Linear Integer Secret Sharing

Rune Thorbek

PhD Dissertation

Department of Computer Science
University of Aarhus
Denmark

Linear Integer Secret Sharing

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Rune Thorbek
May 9, 2009

Abstract

In this work, we introduce the Linear Integer Secret Sharing (LISS) scheme, which is a secret sharing scheme done directly over the integers. I.e., the generation of shares is done by an integer linear combination of the secret and some random integer values. The reconstruction of the secret is done directly by a linear integer combination of the shares of a qualified subset of the parties.

The goal of this thesis is to investigate the advantages of the LISS scheme. That is, we investigate the following two questions.

(i) Can we generalize previous results by using the LISS scheme?

(ii) Can the LISS scheme be used to solve problems with new features previously not possible?

To address question (i), we show that any LISS scheme can be used to build a secure distributed protocol for exponentiation in any group. This implies, for instance, that distributed RSA protocols for arbitrary access structures and with arbitrary public exponents, which generalizes previous results.

The second question (ii) is answered when we present two universally composable and practical protocols by which a dealer can, verifiably and non-interactively, secret share an integer among a set of parties. Moreover, at small extra cost and using a distributed verifier proof, it can be shown in zero-knowledge that three shared integers a, b, c satisfy $ab = c$. This implies by known reductions, non-interactive zero-knowledge proofs that a shared integer is in a given interval, or that one secret integer is larger than another. Such primitives are useful, e.g., for supplying inputs to a multiparty computation protocol, such as an auction or an election. The protocols use various set-up assumptions, but do not require the random oracle model.

While this answers the two questions in the affirmative, we continue to investigate the LISS scheme in this work. To emphasize one main result of this thesis, we reconsider Yao's celebrated and heavily investigated question from 1982 [89], where a set of n parties want to evaluate an integer function $f(x_1, \ldots, x_n)$ of n integer variables of bounded range. Initially, party P_i knows the value of x_i and no other x_i's. Is it possible for the parties to compute the value of f, by computing among themselves, without leaking information about their own secret input? One common restriction on previous results is that the function f is always assumed to be represented by an arithmetic circuit over a finite field $\mathbb{F}$, i.e., the arithmetic is done in $\mathbb{F}$. In most scenarios f should be an integer function like Yao proposed. This is solved by choosing $\mathbb{F}$ such that the computations can simulate it over the integers. This introduces two problems.

1. It requires that an upper bound on the input is known before protocol execution.

2. Arithmetic over $\mathbb{F}$ lacks some desirable properties that arithmetic over $\mathbb{Z}$ possesses.

In some scenarios, like in an on-going computation, the upper bound of the input is *not* known before protocol execution. To solve this, the field $\mathbb{F}$ can either be chosen very large or a conversion to a bigger field can be made later. The first solution makes all computations more expensive (since the field size is bigger), the latter conversion is quite expensive as well.

For problem 2, there are several differences between arithmetic over $\mathbb{F}$ and $\mathbb{Z}$. E.g. from integer x, one can easily compute, using integer arithmetic, $x \bmod q$ for any q, so for instance the parity of x can be computed using $q = 2$. If x is represented in $\mathbb{F}$, it is much more complicated to do the same using arithmetic in $\mathbb{F}$.

We solve Yao's problem directly over the integers by using the LISS scheme. This is the first solution of Yao's problem, that solves it directly over the integers. This obviously solves the problems 1 and 2, since LISS uses integer arithmetic, and since the LISS scheme has no upper bound on the value to be shared.

Acknowledgements

First and foremost I thank Ivan Damgård for having been my supervisor. For always having five minutes to discuss crypto, for sharing his knowledge and insights with me, and for inspiration for new areas to investigate. Without him, this work would not exist.

I would also like to thank the crypto group at CWI, Ronald Cramer, Willemien Ekkelkamp, Serge Fehr, Robbert de Haan, Dennis Hofheinz, Eike Kiltz, and Krzysztof Pietrzak.

I'm very thankful for all my team mates and colleagues, Matthias Fitzi, Jakob Funder, Martin Geisler, Jacob Johannsen, Marcel Keller, Mikkel Krøigård, Carolin Lunemann, Gert Læssøe Mikkelsen, Thomas Mølhave, Jesper Buus Nielsen, Janus Dam Nielsen, Claudio Orlandi, Michael Pedersen, Thomas B. Pedersen, Tord Reistad, Louis Salvail, Christian Schaffner, Miroslava Sotakova, Tomas Toft, and Nikos Triandopoulos.

Last but definitely not least, a very special thank you goes to my family, Marta and Thorkil, for all the support and good moods.

Rune Thorbek,
Århus, May 9, 2009.

Contents

Abstract **v**

Acknowledgements **vii**

1 Introduction **3**
1.1 Secret Sharing . . . 3
1.1.1 Threshold Cryptography and Distributed Exponentiation 3
1.1.2 Secure Multi-Party Computation . . . 5
1.2 Linear Integer Secret Sharing . . . 6
1.3 Outline . . . 7
1.4 Work not Presented in this Thesis . . . 8

I The Setting and the Main Tool **11**

2 Preliminaries **13**
2.1 UC Model . . . 13
2.1.1 Notational conventions . . . 13
2.1.2 The UC model . . . 13
2.2 Access Structures and Adversary Structures . . . 15
2.3 Groups and Rings . . . 16
2.4 Modules . . . 17
2.5 Random Variables . . . 17

3 Linear Integer Secret Sharing **19**
3.1 Introduction . . . 19
3.2 Linear Integer Secret Sharing . . . 20
3.3 Constructions . . . 23
3.3.1 Benaloh-Leichter . . . 23
3.3.2 Cramer-Fehr . . . 29
3.3.3 Comparison . . . 30

II The Computational Model **33**

4 Distributed Exponentiation **35**
4.1 Introduction . . . 35

4.2 Distributed Exponentiation 36
4.3 Active Adversaries and Distributed RSA 38

5 Non-Interactive VSS and Multiplication Proofs 43
5.1 Introduction 43
5.2 Verifiable Secret Sharing (VSS) and Distributed Verifier Proofs . 46
5.2.1 Model and Definition 46
5.2.2 An Integer Commitment Scheme 48
5.2.3 Public-key Encryption with Verifiable Opening 49
5.2.4 VSS using Integer Commitments 54
5.2.5 Verifiable Commitment Multiplication Proof 59
5.3 Verifiable Multiplication Proof Based on Pseudo-Random Sharing 60
5.3.1 Replicated Integer Secret-Sharing and Share Conversion . 60
5.3.2 Application to VSS 63
5.3.3 Multiplication Proof 65
5.3.4 RISS-based protocols for General Adversary Structures . 68
5.4 Interval Proofs and Application to Secure Computing 69

6 Efficiency of the Non-Interactive Multiplication Proofs 71
6.1 Introduction 71
6.2 Preliminaries 72
6.3 Comparing the Interval Proofs Approaches 73
6.3.1 Non-Interactive Interval Proofs (I) 73
6.3.2 Non-Interactive Interval Proofs (II) 74
6.3.3 Standard Interval Proof 74
6.4 Discussion 75
6.5 Further Considerations 76

7 Conversion between Fields 77
7.1 Introduction 77
7.2 Preliminaries 78
7.3 Bit Conversion 79
7.3.1 Conversion to a Field of Characteristic 2 80
7.3.2 Conversion Between Two Prime Fields 80
7.3.3 Multiple Conversions 81
7.3.4 The Initial RISS Share 81

III The Information Theoretic Model 83

8 Information Theoretic Integer Commitment Scheme 85
8.1 Introduction 85
8.2 Model and Definitions 86
8.3 Committing to Integers 87

9 Multi-Party Computations **93**
9.1 Introduction . . . 93
9.2 Multiplicative Property of LISS . . . 94
9.3 Multi-Party Computation for Passive Adversary . . . 97
9.3.1 Model and Definitions . . . 97
9.3.2 The Protocols and the Indistinguishability Proof . . . 99
9.3.3 Share Sizes and Communication Complexity . . . 105
9.4 Multi-Party Computation for Active Adversary . . . 105
9.4.1 Auxiliary protocols . . . 107
9.4.2 Communication Complexity . . . 112

10 Interval Proofs in the Information Theoretic Model **113**
10.1 Introduction . . . 113
10.2 Preliminaries . . . 114
10.3 The UC Functionality . . . 115
10.4 Non-negative Number Proofs . . . 115
10.5 Further Considerations . . . 117

IV Beyond this Thesis **119**

11 Future Work **121**
11.1 Conversion . . . 121
11.2 Further Research . . . 122

Bibliography **123**

A One-Time Signature **131**

Index **133**

Introduction

Chapter 1

Introduction

1.1 Secret Sharing

In a secret sharing scheme, a *dealer* distributes *shares* of a secret to a number of shareholders (or parties), such that only certain designated subsets of them (the *qualified sets*) can reconstruct the secret, while other subsets have no information about it. The collection of qualified sets is called the *access structure*. In particular, the access structure consisting of all sets of cardinality greater than t is called a *threshold-t* structure.

Secret sharing was first introduced [82] as a way to store critical information such that we get at the same time protection of privacy and security against losing the information. Later, secret sharing has proved extremely useful, not just as a passive storage mechanism, but also as a tool in interactive protocols.

1.1.1 Threshold Cryptography and Distributed Exponentiation

In threshold cryptography, the private key in a public key scheme is secret shared among a set of servers, and the idea is that a qualified subset of the servers can use their shares to help a client to decrypt or sign an input message, but without having to reconstruct the private key in a single location. As long as an adversary cannot corrupt too large a subset of the servers, he cannot prevent the system from working, nor can he learn any information on the private key.

The central operation we need to perform securely in these applications is typically an exponentiation, that is, we are given some finite group G and an input $a \in G$, and we want to compute a^s, where s is a secret exponent that has been secret-shared among the servers. In some cases the group order is a public prime q. The problem is then straightforward to solve since we can use any standard linear secret sharing scheme over the field $\mathbb{Z}_q$. The observation is simply that for any linear scheme (such as Shamir's) over $\mathbb{Z}_q$, the secret can be written as a linear combination $s = \sum_{i \in I} \alpha_i s_i \bmod q$, where I is any qualified set of servers holding shares $\{s_i\}_{i \in I}$, and where the α_i's can be computed from the index set I. Now, if the servers provide $a_i = a^{s_i}$ (and prove they did so correctly), we can compute $a^s = \prod_{i \in I} a_i^{\alpha_i}$. However, there are other cases where the group order is not prime and is not public (or even unknown to everyone),

such as when G is $\mathbb{Z}_N^*$ for an RSA modulus N or when G is a class group. This leads to various problems: it would be natural to try to build a secret sharing scheme over $\mathbb{Z}_t$ where t is the order of G, but the standard constructions do not immediately work if t is not a prime. Matters are of course even worse if t is unknown to everyone.

