valid, one must prove that every sub-ballot is valid (i.e., an encryption of 0 or 1), and the sum of all sub-ballots is an encryption of K. This proof grows only linearly in L, independently of the size of K. A detailed analysis of the complexity of the modified protocol yields that $(17L + 9)MB$ bits must be sent through the untappable channels, and $(6L + 2)MB$ bits must be posted to the bulletin board. For $K \geq 2$, the communication complexity of this variant is lower than the communication complexity of the original protocol.

This protocol has two other advantages: First, in the original protocol, the size of L is limited by the size of B (the encoding requires that $L \log M \leq B$ holds). As a consequence, B must be increased for large L, which slows down the whole computation. Second, the original protocol requires decryption of an arbitrarily large tally, which makes the protocol unusable with the ElGamal encryption function in many settings. In the modified protocol, L sub-tallies of size each at most M are

decrypted, which can be performed efficiently with both proposed encryption schemes.

5.7 Analysis of the Benaloh-Tuinstra Protocol

The notion of receipt-freeness was first introduced by Benaloh and Tuinstra in [BT94]. They present two protocols that are claimed to be receipt-free. In the single-authority protocol, the authority learns how each vote was cast. This is of course far from satisfactory. In this section, we analyze the receipt-freeness of their protocol with multiple voting authorities and show how a voter can construct a receipt for the vote he casts.

5.7.1 Key Ideas of [BT94]

The basic idea of the multiple-authority protocol [BT94] is to have every voter secret-share his vote among the authorities (using Shamir's secret-sharing scheme [Sha79]), who then add up the shares and interpolate the tally. This idea works due to the linearity of the secret-sharing scheme. There are two major tasks to solve: First, the voter must send one share to each authority in a receipt-free manner, and second, the voter must prove that the secret (the vote) is valid.

We concentrate on the second task: In order to secret-share the vote, the voter selects a random polynomial P of an appropriate degree, such that $P(0) \in \{0,1\}$ is his vote. The share for the j-th authority is hence $P(j)$. Clearly, it is inherently important that the vote is valid, i.e., $P(0) \in \{0,1\}$, since otherwise the tally will be incorrect. Hence, the voter must provide a proof of validity for the cast vote.

For the sake of the proof of validity, the voter wishing to cast a vote v_0 submits (in a receipt-free manner) a bunch of $\ell+1$ vote pairs, where ℓ is a security parameter. That is, the voter submits the votes $(v_0, v_0'), \ldots, (v_\ell, v_\ell')$, and each pair (v_i, v_i') of votes must contain a 0-vote and a 1-vote in random order. For each pair (v_i, v_i') but the first, a coin is tossed and the voter is either asked to open the pair and show that indeed there is a 0-vote and a 1-vote, or he is asked either to prove that $v_i = v_0$ and $v_i' = v_0'$ is satisfied, or to prove that $v_i = v_0'$ and $v_i' = v_0$ is satisfied. This proofs must be performed in public (onto a bulletin board). If the voter passes these tests, then with probability at least $1 - 2^{-\ell}$, v_0 is valid and is accepted as the voters vote.

5.7.2 How to Construct a Receipt

This cut-and-choose proof of validity offers an easy ability for the voter to prove which particular vote he casts: In advance, the voter commits to the ordering of each pair of votes (i.e., he commits to the bit string $v_0, \ldots, v_\ell$). In each round of the cut-and-choose proof, everyone can verify whether the published data is consistent with this commitment. If no inconsistencies are detected while proving the validity of the vote, then with probability at least $1 - 2^{-\ell}$ the voter has chosen the ordering as committed, and also v_0 is as announced.

In order to obtain a receipt, the voter could select an arbitrary string s, and set the string $(v_0, \ldots, v_\ell)$ as the bitwise output of a known cryptographic hash function (e.g. MD5 or SHA) for that string s. Then, s is a receipt of the vote v_0.