The literature contains many techniques for getting around these problems. The techniques work in various particular scenarios, but they all have shortcomings in general. We give a short overview here:

- The black-box secret sharing schemes of [32, 45, 80] can be used to share a secret chosen from any Abelian group, including $\mathbb{Z}_t$. This requires, of course, that the dealer knows t so he can do computations in $\mathbb{Z}_t$. This is never the case if G is a class group, and if $G = \mathbb{Z}_N^*$, the dealer must know the factorization of N. Note that in proactive threshold RSA schemes, each party typically has to reshare his share of the private key from time to time, however, we can of course not afford to reveal the factorization of N to every shareholder.

- In Shoup's threshold RSA protocol [83], the idea is to restrict the modulus N to be a safe prime product, which allows us to work in a subgroup of $\mathbb{Z}_N^*$ whose order is the product of two large primes. This is "close enough" to a prime so that standard Shamir sharing of s will work. This requires that the dealer knows the factorization. Moreover, for technical reasons, the protocol can only compute $a^{s \cdot n!}$ where n is the number of servers. This is solved by exploiting that we have the public exponent e available. Assuming e is relatively prime to $n!$, we can compute a^s efficiently. The problem in general is of course that we may not always be able to choose the group order as we like, and the inverse of s modulo the group order may not always be available or it may not be prime to $n!$. For instance, we cannot use small public exponents such as 3.[1]

- The secret sharing scheme of [48] which was also used in [35,39] is a variant of Shamir's scheme, where we use polynomials over the integers. Using this to share s does not require any knowledge of the order of G. However, the scheme does not allow reconstruction of s by a linear combination of shares, instead one obtains the secret times some constant, typically $s \cdot n!$. This causes the protocol to produce $a^{s \cdot n!}$ as output, and we have the same problem as with Shoup's protocol.

- Finally, the method of Rabin [78] uses secret sharing in "two levels", i.e., the secret exponent s is shared additively, such that $s = s_1 + ... + s_n$ where server i knows s_i, and then s_i is itself secret shared among the servers. Schemes of this type require no knowledge of the group order to do the sharing since in principle, any secret-sharing scheme can be used to share the s_i's. On the other hand, shares become larger than with other schemes and extra rounds of interaction is needed (to reconstruct s_i) as soon as

[1] Shoup suggests an alternative solution where any public exponent can be used, but this requires that one additionally assumes that the DDH assumption holds in the RSA group

even one server i fails to participate correctly. Hence (in contrast to the other types of protocols) this approach cannot be made non-interactive, not even in the random oracle model.

A final issue with current state of the art of distributed exponentiation is that known solutions (except the two-level method) do not generalize to non-threshold access structures. The point of general structures is that when we secret share the private key according to a threshold structure, we are implicitly assuming that all servers are equally easy to break into, and so the only important parameter is the *number* of corrupted servers. In reality, some servers may well be more reliable than others, and so we may need to specify which sets should be qualified in a more flexible way, that is, we need a more general access structure.

1.1.2 Secure Multi-Party Computation

The goal of *secure multi-party computation* (MPC), as introduced by Yao [89], is to enable a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to solve the following general problem. Given an arbitrary function $f(x_1, \ldots, x_n)$, which is an integer-valued function of n integer variables x_i of bounded range. Assume that initially party P_i knows the value of x_i, but no other x_j's. Is it possible for them to compute the value of f, by communicating among themselves, without leaking information about their own secret input? The computation must guarantee the correctness of the result while preserving the privacy of the parties' inputs, even if some of the parties are corrupted by an adversary and misbehave in an arbitrary malicious way. Since the first results in this area [9, 24, 57, 90], the focus has mainly been put into improving these results, e.g., improving the communication complexity [49, 54, 61] or round complexity [5, 7, 8, 63], and coping with more powerful [20, 76] or more general adversaries [30, 47, 61].

While Yao [89] proposed the problem for an integer valued function f, a common restriction on all these results is that the function f is always assumed to be represented by an arithmetic circuit over a finite field $\mathbb{F}$. This implies that all computations are done over $\mathbb{F}$. Since in most realistic problems the function f is an integer function, we would like to solve the problem over the integers as Yao proposed. There are mainly two strategies.

1. Choose $\mathbb{F}$ big enough to simulate the computation over the integers.

2. Make secret sharings of the bit-wise representation of the integers.

The first solution has the drawback that an upper bound on the input must be known before protocol execution. In an on-going MPC computation the input can be provided in any round and the input size might depend on unpredictable factors. Hence, the size of $\mathbb{F}$ might be impossible to predict or must be chosen unreasonably big to simulate an integer computation of a function f. Note that the size of $\mathbb{F}$ determines the efficiency of the computation, hence, it is always preferable to choose $\mathbb{F}$ as small as possible. Another strategy would be to choose $\mathbb{F}$ to be a reasonable size, and make a conversion if necessary at a later point

to a bigger field size. The problem with this solution is that the conversion is quite cumbersome and expensive.

While solution 2 obviously does not have the above drawback, it is too expensive to do general computations with the binary representation. Note, that some problems can be solved elegantly with binary representation (see e.g. [37]), but this is not the case in general.

Another problem is that when representing the integers in a finite field $\mathbb{F}$ (or binary), then there is somehow limited information about the integer value v. Say, given an integer value v represented in $\mathbb{Z}_p$, then we can only perform arithmetic in $\mathbb{Z}_p$. Hence, it requires extra computations to find, e.g., $v \bmod q$ for $q < p$. If v is represented directly as an integer, and hence the arithmetic is done over $\mathbb{Z}$, then this information is directly available by reducing v modulo q.

This shows, that there are some obvious limitations of the current solution strategies of Yao's original problem, which both from a theoretical and practical point of view, would be interesting to solve.

1.2 Linear Integer Secret Sharing

In this thesis we propose the *linear integer secret sharing* (LISS) scheme. In a LISS scheme all the computations are done over the infinite set of integers.

The secret in a LISS scheme is an integer s chosen from some publically known interval, e.g. $[-2^l..2^l] \subset \mathbb{Z}$. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote the set of n parties that the secret s is shared among. We say that a subset $A \subseteq \mathcal{P}$ is *qualified* if the parties in A jointly are allowed to reconstruct the secret s. In a LISS scheme, the shares consists of a collection of integers $\{s_i\}_{i \in I}$, where for each $i \in I$ the integer s_i belongs to exactly one party and s_i is computed by a linear integer combination of s and some randomness chosen by the dealer. Given a qualified subset of shares $\{s_i\}_{i \in I'}$, then the secret can be reconstructed by a linear integer combination

$$s = \sum_{i \in I'} \lambda_i s_i,$$

where $\{\lambda_i\}_{i \in I'}$ are integer coefficients that are determined by the index set I'.

While this defines the requirement of the LISS scheme, we will later give a construction for the LISS scheme for any *monotone access structure*. That is, for each collection of qualified subsets Γ that fulfills that if $A \in \Gamma$ and $A \subseteq B \subseteq \mathcal{P}$ then $B \in \Gamma$.

Since we require that the secret can be reconstructed by a linear integer combination of a qualified subset of parties, it gives the following advantages.

- It solves the distributed exponentiation in a group of any order, even if the group has unknown order. This follows by the following simple observation. Say, $\{s_i\}_{i \in I}$ is a set of LISS shares for a qualified subset of parties. Then there exists a set of integer coefficients $\{\lambda_i\}_{i \in I}$, such that $\sum_{i \in I} \lambda_i s_i = s$, where s is the secret shared over the LISS scheme. Let G be an arbitrary group, possibly of unknown order. Then if $s \in \mathbb{Z}_{|G|}$ is the

secret exponent, then for each $i \in I$ compute $a_i = a^{s_i}$. Finally note,

$$\prod_{i \in I} a_i^{\lambda_i} = a^{\sum_{i \in I} \lambda_i s_i \bmod |G|} = a^{\sum_{i \in I} \lambda_i s_i} = a^s.$$

- Given any qualified subset of a LISS secret sharing $\{s_i\}_{i \in I}$, it follows that $s = \sum_{i \in I} \lambda_i s_i$, where $\{\lambda_i\}_{i \in I}$ are integers. Let $m \geq 2$ be any modulus, then note that the set $\{s_i \bmod m\}_{i \in I}$ can be reconstructed to $s \bmod m$ by linear combination of $\sum_{i \in I} \lambda_i s_i \bmod m$ in $\mathbb{Z}_m$. Put in another way, the LISS scheme can be locally converted to a sharing of the secret modulo any value. That is, the LISS scheme contains more information about the secret s, than a secret sharing scheme over a finite field, ring, or group.

We stress that the above is only possible since all computations are done over the integers.

The LISS scheme is linear, that is, given two LISS shares of the integers a and b, respectively, then it is possible to locally compute without interaction LISS shares of $a + b$ and αa, for any public constant α. Another feature of the LISS scheme is that it can be made multiplicative. That is, given two LISS shares of a and b, respectively, it is possible to compute a LISS share of $c = ab$. This solves Yao's original problem using integer computations only.

Since there is no upper bound on the size of the secret shared over LISS[2], it can be used in on-going computations, where no upper bound on the input is known before protocol execution. Put differently, the shares in a LISS scheme can depend upon the secret size directly, without limiting the size of other secrets used in the computations. Another advantage is the fact that the LISS scheme works directly over the integers and therefore the arithmetic is done over the integers, e.g., as pointed out above, a LISS share can be reduced modular any modulus.

Another advantage of the integers is that every non-negative integer can be written as four squares, i.e., for all $x \in \mathbb{Z}$ and $x \geq 0$ there exist $a, b, c, d \in \mathbb{Z}$, such that $x = a^2 + b^2 + c^2 + d^2$. Assume that a dealer D needs to prove in zero knowledge, that some provided secret shared input x is non-negative. Then D can simply find four integers a, b, c, d and secret share them and finally open $x - a^2 - b^2 - c^2 - d^2$ to reveal 0. This is useful in many scenarios, e.g., if D needs to prove that a secret shared value x is contained in some publically known interval $[x_l..x_h]$, he shows that $x - x_l$ and $x_h - x$ are non-negative, or if D provides secret shared values $x_1, \ldots, x_m$ and wants to prove that $x_1 \leq \ldots \leq x_m$, then D shows that $x_i - x_{i-1}$ is non-negative for $i = 2, \ldots, m$.