5.8 Analysis of the Kim-Lee Protocol

In this section, we show that the protocol of Kim and Lee [LK00] is not receipt-free, opposed to what is claimed in the paper.

5.8.1 Key Ideas of [LK00]

The protocol of [LK00] is based on the assumption of an *honest verifier* who ensures the validity of all cast votes. Each voter sends an encryption e of his vote to this honest verifier and proves its validity. Then, the honest verifier sends a random encryption e' of 0 to the voter and proves (with a three-move honest-verifier zero-knowledge protocol) that indeed e' is an encryption of 0. The final ballot of the voter is $e^* = e + e'$, which obviously contains the same vote as e, but different randomness. All communication between the voter and the randomizer must take place over an untappable channel.

Note that in this protocol a malicious "honest verifier" can help a voter to cast an invalid vote and thereby falsify the outcome of the whole vote. In our opinion, such a protocol in which the correctness of the tally relies on the trustworthiness of a single entity is questionable.

5.8.2 How to Construct a Receipt

The voter can easily construct a receipt: In the protocol where the honest verifier proves to the voter that indeed e' is an encryption of 0, the voter can choose the challenge as the output of a hash function applied to the message in the first move. This makes the transcript of the protocol a non-interactive proof (according to Fiat-Shamir heuristics) that e' is an encryption of 0. Hence, the values e', e, the witness of e, and this proof are a receipt of the cast vote e^*.

Chapter 6

Concluding Remarks

In this thesis, two related problems were studied: The problem of general secure multi-party computation, and as an application of it, the problem of secret-ballot voting. We first briefly summarize our main achievements concerning secure multi-party protocols:

- The classical distinction between the active model and the passive model was abolished. Instead, we have introduced a model in which some of the players are passively corrupted, some other players are actively corrupted, and some more players are fail-corrupted. An exact characterization of how many corruption of each type can be considered was derived for a variety of models. It turned out that in several models, strictly more corruptions can be tolerated with our protocols than with previous protocols.

- The classical characterization of tolerable adversaries, namely by a threshold on the number of players that at most may be corrupted, was generalized. Strictly more general, we have characterized the tolerable adversaries by a so-called adversary structure, a set of potentially corrupted player subsets. We have given necessary and sufficient conditions on this adversary structure for the existence of secure multi-party computation, for various models. It turned out that in many settings, strictly more corruptions can be tolerated than with previous protocols.

- Multi-party protocols were known to be very inefficient when some of the players misbehave. We have proposed a framework for efficient multi-party protocols and have shown a construction of such a protocol which tolerates arbitrary misbehavior of players, and still is almost as efficient as the most efficient protocols that are not robust against misbehavior of players.

The main achievements in the area of secret-ballot voting are as follows:

- A framework for constructing voting protocols based on homomorphic encryption was proposed. This framework allows to construct modular protocols with generic proofs, and contrasts the special-purpose proofs of many voting protocols in the literature.
- Within this framework, we have designed two voting protocols that are receipt-free. Under reasonable circumstances, the first protocol even prevents vote-buying by the authorities themselves. Both protocols are more efficient than any previous receipt-free voting protocol.

The research on this thesis has also brought several problems to light that could not be solved yet. These problems might inspire further research.

- Find and prove an exact characterization of tolerable adversaries in secure multi-party computation for the model with simultaneous active, passive, and fail-corruption for general adversary structures.
- Find more efficient multi-party protocols for non-threshold adversaries. There are (at least) two approaches towards this goal: First, one can apply the proposed framework for efficient protocols to this setting. In principle, this should work, but still many problems must be solved. Second, it would be interesting to find restricted classes of adversary structures, for which more efficient protocols exist.
- Extend the framework for efficient protocols to the setting where any minority of the players may be corrupted. The current framework seems not to be applicable to this setting.