1.3 Outline

The thesis is structured as follows. In Part I we briefly describe the setting and formally define the LISS scheme, including a proof of security. After defining the LISS scheme, we give a construction of it for any given monotone access structure.

[2]Only the randomness used in the computation of shares depends upon the share size.

In Part II we investigate the LISS scheme in the computational model. We start by giving a solution to the distributed exponentiation problem, which generalizes previous solutions with the following advantages:

- It works for any public exponent and any modulus (e.g., an RSA modulus).
- The dealer does not need to know the order of the group.
- It works for any monotone access structure.
- The protocol is efficient and runs in constant rounds.

The distributed exponentiation protocol was also presented in [41]. While this is a generalization of previous results, we continue to exploit that each non-negative integer can be written as the sum of four squares. This is done by providing two protocols that by a non-interactive distributed verifier proof show in zero-knowledge that a shared integer is the product of two other shared integers. By the above observation, this implies that we can, non-interactively in zero-knowledge, prove that a shared integer is non-negative without using the random oracle model. This was also presented in [42].

We finish the computational part of the thesis with a protocol that can convert a secret shared bit b from one finite field $\mathbb{F}$ to another finite field $\mathbb{F}'$. This is done by using a *replicated integer secret sharing*, which loosely speaking is an inefficient LISS share. The protocol was also presented in [43].

In Part III we proceed in the information theoretic model, where we solve Yao's proposed problem in the first direct solution over the integers. That is, all the computations are done directly of secret sharings over LISS, i.e., secret sharings directly over the integers. As already pointed out, this has some advantages that are not possible in the previous solutions of the problem. To solve the problem for an active adversary we need various tools, among them an information theoretic commitment scheme over the integers.

Finally, we also solve the non-negative number proof problem in the information theoretic model. This is done at the additional cost of one round (two rounds in total), compared to the solution in the computational model.

Finally, in Part IV we consider whether further investigations are necessary for secret sharing over the integers.

1.4 Work not Presented in this Thesis

While the subject of this thesis is the LISS scheme, the focus has been on the advantages of secret sharing over the integers opposed to secret sharing over finite groups or fields. This thesis will therefore not explore all possible applications of LISS. Concretely, the LISS scheme can be made proactive as shown in [84]. However, there is no direct advantage in doing it over the LISS scheme compared to doing it over any other secret sharing scheme. Therefore, the result of [84] is not included in this thesis.

In [38] the public key encryption scheme with non-interactive openings (PKENO) was formally introduced. The work includes a UC functionality along

with a game-based definition of the PKENO, which are shown to be equivalent. Furthermore, a solution based directly on an identity based encryption (IBE) scheme is presented, along with a more direct solution.[3] The PKENO notion was informally introduced in [42], where it is used as a primary tool to make the presented protocol non-interactive.

In Chapter 5 (which mainly presents the work of [42]) we give the game-based definition along with the IBE based solution from [38]. The rest of the work (the UC definition, the equivalence proof, and the direct solution) is not included in this thesis. While the results in [38] still are interesting, they are simply outside the scope of the thesis.

[3]The direct solution has a flaw, which was pointed out by David Galindo and reparied by him in [53].

Part I

The Setting and the Main Tool

Chapter 2

Preliminaries

2.1 UC Model

We assume the reader to be familiar with the universal composability (UC) framework [19] and only sketch it at a high level here. For a detailed introduction of the UC framework we refer the reader to the full version of [19].

2.1.1 Notational conventions

If x is a string, then $|x|$ denotes its length, while if S is a set then $|S|$ denotes its size. If $k \in \mathbb{N}$ then 1^k denotes the string of k ones. If S is a set then $s \leftarrow_{\mathrm{R}} S$ denotes the operation of picking an element s of S uniformly at random. Unless otherwise indicated, algorithms are randomized and polynomial time. An adversary is an algorithm or a tuple of algorithms. A function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if and only if there exists $c < 0$ such that $|f(k)| < k^c$ for all sufficiently large k. We write $f \approx g$ if $f - g$ is negligible.

2.1.2 The UC model

Canetti's Universal Composability (UC) framework [19] for multi-party computation allows to formulate security and composition of multi-party protocols in a very general way. The idea of the UC model is to compare a protocol to an idealization of the protocol task. Security means that the protocol "looks like" the idealization even in face of arbitrary attacks and in arbitrary protocol environments. This notion of security is very strict [4,21,44], but implies useful compositional properties [19]. In fact, in a certain sense, this notion is even necessary for secure composition of protocols [70].

The real model. We shortly outline the framework for multi-party protocols defined in [19]. First of all, parties (denoted by P_1 through P_n) are modeled as interactive Turing machines (ITMs) and are supposed to run some fixed protocol (i.e., program) Π. There also is an adversary, denoted $\mathcal{A}$ and modeled as an ITM as well, that carries out attacks on protocol Π. Therefore, $\mathcal{A}$ may corrupt parties (in which case it learns the party's state and controls its future actions), and intercept or inject messages sent between parties. If $\mathcal{A}$ corrupts parties only before the actual protocol run of Π takes place, $\mathcal{A}$ is called

static (or non-adaptive), otherwise $\mathcal{A}$ is said to be adaptive. In this work, we only consider static corruptions. The respective local inputs for all parties of protocol Π are supplied by an environment machine (modeled as an ITM and denoted $\mathcal{Z}$) that may also read all protocol outputs locally made by the parties and communicate with the adversary.

The ideal model. The model we have just described is called the real model of computation. In contrast to this, the ideal model of computation is defined just like the real model, with the following exceptions: all party ITMs are replaced with one single ideal functionality $\mathcal{F}$. The ideal functionality may not be corrupted by the adversary, yet it may send messages to and receive messages from it. Finally, the adversary in the ideal model is called "simulator" and denoted $\mathcal{S}$. The only means of attack the simulator has in the ideal model are corruptions (in which case $\mathcal{S}$ may supply inputs to and read outputs from $\mathcal{F}$ in the name of the corrupted party), delaying or suppressing outputs of $\mathcal{F}$, and all actions that are explicitly specified in $\mathcal{F}$. However, $\mathcal{S}$ has no access to the inputs $\mathcal{F}$ gets and to the outputs $\mathcal{F}$ generates, nor are there any protocol messages to intercept. Intuitively, the ideal model of computation (or, more precisely, the ideal functionality $\mathcal{F}$ itself) should represent what one ideally expects the protocol to do. In fact, for a number of standard tasks, there are formulations as such ideal functionalities (see, e.g., [19]).

Security definition. To decide whether or not a given protocol Π fulfills the requirements of our ideal specification $\mathcal{F}$, the framework of [19] uses a simulatability-based approach: at a time of its choice, $\mathcal{Z}$ may halt and generate output. The random variable describing the first bit of $\mathcal{Z}$'s output will be denoted by $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run with security parameter $k \in \mathbb{N}$ and initial input $z \in \{0,1\}^*$ in the real model of computation, and $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run in the ideal model. Now Π is said to *securely realize* $\mathcal{F}$ if and only if for any real adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ and any (possibly non-uniform) family of initial inputs $z = (z_k)_k$, we have

$$\mathsf{Pr}\left[\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z_k)=1\right] \approx \mathsf{Pr}\left[\mathrm{IDEAL}_{\mathbb{F},\mathcal{S},\mathcal{Z}}(k,z_k)=1\right]. \tag{2.1}$$

This differs slightly from the original formulations in [19], but is equivalent and eases our presentation. Intuitively, (2.1) means that any attack against the protocol can be simulated in the ideal model. Hence, any weakness of the real protocol is already contained in the ideal specification (that does not contain an "actual" weakness by definition). Interestingly, the "worst" real attack possible is the one carried out by the dummy adversary $\tilde{\mathcal{A}}$ that simply follows $\mathcal{Z}$'s instructions. That means that for security, it actually suffices to demand existence of a simulator that simulates attacks carried out by $\tilde{\mathcal{A}}$.

In general all entities are polynomial time bounded, which gives security in the computational model, where security can be based on computational assumptions (e.g., the RSA assumption). When considering the information theoretic model, we allow the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ to be computational unbounded, and we allow $\mathcal{S}$ to be polynomial in the complexity of $\mathcal{A}$.

Composition of protocols. To formalize the composition of protocols, there also exists a model "in between" the real and ideal model of computation. Namely, the hybrid model of computation is identical to the real model, except that parties have access to (multiple instances of) an ideal functionality that aids in running the protocol. This is written as $\phi^{\mathcal{F}}$ for the actual protocol ϕ and the ideal functionality $\mathcal{F}$. Instances of $\mathcal{F}$ are distinguished via session identifiers (short: session ids, or *sid*s). Note that syntactically, instances of $\mathcal{F}$ can be implemented by a protocol Π geared towards realizing $\mathcal{F}$. And indeed, the universal composition theorem [19] guarantees that if one protocol instance of Π is secure, then many protocol instances are, even when used in arbitrary protocols ϕ. More concretely, if Π securely realizes $\mathcal{F}$, then ϕ^{Π} securely realizes $\phi^{\mathcal{F}}$ for any protocol ϕ. Here, $\phi^{\mathcal{F}}$ denotes that ϕ uses (up to polynomially many) instances of $\mathcal{F}$ as a subprimitive, and ϕ^{Π} denotes that ϕ uses instances of Π instead.

Conditional security and composability. Universal composability is a very strict notion. So sometimes (e.g., in the case of bit commitments), it is not possible to achieve full UC security. Hence, several weakenings of the notion have been proposed. One method that will be useful in our case is to consider only protocol environments that conform to certain rules (see [3, 73]). Concretely, secure realization with respect to a certain class $\mathfrak{Z}$ of environments means that in 2.1, we quantify only over environments in $\mathfrak{Z}$. This relaxed security notion still gives precisely those compositional guarantees one would expect: secure composition with larger protocols that can be seen as restricted environments from $\mathfrak{Z}$ (see [3, 73] for details).

We will explicitly note when we quantify over specific environments. Mainly, we need to restrict the environment to let the honest parties use the functionality as "intended". That basically means, that honest parties follow the protocol description, which is a reasonable assumption.

2.2 Access Structures and Adversary Structures

Let n be a positive integer, then consider the set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties.

Definition 2.1. *A non-empty collection Γ of subsets of $\mathcal{P}$ is called a* monotone access structure *on $\mathcal{P}$ if $\varnothing \notin \Gamma$ and if Γ is closed under taking supersets, that is, for all $A \in \Gamma$ and all $B \subseteq \mathcal{P}$ with $A \subseteq B$ it holds that $B \in \Gamma$.*

Definition 2.2. *A collection Δ of subsets of $\mathcal{P}$ is called a* monotone adversary structure *on $\mathcal{P}$ if $\varnothing \in \Delta$ and if Δ is closed under taking subsets, that is, for all $A \in \Delta$ and for all $B \subseteq \mathcal{P}$ with $B \subseteq A$ it holds that $B \in \Delta$.*

There is a natural one-to-one correspondence between access structures and adversary structures on $\mathcal{P}$: Let $\bar{\Gamma} = \{A \subseteq \mathcal{P} \mid A \notin \Gamma\}$ be the complement of an access structure Γ, then $\bar{\Gamma}$ is an adversary structure. Likewise, $\bar{\Delta} = \{A \subseteq \mathcal{P} \mid A \notin \Delta\}$ for an adversary structure Δ is an access structure. Given a monotone access structure Γ, we call the complement $\bar{\Gamma}$ the *corresponding adversary structure*. Likewise, we call the complement $\bar{\Delta}$ of a given adversary structure Δ the *corresponding access structure*.

A *minimal qualified set* $A \in \Gamma$ is a set such that there exists no proper subset of A contained in Γ, i.e., there exists no set $B \in \Gamma$ such that $B \subset A$ and $A \neq B$. A *maximal forbidden set* $A \in \Delta$ is a set such that there exists no proper superset of A contained in Δ, i.e., there exists no set $B \in \Delta$ such that $A \subset B$ and $A \neq B$. The collection of minimal qualified sets is denoted Γ^- and the collection of maximal forbidden sets is denoted Δ^+.

Definition 2.3. *Let t be an integer in the range $0 \leq t \leq n-1$. The* threshold access structure *$T_{t,n}$ is the collection of all subsets $A \subseteq \mathcal{P}$ with $|A| > t$.*

Definition 2.4. *An adversary structure Δ on $\mathcal{P}$ is $Q2$ ($Q3$) if no two (three) sets in the structure cover the full set of parties $\mathcal{P}$, i.e., there does not exist $A, B \in \Delta$ ($A, B, C \in \Delta$) such that $A \cup B = \mathcal{P}$ ($A \cup B \cup C = \mathcal{P}$).*

The $Q2$ ($Q3$) condition was introduced in [61] and generalizes the threshold condition $t < n/2$ ($t < n/3$) for threshold adversary structures to arbitrary adversary structures.

2.3 Groups and Rings

We assume the reader to be familiar with basic concepts of group and ring theory or refer the reader to an algebra book, e.g., [69, 71]. In this section we fix some notation and terminology.

We let $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ denote the ring of integers, where we use addition and multiplication as usual.

Let $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$, where we write $a + b \bmod m$ and $ab \bmod m$ to denote addition and multiplication in $\mathbb{Z}_m$, respectively. When clear from the context, we only write $a + b$ and ab. We let $\mathbb{Z}_m^*$ denote the multiplicative invertible elements of $\mathbb{Z}_m$, that is, all the elements $x \in \mathbb{Z}_m$ that has a $y \in \mathbb{Z}_m$ such that $xy \bmod m = 1 \bmod m$. We write x^{-1} to represent the multiplicative inverse of an element $x \in \mathbb{Z}_m^*$.

We often need a number to be restricted from some interval of the integers. We represent intervals by $[a..b]$, where $a < b$, that represents the integers $\{a, a+1, \ldots, b-1, b\} \subseteq \mathbb{Z}$. Note here, that when we pick numbers $x, y \in [a..b]$, then we do the addition $a + b$ and multiplication ab over the integers, hence $a + b$ and ab might not be contained in the interval $[a..b]$.

We let $\mathbb{Z}^d$ denote ordered integer tuples of d integers. We call an element $\boldsymbol{x} \in \mathbb{Z}^d$ for $d > 1$ a vector, and represent it by $\boldsymbol{x} = (x_1, \ldots, x_d)^T$. We denote an integer matrix $M \in \mathbb{Z}^{d,e}$ by,

$$M = \begin{pmatrix} m_{1,1} & \cdots & m_{1,e} \\ \vdots & \ddots & \vdots \\ m_{d,1} & \cdots & m_{d,e} \end{pmatrix}.$$

For notational convenience we often write $M = [m_{i,j}]$, where the dimensions of M should be clear from the context.

2.4 Modules

In this section we introduce the concept of modules, which loosely speaking is a vector space over a ring.

Definition 2.5. *Let R be any ring. An* R-module *A is an additive abelian group together with a function $R \times A \to A$, subject to the following axioms, for all elements $x, y \in R$ and $a, b \in A$:*

i) $x(a+b) = xa + xb$.

ii) $(x+y)a = xa + ya$.

iii) $(xy)a = x(ya)$.

vi) $1a = a$.

Such a module is more explicitly called a *left* module, because the scalars is written on the left of the module element.

Example 2.1. $\mathbb{Z}^2$ is an $\mathbb{Z}$-module, where the group operation on $\mathbb{Z}^2$ is defined as $(a_1, a_2)^T + (b_1, b_2)^T = (a_1 + b_1, a_2 + b_2)^T$ and the scalar by $x(a_1, a_2)^T = (xa_1, xa_2)^T$.

In this work we only consider $\mathbb{Z}$-modules of the form $\mathbb{Z}^n$.

2.5 Random Variables

Let Ω denote the collection of all outcomes for a given experiment.

Definition 2.6. *A collection $\mathscr{A}$ of subsets of Ω is an* algebra *if*

i) $A, B \in \mathscr{A}$ *implies* $A \cup B \in \mathscr{A}$.

ii) $A \in \mathscr{A}$ *implies* $A^{\complement} \in \mathscr{A}$.

iii) $\Omega \in \mathscr{A}$.

Definition 2.7. *A collection $\mathscr{F}$ of subsets of Ω is a* σ-algebra *if*

i) $\mathscr{F}$ *is an algebra.*

ii) If $\{A_j\}$ *is an infinite sequence in* $\mathscr{F}$, *then* $\bigcup A_j \in \mathscr{F}$.

Definition 2.8. *A* probability space *is a triple* $(\Omega, \mathscr{F}, \Pr)$ *where* $\mathscr{F}$ *is a non-empty* σ*-algebra of subsets of* Ω *and* $\Pr$ *is a mapping from* $\mathscr{F}$ *to* $\mathbb{R}$ *satisfying*

i) $\Pr(\Omega) = 1$.

ii) $0 \leq \Pr(A) \leq 1$ *for all* $A \in \mathscr{F}$.

iii) If $\{A_j\}$ *is a finite sequence of infinite disjoint sequence in* $\mathscr{F}$, *then*

$$\Pr(\cup A_j) = \sum \Pr(A_j).$$

The elements of $\mathscr{F}$ are called events.

Definition 2.9. *Let $(\Omega, \mathscr{F}, \Pr)$ be a probability space. A mapping $X\colon \Omega \to \mathbb{R}$ is called a* random variable *if $(X \leq x) = \{\omega \mid X(\omega) \leq x\} \in \mathscr{F}$ for all $x \in \mathbb{R}$.*

Example 2.2. Let $\Omega = [-2^l..2^l] \subset \mathbb{Z}$ and $X(\omega) = \omega$ for $\omega \in \Omega$. Then $(X \leq x) = \{\omega \mid X(\omega) \leq x\} = \{\omega \mid \omega \leq x\} \subset \mathbb{Z}$. That $(X \leq x) \in \mathscr{F}$ for all $x \in \mathbb{R}$ means that $\Pr(X \leq x)$ is defined for all $x \in \mathbb{R}$ and defines a function on $\mathbb{R}$.

If the domain of a random variable is a countable set $\{x_1, x_2, \ldots\}$, then it follows that $\{x_i\} \in \mathscr{F}$, hence, $\Pr(X = x_i)$ is well defined for all $i = 1, 2, \ldots$.

We will only consider countable finite random variables, similar to the one given in the example above.

Definition 2.10. *Let X and Y be countable random variable taking values on set V then the* statistical distance *is defined as*

$$[X; Y] = \frac{1}{2} \sum_{v \in V} \left| \Pr(X = v) - \Pr(Y = v) \right|.$$

Example 2.3. Let X be a random variable with the domain $\{1, 2, \ldots, n\}$ and define the random variable $Y = X + m$ for a positive m, i.e., Y has domain $\{1 + m, 2 + m, \ldots, n + m\}$ and $\Pr(X = i) = \Pr(Y = i + m)$ for $i = 1, 2, \ldots, n$. First observe that if $n < m + 1$ then the statistical distance $[X; Y] = 1$. Assume that $m < n$ and that X has a uniformly random distribution, then the statistical distance is given by

$$[X; Y] = \frac{m}{n}$$

Theorem 2.1 (Union bound [60])**.** *If $\{A_i\}$ is any sequence of events, then*

$$\Pr\left[\bigcup_i A_i\right] \leq \sum_i \Pr[A_i]$$

Chapter 3

Linear Integer Secret Sharing

3.1 Introduction

In this chapter, we introduce the main tool of this thesis: the *Linear Integer Secret Sharing* (LISS) scheme. In the LISS scheme, the secret is an integer chosen from a (publically known) interval, and each share is computed as an integer linear combination of the secret and some random numbers chosen by the dealer. Reconstruction of the secret is done by computing a linear combination with integer coefficients of the shares in a qualified set.

LISS schemes are closely related to - but not the same as - the black-box secret sharing schemes (BBSS) of Desmedt-Frankel [45] and Cramer-Fehr [32]. Whereas BBSS schemes are designed to secret share elements from any *finite* abelian group, we work over the (infinite) group of integers. This difference has a number of consequences that we return to below. LISS schemes are also different from the method in [48] based on integer polynomials, since they require a final division to get the secret while for LISS schemes we insist that linear combinations to be sufficient.