- Furthermore, all results in this thesis are for the synchronous model, where the delay of messages in the network is bounded by a known constant. This assumption is rather strong in many real-world settings, and it would be interesting to adapt the results for the asynchronous model.
- In the context of receipt-free voting, the main open problem is a formalization and a crisp definition of receipt-freeness. So far, there are many incompatible definitions of receipt-freeness in use, and it is even unclear how they compare. A good definition would allow to analyze and prove the security of existing and of new protocols.

Bibliography

[AR63] Sheldon Akers and Theodore Robbins. Logical design with three-input majority gates. *Computer Design*, pages 12–27, March 1963.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 201–210, August 1989.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proc. 25th ACM Symposium on the Theory of Computing (STOC)*, pages 52–61, 1993.

[Bea89] Donald Beaver. Perfect privacy for two-party protocols. In *Proc. of the DIMACS Workshop on Distributed Computing and Cryptography*, October 1989.

[Bea91a] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, 1991.

[Bea91b] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, pages 75–122, 1991.

[BFKR90] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pages 410–415, 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer-Verlag, 1988.

[Bla79] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 1979*, volume 48 of *American Federation of Information Processing Societies Proceedings*, pages 313–317, 1979.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. 22nd ACM Symposium on the Theory of Computing (STOC)*, pages 503–513, 1990.

[BPP98] Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. Technical Report PP-1998-19, Department of Mathematics and Computer Science, Odense University, October 1998.

[BPW91] Birgit Baum-Waidner, Birgit Pfitzmann, and Michael Waidner. Unconditional byzantine agreement with good majority. In *8th Annual Symposium on Theoretical Aspects of Computer Science*, volume 480 of *lncs*, pages 285–295, Hamburg, Germany, 14–16 February 1991. Springer.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A pardigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proc. 26th ACM*

Symposium on the Theory of Computing (STOC), pages 544–553. ACM, 1994.

[BY86] Josh Cohen Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *Proc. 5th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 52–62, August 1986.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

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

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, 1988.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge for finite field arithmetic. Or: Can zero-knowledge be for free? In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441, 1998.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326, 1999.

[CDG87] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119. Springer-Verlag, 1987.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli 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*, 2000.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology — EUROCRYPT '01*, Lecture Notes in Computer Science, 2001. To appear.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 90–104, 1997.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*. IACR, Springer-Verlag, 1994.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proc. 26th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 372–382. IEEE, 1985.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc. 28th ACM Symposium on the Theory of Computing (STOC)*, pages 639–648, November 1996.

[CFSY96] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. IACR, Springer-Verlag, May 1996.

[CG96] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. In *Proc. 37th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 504–513, 1996.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 383–395, 1985.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election

scheme. In *Advances in Cryptology — EUROCRYPT '97*, Lecture Notes in Computer Science, 1997.

[CGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology — CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer-Verlag, 1995.

[CH94] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 425–438, 1994.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cha89] David Chaum. The spymasters double-agent problem. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 591–602. Springer-Verlag, 1989.

[CK89] Benny Chor and Eyal Kushilevitz. A zero-one law for Boolean privacy. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, volume 21, pages 62–72, 1989.

[CKOR97] Ran Canetti, Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Randomness vs. fault-tolerance. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 35–44, August 1997.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinopole: Practical asynchronous Byzantine agreement using cryptography. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, July 2000. To appear.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and Univ. of Amsterdam, November 1996.

[CW89] Brian A. Coan and Jennifer L. Welch. Modular construction of nearly optimal Byzantine agreement protocols. In

Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC), pages 295–305, August 1989.

[Dam99] Ivan Damgård. BRICS technical report RS-99-2, February 1999.

[DDWY93] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM*, 40(1):17–47, January 1993.

[DFF+82] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Control*, 52(3):257–274, March 1982.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC) 2001*, 2001.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, January 1985.

[DRS82] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. 'Eventual' is earlier than 'Immediate'. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 196–203, 1982. Final version: Early Stopping in Byzantine Agreement. In *Journal of the ACM*, 37(4):720-741, October 1990.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, 1984.

[Feh00] Serge Fehr, 2000. Personal communication.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 427–437, 1987.

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multi-party computation. In *Advances in Cryptology — CRYPTO '98*, volume 1462

of *Lecture Notes in Computer Science*, pages 121–136, 1998. Corrected version is available online.

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

[Fit96] Matthias Fitzi. Erweiterte Zugriffstrukturen in Multi-Party-Computation. Student's project, ETH Zurich, 1996. Supervised by Martin Hirt.

[FKN94] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pages 554–563, 1994.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 148–161, 1988.

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

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology — AUSCRYPT '92*, pages 244–251, 1992.

[FPS00] Pierre-Alain Fouque, Gouillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography '00*, Lecture Notes in Computer Science, 2000.

[Fra93] Matthew K. Franklin. *Complexity and Security of Distributed Protocols*. PhD thesis, Columbia University, 1993.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, 1986.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation. In *Proc. 24th ACM Symposium on the Theory of Computing (STOC)*, pages 699–710, 1992.

[Gen96] Rosario Gennaro. *Theory and Practice of Verifiable Secret Sharing*. PhD thesis, Massachusetts Institute of Technology (MIT), May 1996.

[GHY87] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155. Springer-Verlag, 1987.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer Security*, 28:270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

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

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[HH91] Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 309–324, August 1991.

[HM97] Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 25–34, August 1997.

[HM00] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000. Extended abstract in *Proc. 16th of ACM PODC '97*.

[HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer-Verlag, August 2001.

[HMP00] Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer-Verlag, December 2000.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology — EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556, 2000.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st IEEE Symposium on the Foundations of Computer Science (FOCS)*, October 2000.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pages 99–102. IEEE, 1987.

[Ive91] Kenneth R. Iversen. A cryptographic scheme for computerized general elections. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 405–419. IACR, Springer-Verlag, 1991.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154, 1996.

[Kus89] Eyal Kushilevitz. Privacy and communication complexity (extended abstract). In *Proc. 30th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 416–421, 1989.

[KY] Anna Karlin and Andrew C. Yao. Manuscript.

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

[LK00] Byoungcheon Lee and Kwangjo Kim. Receipt-free electronic voting through collaboration of voter and honest verifier. In *Japan-Korea Joint Workshop on Information Security and Cryptology (JW-ISC2000)*, pages 101–108, 2000.

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

[MH96] Markus Michels and Patrick Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 125–132, 1996.

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

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer-Verlag, 1991.

[MR98] Silvio Micali and Phillip Rogaway. Secure computation: The information theoretic case. Manuscript, 1998.

[Oka96] Tatsuaki Okamoto. An electronic voting scheme. In *Proc. of IFIP '96, Advanced IT Tools*, pages 21–30. Chapman & Hall, 1996.

[Oka97] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Proc. of Workshop on Security Protocols '97*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35, 1997.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–59, August 1991.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, 1991.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all / nothing election scheme. In *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259, 1993.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[Rab94] Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of the ACM*, 41(6):1089–1109, November 1994.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pages 73–85, 1989.

[Sak94] Kazue Sako. Electronic voting schemes allowing open objection to the tally. *Transactions of IEICE*, E77-A(1), January 1994.

[Sch91] Claus P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991.

[Sch99] Berry Schoenmakers, 1999. Personal communication.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Sho00] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology — EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, 2000.

[SK94] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 411–424. IACR, Springer-Verlag, 1994.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme – A practical solution to the implementation of a voting booth. In *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[TY98] Yiannis Tsiounis and Moti Yung. On the security of ElGamal-based encryption. In *International Workshop on Practice and Theory in Public Key Cryptography ' 98 (PKC '98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer-Verlag, February 1998.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164. IEEE, 1982.