Note that it was shown in [25, 27] that perfect secret sharing and private computation over countably infinite domains (like the integers) is not possible. However, this does not rule out schemes of our type since we restrict our secrets to be chosen from a publically known interval and only aim for statistical rather than perfect privacy.

Cramer and Fehr introduce the concept of an integer span program (ISP) and use it to construct BBSS schemes. We show that any ISP can also be used to build a secure LISS scheme. Roughly speaking, an ISP is specified by a matrix with integer entries, and these entries are used as coefficients in the linear combinations that produce the shares from secret and randomness. In particular, the construction from [32] of an ISP for threshold-t access structures implies a LISS scheme for the same structure. Moreover, we revisit the well known construction of Benaloh and Leichter [10] based on monotone formulas that was originally conceived for a finite Abelian group, and we show that a LISS scheme can be built from any monotone formula. This implies that LISS schemes exists for any access structure, though they are not necessarily efficient.

The ISP construction of Cramer and Fehr was shown to imply optimal

threshold BBSS schemes. We show that this is not always the case for LISS schemes: if we base the Benaloh-Leichter construction on a monotone formula for the threshold function, we obtain threshold LISS schemes. It now turns out that, depending on how small a formula we can produce, this construction may produce a threshold LISS scheme with smaller shares or smaller randomness complexity than those coming from the Cramer-Fehr construction. With current state of the art, this does not happen in general, but we find that for a fixed threshold and a large number of parties, there are monotone formula constructions that produce smaller shares than Cramer-Fehr[1].

The reason why BBSS schemes are different from LISS schemes in this respect is that when we use an ISP for building a BBSS scheme, the size of shares we get is independent of the size of the integers occurring in the description of the ISP, but this is no longer true when we build a LISS scheme.

3.2 Linear Integer Secret Sharing

Assume that there are n parties that are denoted by $P_1, \ldots, P_n$. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of all the parties, and let the power set of $\mathcal{P}$ be denoted by $P(\mathcal{P})$. In the context of secret sharing, a secret s is shared among the parties in $\mathcal{P}$. A set A in $P(\mathcal{P})$ is denoted *qualified* if the parties in the set A are allowed to reconstruct the secret s. Furthermore, a set B in $P(\mathcal{P})$ is denoted *forbidden* if the parties in the set B should not be able to obtain any information about the secret s.

If Γ is the collection of all qualified subsets of parties in $\mathcal{P}$ and Γ is a monotone access structure, then the corresponding adversary structure, Δ, is the collection of all the forbidden sets. Note that, Δ is monotone as required by an adversary structure, and that $\Gamma \cup \Delta = P(\mathcal{P})$ and $\Gamma \cap \Delta = \emptyset$.

Note 3.1. In the context of secret sharing the collection of qualified subsets Γ needs to be monotone. Otherwise there would exist $A \in \Gamma$ and a $B \notin \Gamma$ such that $A \subset B$. This implies that the parties in A should be able to reconstruct the secret, while the parties in B should not.

In a *linear integer secret sharing* (LISS) scheme a dealer D can share a secret s from a publically know interval $[-2^l..2^l]$ over any monotone access structure Γ between the parties in $\mathcal{P}$ such that only qualified subsets can reconstruct the secret while other subsets do not gain any information about the secret. More precisely:

Definition 3.1. *A LISS scheme is* correct*, if the secret can be reconstructed from shares of any qualified set in $A \in \Gamma$, by taking an integer linear combination of the shares with coefficient that depends only on the index set A.*

Definition 3.2. *A LISS scheme is* private*, if for any forbidden set $B \in \Delta$, any two secret $s, s' \in [-2^l..2^l]$, and independent random coins r and r', the*

[1] Note that in a later paper [34], Cramer, Fehr and Stam propose a construction that they conjecture to be more efficient than [32], but so far, the asymptotic efficiency of the scheme remains unproved.

statistical distance between the distributions of the shares $\{s_i(s,r,k) \mid i \in B\}$ *and* $\{s_i(s',r',k) \mid i \in B\}$ *is negligible in the* security *parameter* k.

Formally, a LISS scheme is described as follows. Let $[-2^l..2^l]$ be the set of secrets, then each party j is associated a positive integer $d_j = |\psi^{-1}(j)|$ for $1 \leq j \leq n$, such that the set of possible shares for party j, is a subset $\mathcal{S}_j \subseteq \mathbb{Z}^{d_j}$ of the $\mathbb{Z}$-module $\mathbb{Z}^{d_j}$. Each possible share for party j is in the subset $\mathcal{S}_j$. The *share size* of party j is defined to be the number of bits used to uniquely represent the share from $\mathcal{S}_j$. Note, that $d = \sum_{j=1}^{n} d_j$, where d is the number of share components. Then let $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \subseteq \mathbb{Z}^d$ that defines the subset of possible shares for the parties. Define the *expansion rate* to be $\mu = d/n$, where d is the number of share components and n is the number of parties. Note, that for a given distribution of a secret, the shares of the shareholders can be considered as an element in the subset $\mathcal{S}$. If we use m bits to uniquely represent the shares in $\mathcal{S}$, then we define the *average share size* to be m/μ, i.e., the number of share bits each party will get on average.

To realize a LISS scheme, we need the following tools. A *labeled matrix* consists of a $d \times e$ matrix M and a corresponding surjective function $\psi : \{1,\ldots,d\} \to \{1,\ldots,n\}$. We say that the i-th row is *labeled* by $\psi(i)$ or *owned* by party $P_{\psi(i)}$. For any subset $A \subset \mathcal{P}$, we let M_A denote the restriction of M to the rows jointly owned by the parties in A. We denote d_A for the number of rows in M_A. For any d-vector $\boldsymbol{x}$, we similarly denote $\boldsymbol{x}_A$ to be the restriction of entries jointly owned by the parties in A. For any two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, let $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ denote the inner product.

Definition 3.3. $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ *is called an* Integer Span Program *(ISP), if* $M \in \mathbb{Z}^{d \times e}$ *and the* d *rows of* M *are labelled by a surjective function* $\psi : \{1,\ldots,d\} \to \{1,\ldots,n\}$. *Finally,* $\boldsymbol{\varepsilon} = (1,0,\ldots,0)^T \in \mathbb{Z}^e$ *is called the* target vector. *We define* $\text{size}(\mathcal{M}) = d$, *where* d *is the number of rows of* M.

Definition 3.4. *Let* Γ *be a monotone access structure and let* $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ *be a integer span program. Then* $\mathcal{M}$ *is an* ISP for Γ, *if for all* $A \subseteq \mathcal{P}$ *the following holds.*

- *If* $A \in \Gamma$ *there exists a* reconstruction vector $\boldsymbol{\lambda} \in \mathbb{Z}^d$ *such that* $M_A^T \boldsymbol{\lambda} = \boldsymbol{\varepsilon}$.
- *If* $A \notin \Gamma$ *there exists a* sweeping vector $\boldsymbol{\kappa} \in \mathbb{Z}^e$ *such that* $M_A \boldsymbol{\kappa} = \mathbf{0}$ *and* $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon} \rangle = 1$.

In other words, the rows owned by a qualified set must include the target vector in their span, while for a forbidden set, there must exist a sweeping vector, which is orthogonal to all rows of the set, but has inner product 1 with the target vector. We also say that $\mathcal{M}$ *computes* Γ.

We define $\kappa_{\max} = \max\{|a| \mid a \text{ is an entry in some sweeping vector}\}$. *Let* $l_0 = l + \lceil \log_2(\kappa_{\max}(e-1)) \rceil + 1$. *The size of* $\mathcal{M}$ *is defined to be* d.

Note 3.2. In the case of a *span program*, which works over a field, the explicit requirement of a sweeping vector is not necessary. This is because the following holds for fields, $\boldsymbol{\varepsilon} \in \text{im}(M_A^T)$ if and only if there exists a sweeping vector. When working with the integers then only the "only if" implication is guaranteed.

To share a secret $s \in [-2^l..2^l]$, we use a *distribution vector* $\boldsymbol{\rho}$, which is a uniformly random vector in $[-2^{l_0+k}..2^{l_0+k}]^e$ with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$ and where k is the statistical security parameter. The *share vector* is computed by

$$\boldsymbol{s} = (s_1, \ldots, s_d)^T = M\boldsymbol{\rho}, \tag{3.1}$$

where the *share component* s_i is given to party $P_{\psi(i)}$ for $1 \leq i \leq d$. The *share* of party P_j is the subset of share components $\boldsymbol{s}_{\{P_j\}}$.

Now, the first requirement in Definition 3.4 obviously makes the scheme correct, in that a qualified set A can compute the secret by taking a linear combination of their values, since there exists $\boldsymbol{\lambda}_A \in \mathbb{Z}^{d_A}$ such that $M_A^T \cdot \boldsymbol{\lambda}_A = \boldsymbol{\varepsilon}$ which gives

$$\boldsymbol{s}_A^T \cdot \boldsymbol{\lambda}_A = (M_A \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_A = \boldsymbol{\rho}^T \cdot (M_A^T \cdot \boldsymbol{\lambda}_A) = \boldsymbol{\rho}^T \cdot \boldsymbol{\varepsilon} = s$$

The Lemma below shows that the second requirement in Definition 3.4 is sufficient to make the scheme private.

Lemma 3.1. *If $s \in [-2^l..2^l]$ is the secret to be shared and $\boldsymbol{\rho}$ is chosen uniformly at random from $[-2^{l_0+k}..2^{l_0+k}]^e$ with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$, then the LISS scheme derived from $\mathcal{M}$ is private.*

Proof. We have chosen $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$, with $\rho_i \in [-2^{l_0+k}..2^{l_0+k}]$ as uniformly random numbers for $2 \leq i \leq e$, and the secret $s \in [-2^l..2^l]$.

Let $s' \in [-2^l..2^l]$ be arbitrary. We first observe, that $\boldsymbol{s}_A = M_A\boldsymbol{\rho}$ are shares that a subset A can see. If $A \notin \Gamma$, then we by definition know that there exists a sweeping vector $\boldsymbol{\kappa}$ such that $M_A\boldsymbol{\kappa} = \mathbf{0} \in \mathbb{Z}^{d_A}$.

Define $\boldsymbol{s}' = M(\boldsymbol{\rho} + (s' - s)\boldsymbol{\kappa})$. We note that $\boldsymbol{s}'_A = \boldsymbol{s}_A$, i.e., the parties in A see the same shares, but the secret s' was shared instead of s. Define $\boldsymbol{\rho}$ to be *good* if $\boldsymbol{\rho}' = \boldsymbol{\rho} + (s' - s)\boldsymbol{\kappa}$ has entries in the specified range. Then the above implies that if we restrict the distribution of A's shares of s to the cases where $\boldsymbol{\rho}$ is good, the resulting distribution equals the one generated from s' and $\boldsymbol{\rho}'$.

It follows that the statistical distance between the distributions of A's shares of s and s' is at most twice the probability that $\boldsymbol{\rho}$ is not good, which we can estimate by the union bound as $e - 1$ times the probability that a single entry is out of range. So since $|s' - s| \leq 2^l$, the distance is at most

$$2 \cdot \frac{2^l \kappa_{\max}(e-1)}{2^{l_0+k}} \leq 2^{-k},$$

which is negligible in the security parameter k. □

A description of a LISS scheme is given by $\mathcal{L} = (\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon}), \Gamma, \mathcal{R}, \mathcal{K})$, where $\mathcal{M}$ is an ISP for Γ, for each $A \in \Gamma$ there is a reconstruction vector $\boldsymbol{\lambda}^{(A)} \in \mathcal{R}$, and for each set $B \subseteq \mathcal{P}$ with $B \notin \Gamma$ there is a sweeping vector $\boldsymbol{\kappa}^{(B)} \in \mathcal{K}$. In the following sections and chapters we will only refer to the ISP as the LISS scheme for some access structure Γ, where we implicitly assume that there is a full description of a LISS scheme as given above.

3.3 Constructions

3.3.1 Benaloh-Leichter

In this section we show how to construct an ISP based on Benaloh and Leichter Generalized Secret Sharing scheme [10]. This scheme was already shown to work for secret sharing in any finite group, but to use it over the integers, we need to revisit the scheme to make sure that the required sweeping vectors exist and estimate the size of their coordinates.

A *Boolean formula* $f\colon \{0,1\}^n \to \{0,1\}$ takes n binary inputs $x_1, \ldots, x_n$ and gives one binary output.

Definition 3.5. *A* monotone formula *on $x_1, \ldots, x_n$ is a Boolean formula f that only uses the binary connectives logical* `AND` *and logical* `OR`*.*

The operator precedence of a monotone formula is defined as follows. The parenthesis precede the `AND` operator, and the `AND` operator precede the `OR` operator. We say that an evaluation of an monotone formula $f(A)$ is `true` if and only if $f(A) = 1$, and `false` otherwise.

Example 3.1. The monotone formula,

$$f(x_1, x_2, x_3, x_4) = x_1 \vee x_2 \wedge x_3 \vee x_4,$$

is interpreted,

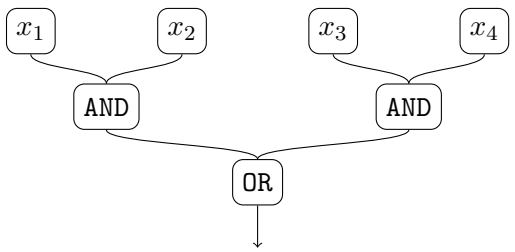
$$f(x_1, x_2, x_3, x_4) = x_1 \vee (x_2 \wedge x_3) \vee x_4.$$

We often neglect to write the arguments of a monotone formula explicitly, that is, from Example 3.1 we write

$$f = x_1 \vee (x_2 \wedge x_3) \vee x_4.$$

As pointed out in [10], there is a one-to-one correspondence between monotone access structures and monotone formulas. Every monotone access structure can be described by a monotone formula and vica versa, where each variable in the formula is associated with a party in $\mathcal{P}$. More precisely, for $i = 1, \ldots, n$ party P_i is assigned the variable x_i, we say that party P_i *owns* variable x_i. Given a monotone formula f with n Boolean variables and let $A \subseteq \mathcal{P}$, then we denote $f(A)$ to be the evaluation of $f(x_1, \ldots, x_n)$ with $x_i = 1$ if and only if $P_i \in A$. Given such a monotone formula f, then there exists a corresponding monotone access structure Γ such that $A \in \Gamma$ if and only if $f(A) = 1$, and for any monotone access structure Γ there exits a monotone formula f with $f(A) = 1$ if and only if $A \in \Gamma$.

Example 3.2. Consider the following monotone formula $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$, represented graphically below.



The corresponding monotone access structure is given by

$$\Gamma_{(f)} = \{\{P_1, P_2\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \\ \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\},$$

where we use that party P_i owns variable x_i for $i = 1, 2, 3, 4$. This can also be represented by the collection of the minimal qualified sets,

$$\Gamma^-_{(f)} = \{\{P_1, P_2\}, \{P_3, P_4\}\}.$$

In the following we will show how to construct an ISP from an arbitrary monotone formula f. Since every monotone formula can be implemented by only using logical AND- and OR-operators, it is sufficient to show how to construct a matrix representing the two operators given two matrices that express the two terms of the operator.

First we introduce some notation. Let $M^{(u)} \in \mathbb{Z}^{1,1}$ be the matrix with a single entry that is one, i.e.

$$M^{(u)} = (1)$$

We call $M^{(u)}$ and any $M^{(a)} \in \mathbb{Z}^{d_a \times e_a}$ for $0 \leq d_a, e_a$ a matrix for the convenience of consistency. Furthermore, if we have a matrix $M^{(a)} \in \mathbb{Z}^{d_a \times e_a}$, then we define $\boldsymbol{c}^{(a)} = (c^{(a)}_1, \ldots, c^{(a)}_{d_a}))^T \in \mathbb{Z}^{e_a}$ to represent the first column in $M^{(a)}$, and $R^{(a)} = [r^{(a)}_{(i,j)}] \in \mathbb{Z}^{(d_a-1)\times e_a}]$ to represent all but the first column in $M^{(a)}$. In the case where $d_a = 1$, we let R_a denote the "empty" matrix, where we by the "empty" matrix mean the matrix with no entries. Also note, that the vector $\boldsymbol{c}^{(a)}$ sometimes only represents a single entry, but we still denote it a column vector.

Given any monotone formula f for a monotone access structure Γ, we construct the distribution matrix M by the following three rules.

- Each variable x_i in the formula f can be expressed by $M^{(u)}$.
- For any OR-term $f = f^{(a)} \vee f^{(b)}$, let $M^{(a)} \in \mathbb{Z}^{d_a \times e_a}$ and $M^{(b)} \in \mathbb{Z}^{d_b \times e_b}$ be the matrices that express the formulas $f^{(a)}$ and $f^{(b)}$, respectively. Then we can construct a matrix $M^{\mathtt{OR}} \in \mathbb{Z}^{(d_a+d_b-1)\times(e_a+e_b)}$ expressing f that is defined by letting the first column of $M^{\mathtt{OR}}$ be the concatenation of the two column vectors $\boldsymbol{c}^{(a)}$ and $\boldsymbol{c}^{(b)}$, then letting the following $d_a - 1$ columns be the columns of R_a expanded with e_b succeeding zero entries, and the last $d_b - 1$ columns be the columns of R_b expanded with e_a leading zero entries. This is also visualized by,

$$M^{\mathtt{OR}} = \begin{pmatrix} c^{(a)}_1 & r^{(a)}_{(1,2)} & \ldots & r^{(a)}_{(1,e_a)} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c^{(a)}_{d_a} & r_{(d_b,2)} & \ldots & r^{(a)}_{(d_a,e_a)} & 0 & \ldots & 0 \\ c^{(b)}_1 & 0 & \ldots & 0 & r^{(b)}_{(1,2)} & \ldots & r^{(b)}_{(1,e_b)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c^{(b)}_{d_b} & 0 & \ldots & 0 & r^{(b)}_{(d_b,2)} & \ldots & r^{(b)}_{(d_b,e_b)} \end{pmatrix} \tag{3.2}$$

- For any AND-term $f = f^{(a)} \wedge f^{(b)}$, let $M^{(a)} \in \mathbb{Z}^{d_a \times e_a}$ and $M^{(b)} \in \mathbb{Z}^{d_b \times e_b}$ be the matrices that express the formulas $f^{(a)}$ and $f^{(b)}$, respectively. Then we can construct a matrix $M^{\texttt{AND}} \in \mathbb{Z}^{(d_a+d_b)\times(e_a+e_b)}$ for the formula f. It is defined by letting the first column of $M^{\texttt{AND}}$ be the column vector $\boldsymbol{c}^{(a)}$ expanded with e_b succeeding zero entries, the next column to be the concatenation of $\boldsymbol{c}^{(a)}$ and $\boldsymbol{c}^{(b)}$, the following $d_a - 1$ columns be the columns of R_a expanded with e_b succeeding zero entries, and the last $d_b - 1$ columns be the columns of R_b expanded with e_a leading zero entries. This can also be visualized by,

$$M^{\texttt{AND}} = \begin{pmatrix} c_1^{(a)} & c_1^{(a)} & r_{(1,2)}^{(a)} & \dots & r_{(1,e_a)}^{(a)} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{d_a}^{(a)} & c_{d_a}^{(a)} & r_{(d_b,2)} & \dots & r_{(d_a,e_a)}^{(a)} & 0 & \dots & 0 \\ 0 & c_1^{(b)} & 0 & \dots & 0 & r_{(1,2)}^{(b)} & \dots & r_{(1,e_b)}^{(b)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_{d_b}^{(b)} & 0 & \dots & 0 & r_{(d_b,2)}^{(b)} & \dots & r_{(d_b,e_b)}^{(b)} \end{pmatrix} \tag{3.3}$$

For the sake of clarity we provide a simple example here to demonstrate the procedure.

Example 3.3. Let

$$f = ((x_1 \wedge x_2) \wedge (x_3 \vee x_4)).$$

Each variable in f can be expressed by the matrix $M^{(u)}$. If we let $f = f^{(a)} \wedge f^{(b)}$ where $f^{(a)} = (x_1 \wedge x_2)$ and $f^{(b)} = (x_3 \vee x_4)$, then we can express $f^{(a)}$ by the matrix

$$M^{(a)} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix},$$

by using the AND-rule on x_1 and x_2, which both are expressed by $M^{(u)}$. Furthermore, we can express $f^{(b)}$ by the matrix

$$M^{(b)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

by using the OR-rule on x_3 and x_4, which also are expressed by $M^{(u)}$. Then finally we can express f by the matrix

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

by using the AND-rule on $f^{(a)}$ and $f^{(b)}$ and their respectively matrix representations $M^{(a)}$ and $M^{(b)}$.

Note that each occurrence of a variable x_i in the formula f is represented by a row in the resulting matrix. We say that a given row is *owned* by the variable that it represents or the party that owns the variable. Furthermore, note that a variable can own more than one row in the resulting matrix. This happens when a variable x_i is represented more than once in the formula f.

Example 3.4. Consider the formula

$$f = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3),$$

which represents the threshold 2 out of 3 access structure, i.e.,

$$\Gamma^- = \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\},$$

where party P_i owns variable x_i for $i = 1, 2, 3$. Using the above construction, this will result in the following matrix of the ISP,

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where row 1 and 3 is owned by party P_1 (variable x_1), row 2 and 5 is owned by party P_2 (variable x_2), and finally row 4 and 6 is owned by party P_3 (variable x_3).

In the following we show that $(M, \psi, \boldsymbol{\varepsilon})$ defines an ISP, where M is the matrix defined above, ψ is the function that maps share components to the parties, and $\boldsymbol{\varepsilon}$ is the target vector. Note that ψ is easily defined from the monotone formula and the corresponding matrix M, it simply maps a row in the matrix to the party that owns it.

Example 3.5. Proceeding with Example 3.4 the ψ function would be given by

row	1	2	3	4	5	6
$\psi(\cdot)$	P_1	P_2	P_1	P_3	P_2	P_3

We proceed to show that $(M, \psi, \boldsymbol{\varepsilon})$ is an ISP for the given access structure Γ in the following two lemmas. Note, that in the first lemma we give the construction of the reconstruction vector and the following lemma give the construction of the sweeping vector.

Lemma 3.2. *If $A \in \Gamma$, then there exits a reconstruction vector $\boldsymbol{\lambda}_A$ such that $M_A^T \boldsymbol{\lambda}_A = \boldsymbol{\varepsilon}$.*

Proof. We show how to construct the reconstruction vector $\boldsymbol{\lambda}_A$ by induction in the number of variables in the formula f that represents the access structure Γ. If $M \in \mathbb{Z}^{d \times e}$, then we construct the reconstruction vector $\boldsymbol{\lambda} \in \mathbb{Z}^d$ for $A \in \Gamma$, such that $M^T \boldsymbol{\lambda} = \boldsymbol{\varepsilon}$ but $\boldsymbol{\lambda}_{A^{\complement}} = \mathbf{0}$, i.e., we do not use rows of M that are not owned by some party in A.

In the base case, $f = x$, there is only one party, and the distribution matrix is $M = (1)$, i.e., the party gets the secret s, so the reconstruction vector is $\boldsymbol{\lambda} = (1)^T$.

In the case of an `OR`-term, $f = f^{(a)} \vee f^{(b)}$, then let $M^{(a)}$ and $M^{(b)}$ be the matrices that represent the the formulas $f^{(a)}$ and $f^{(b)}$, respectively. By

assumption we know that one of the matrices is able to reconstruct the secret. Take the reconstruction vector belonging to the matrix able to to reconstruct the secret, e.g., $\boldsymbol{\lambda}_a = (\lambda_1, \ldots, \lambda_t)^T$. The new reconstruction vector for the `OR`-term is $\boldsymbol{\lambda}_{\texttt{OR}} = (\lambda_1, \ldots, \lambda_t, 0, \ldots, 0)^T$, i.e. we put zeros in the entries that represent the shares from $M^{(b)}$.

In the case where there is an `AND`-term, $f = f^{(a)} \wedge f^{(b)}$, then let the matrices $M^{(a)}$ and $M^{(b)}$ represent the formulas $f^{(a)}$ and $f^{(b)}$, respectively. By assumption each of the matrices $M^{(a)}$ and $M^{(b)}$ can reconstruct their part of the secret, let $\boldsymbol{\lambda}_a = (\lambda_{a_1}, \ldots, \lambda_{a_t})^T$ and $\boldsymbol{\lambda}_b = (\lambda_{b_1}, \ldots, \lambda_{b_v})^T$ be the reconstruction vectors for $M^{(a)}$ and $M^{(b)}$. Then we claim that the reconstruction vector for the `AND`-term is

$$\boldsymbol{\lambda}_{\texttt{AND}} = (\lambda_{a_1}, \ldots, \lambda_{a_t}, -\lambda_{b_1}, \ldots, -\lambda_{b_v})^T,$$

because, if we define

$$\begin{aligned} \boldsymbol{\lambda}_{a'} &= (\lambda_{a_1}, \ldots, \lambda_{a_t}, 0, \ldots, 0)^T \\ \boldsymbol{\lambda}_{b'} &= (0, \ldots, 0, \lambda_{b_1}, \ldots, \lambda_{b_v})^T, \end{aligned}$$

we know that if M is the distribution matrix that represents the `AND`-term (see Equation 3.3), then

$$\begin{aligned} (M \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_{a'} &= s + \rho_2 \\ (M \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_{b'} &= \rho_2, \end{aligned}$$

i.e., we have that

$$\begin{aligned} (M \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_{a'} - (M \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_{b'} &= (M \cdot \boldsymbol{\rho})^T \cdot (\boldsymbol{\lambda}_{a'} - \boldsymbol{\lambda}_{b'}) \\ &= (M \cdot \boldsymbol{\rho})^T \cdot \boldsymbol{\lambda}_{\texttt{AND}} \\ &= s + \rho_2 - \rho_2 = s, \end{aligned}$$

which concludes the proof. □

Before we proceed with the following lemma we make some observations. First of all, the ISP for the formula,

$$f = x_1 \vee x_2 \vee \cdots \vee x_n,$$

does not have any sweeping vector, since there exist no unqualified non-empty subset of parties $A \subset \mathcal{P}$.

Hence, the simplest case is for a formula on the form,

$$f = (x_1 \vee \cdots \vee x_i) \wedge (x_{i+1} \vee \cdots \vee x_j),$$

which we will call the base case.

Lemma 3.3. *For all $A \notin \Gamma$ there exists a sweeping vector $\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_e)^T$ such that $M_A \cdot \boldsymbol{\kappa} = \mathbf{0}$ with $\kappa_1 = 1$.*

Proof. We show how to construct a sweeping vector $\boldsymbol{\kappa}$ by induction on the construction of the matrix from a formula f that represents the access structure Γ. Note, that we only need to verify that the inner product of $\boldsymbol{\kappa}$ is zero with the rows from M that are owned by the parties in A.

By the above observation, we have that the base case is a formula of the form $f = (x_1 \vee \cdots \vee x_i) \wedge (x_{i+1} \vee \cdots \vee x_j)$, where there is only one AND-term. The variables in f representing the parties in A cannot be on both sides of the AND-term. If a party P_i in A owns a variables on both sides of the AND-term, then P_i would obviously be qualified to reconstruct the secret, which contradicts the assumption that $A \notin \Gamma$. Note, that this also implies that the parties in A cannot own rows in both the upper and lower part of the matrix constructed in Equation (3.3). The sweeping vector can be constructed as follows, if the parties in A own rows is in the bottom of the matrix (Equation (3.3)), then $\boldsymbol{\kappa} = (1, 0, \ldots, 0)^T$ would qualify as sweeping vector, whereas if the parties own rows in the top of the matrix (Equarion (3.3)), then $\boldsymbol{\kappa} = (1, -1, 0, \ldots, 0)^T$ would qualify as sweeping vector.

The induction step with an OR-term, $f = f^{(a)} \vee f^{(b)}$, where we have matrices $M^{(a)}$ and $M^{(b)}$ that represents the formulas $f^{(a)}$ and $f^{(b)}$, respectively. There are two cases to consider, first if both the formulas $f^{(a)}$ and $f^{(b)}$ contain at least one AND-term, then by the induction assumption there exists $\boldsymbol{\kappa}_a$ and $\boldsymbol{\kappa}_b$ such that $(M^{(a)})_A \cdot \boldsymbol{\kappa}_a = \mathbf{0}$ and $(M^{(b)})_A \cdot \boldsymbol{\kappa}_b = \mathbf{0}$. In this case, we define $\boldsymbol{\kappa} = (1, \kappa_{a_2}, \ldots, \kappa_{a_t}, \kappa_{b_2}, \ldots, \kappa_{b_v})^T$, which from Equation (3.2) can be argued to work, because if we take the inner product with $\boldsymbol{\kappa}$ on an upper row owned by a party in A of the matrix (Equation (3.2)), then it will be 0 by definition of $\boldsymbol{\kappa}_a$, and if take the inner product with $\boldsymbol{\kappa}$ on a lower row owned by a party in A of the matrix (Equation (3.2)), then it will be 0 by definition of $\boldsymbol{\kappa}_b$. If only one of the formulas, $f^{(a)}$ or $f^{(b)}$, has an AND-term, say $f^{(a)}$, then neither of the forbidden parties can have variables in $f^{(b)}$, since this would enable them to reconstruct the secret. That is, we need not to worry about the inner product of $\boldsymbol{\kappa}$ with the rows in the lower part of Equation (3.2). A similar argument holds if only $f^{(b)}$ contains AND-terms.

The induction step where we have an AND-term, $f = f^{(a)} \wedge f^{(b)}$, and let $M^{(a)}$ and $M^{(b)}$ be the matrices that represent the formulas $f^{(a)}$ and $f^{(b)}$, respectively. Then the parties in A will either not be able to reconstruct the secret from $f^{(a)}$ or $f^{(b)}$. Observe from Equation (3.3) that if the parties from $A \notin \Gamma$ do not qualify reconstruct the secret from $f^{(a)}$, then by the induction assumption there exists $\boldsymbol{\kappa}_a$ such that $(M^{(a)})_A \cdot \boldsymbol{\kappa}_a = 0$, and we can use $\boldsymbol{\kappa} = (1, 0, \kappa_{a_2}, \ldots, \kappa_{a_t}, 0, \ldots, 0)^T$ as sweeping vector. However, if the parties in A can reconstruct the secret from $f^{(a)}$, then we use $\boldsymbol{\kappa} = (1, -1, 0, \ldots, 0, -\kappa_{b_2}, \ldots, -\kappa_{b_v})^T$, where $\boldsymbol{\kappa}_b = (1, \kappa_{b_2}, \ldots, \kappa_{b_v})^T$ is the sweeping vector for $(M^{(b)})_A$, which exists by assumption. □

We summarize the above two lemmas in the following theorem.

Theorem 3.1. *Given any monotone access structure Γ, then a LISS scheme for Γ can efficiently constructed.*

We proceed to make some observations from the construction of the LISS scheme $\mathcal{L} = (\mathcal{M} = (M, \psi, \varepsilon), \Gamma, \mathcal{R}, \mathcal{K})$ that we will need later.

Given any LISS scheme $\mathcal{L} = (\mathcal{M} = (M, \psi, \varepsilon), \Gamma, \mathcal{R}, \mathcal{K})$ constructed as above, let f denote the monotone formula for Γ. Let $\text{depth}(f)$ denote the greatest depth of the formula and $\text{op}(f)$ the number of `AND`- and `OR`-operators in f.

Remark 3.1. If we look at Equation (3.2) and (3.3), it follows that, if m_{nz} is the number of non-zero entries in a row in the matrix M, then $m_{\text{nz}} \leq \text{depth}(f) + 1$.

Remark 3.2. Each non-zero entry in the matrix M is 1.

Remark 3.3. It follows from Equation (3.2) and (3.3), that if $M \in \mathbb{Z}^{d \times e}$ then $d \leq \text{op}(f) + 1$ and $e \leq \text{op}(f) + 1$.

From Remark 3.1 and 3.2 it follows, that if we have distribution matrix $M \in \mathbb{Z}^{d \times e}$ that represents a formula with depth at most $\text{depth}(f)$, then we at most need $d \cdot \text{depth}(f)$ additions to calculate all the d shares component from Equation (3.1). Each share component is at most the addition of $\text{depth}(f)$ integers of (l_0+k)-bit, i.e., each share component is at most $l_0+k+\log(\text{depth}(f))$ bits long.

Remark 3.3 gives that d is equal to the size of the formula that the matrix M represents.

In [86] Valiant shows the existence of a monotone formula for the majority function of size $\mathcal{O}\left(n^{5.3}\right)$ and of depth $\mathcal{O}(\log n)$. A threshold-t function $T_{t,n}$ can be constructed from the majority function, by fixing some of the inputs of the majority function. This construction implies that we need a majority function of size at most $2n$ to construct the threshold-t function $T_{t,n}$, i.e. [86] gives the existence of a monotone formula for the threshold-t function $T_{t,n}$ of size $\mathcal{O}\left(n^{5.3}\right)$ and of depth $\mathcal{O}(\log n)$.

This result was improved by Hoory et al. in [62], where they give a monotone formula of size $\mathcal{O}\left(n^{1+\sqrt{2}}\right)$ and depth $\mathcal{O}(\log n)$ for the majority function, and hence, for the threshold-t function $T_{t,n}$.

In the table below, we give the share size, computation time, and the number of random bits needed for constructing a threshold-t access structure, when based on the result of [62].

	Complexity
Share size (bits)	$\mathcal{O}\left(n^{\sqrt{2}}(l_0 + k + \log\log n)\right)$
Computation time	$\mathcal{O}\left(n^{1+\sqrt{2}} \log n(l_0 + k + \log\log n)\right)$
Random bits	$\mathcal{O}\left(n^{1+\sqrt{2}}(l_0 + k)\right)$

Where we assume in the computation time, that it takes $\mathcal{O}(b)$ time to add two $\mathcal{O}(b)$ bit numbers.

3.3.2 Cramer-Fehr

In this section we consider the ISP's constructed by Cramer and Fehr in [32].

As described, if we have an ISP $\mathcal{M} = (M, \psi, \varepsilon)$ we use $M \in \mathbb{Z}^{d \times e}$ as the distribution matrix and we calculate the shares from Equation (3.1). Let $m_{\max}$

denote the bit-length of the maximal entry in the distribution matrix M. Then we need $d \cdot e$ multiplications of $\mathcal{O}(l_0 + k + m_{\max})$-bit numbers and $d \cdot (e-1)$ additions of $\mathcal{O}(l_0 + k + m_{\max} + e)$-bit numbers to calculate the shares.

From the proof of Corollary 1 in [32] we have that for a threshold-t access structure $T_{t,n}$ that

$$\begin{aligned} d &= n(\lfloor \log n \rfloor + 2) \\ e &= t(\lfloor \log n \rfloor + 2) + 1 \end{aligned}$$

We also know, that $m_{\max} = \mathcal{O}(n)$. If we assume that we use $\mathcal{O}(b)$ time to choose a b-bit random number, $\mathcal{O}(b)$ time to add two b-bit numbers, and $\mathcal{O}\left(b \log^2 b\right)$ time to multiply two b-bit numbers. Then we need

$$\begin{aligned} &\mathcal{O}\left(tn \log^2 n(l_0 + k + n) \log^2(l_0 + k + n) + tn \log^2 n(l_0 + k + n + t \log n)\right) \\ &= \mathcal{O}\left(tn \log^2 n(l_0 + k + n) \log^2(l_0 + k + n)\right) \end{aligned}$$

time to compute the shares. Furthermore, we have that each share component is of size $\mathcal{O}(l_0 + k + n + \log(t \log n)) = \mathcal{O}(l_0 + k + n)$-bit, hence the average share size is $\mathcal{O}(\log n(l_0 + k + n))$.

3.3.3 Comparison

In this section we compare the constructions based on Benaloh-Leichter with Cramer-Fehr for the LISS scheme in the threshold-t case. We consider the average share size, the computation complexity to generate shares, and the number of random bits required to do the computations of shares.

First we make some observations. Recall that $l_0 = l + \lceil \log_2(\kappa_{\max}(e-1)) \rceil + 1$. In the Benaloh-Leichter construction we get that $l_0 = l + \lceil \log_2(n^{1+\sqrt{2}} - 1) \rceil + 1$, which asymptotically reduces to $l_0 \in \mathcal{O}(l + \log n)$. In the Cramer-Fehr construction we have that $\kappa_{\max} = c2^n$ and $e = t(\lfloor \log n \rfloor + 2) + 1$, i.e., $l_0 = l + \lceil \log_2(c2^n t(\lfloor \log n \rfloor + 2)) \rceil + 1$, which asymptotically reduces to $l_0 \in \mathcal{O}(l + n)$.

First consider the share sizes in each of the constructions of a threshold-t scheme $T_{t,n}$, with secret size of bit length at most l, and statistical security parameter k.

Construction	**Share size**
Cramer-Fehr	$\mathcal{O}((l + k + n) \log n)$
Benaloh-Leichter	$\mathcal{O}\left((l + k + \log \log n) n^{\sqrt{2}}\right)$

In a normal setting the l and k parameters will be the dominating factors. Ignoring the hidden constants, the slight advantage goes to the Cramer-Fehr construction for a normal set of parameters.

Considering the local computation time to generate a set of shares, then the advantage goes to the Benaloh-Leichter construction.

Construction	**Local computation time**
Cramer-Fehr	$\mathcal{O}\left(tn \log^2 n(l + k + n) \log^2(l + k + n)\right)$
Benaloh-Leichter	$\mathcal{O}\left(n^{1+\sqrt{2}} \log n(l + k + \log \log n)\right)$

This is due to the fact, that the Benaloh-Leichter construction only uses additions to generate the shares, whereas the Cramer-Fehr construction uses expensive multiplications.

Consider the number of random bits needed to generate a set of shares, then the Cramer-Fehr construction has the advantage.

Construction	**Random bits**
Cramer-Fehr	$\mathcal{O}\left((l+k+n)t\log n\right)$
Benaloh-Leichter	$\mathcal{O}\left((l+k+\log n)n^{1+\sqrt{2}}\right)$

Results of Radhakrishnan [79] show that the lower bound for a monotone formula that computes the threshold-t function $T_{t,n}$ for $2 \leq t \leq \frac{n}{2}$, has size at least

$$\left\lfloor \frac{t}{2} \right\rfloor n \log\left(\frac{n}{t-1}\right).$$

As he notes, that in the monotone formulas model, the complexities of computing $T_{t,n}$ and $T_{n-t+1,n}$ are the same. In [62] Hoory et al. give a construction of the threshold-t function of size,

$$t^2 n \log\left(\frac{n}{t}\right).$$

Hence, not so far from the shown lower bound. Note that this result might turn out to be more efficient than the one presented in the comparisons.

To summarize the results of this section, we find that Cramer-Fehr construction seems better in the general case of the threshold-t function. However, remember that in most normal set of parameters this conclusion relies greatly on the hidden parameters in the big-Oh notation.

The result of Radhakrishnan gives an even larger gab for improvements from the current state of the art of threshold functions that would favor the Benaloh-Leichter construction. Finally, it must be stressed, of course, that the Benaloh-Leichter construction has the advantage that it can be used over any monotone access structure. However, the construction is only efficient if there is a polynomial-size monotone formula describing the access structure.

Part II

The Computational Model

Chapter 4

Distributed Exponentiation

4.1 Introduction

In this chapter we show that any LISS scheme can be used to build a distributed exponentiation protocol. The protocol does not use multilevel secret sharing. Thus, it can be made non-interactive using any of the known techniques for this purpose, such as the Fiat-Shamir heuristic (the random oracle model) or [28,36,56]. Furthermore, no party, including the dealer, needs to know the order of the group involved. This implies that we obtain the first non-interactive distributed exponentiation protocol that works for any group and any access structure.

We also look at the particular case of distributed RSA. We generalize the results of Damgård and Dupont [35] to arbitrary access structures, and thus obtain a distributed RSA signature scheme for any access structure, any public exponent and any modulus, efficiently and in constant-round without using random oracles or any assumptions other than the RSA assumption.

We emphasize that our result that all LISS schemes can be used for distributed exponentiation does not hold for BBSS schemes, not even if we assume that the dealer knows the group order[1]. The reason for this is that in order to do the proof of security for an exponentiation protocol using known simulation techniques, the secret sharing scheme needs to have the so called *share completion property:* given an unqualified set of shares and the secret, we can compute by linear combinations a *complete* set of shares consistent with what we were given. It is not known whether BBSS or LISS schemes have this property in general, in fact the answer is probably no. Here, we get around this problem by coming up with a different simulation technique where share completion is not needed. This technique always works with a LISS scheme, but fails with BBSS when the group order is not public.

[1]We note that the BBSS constructions of [32, 45] are in fact applicable to distributed exponentiation, but this is due to special properties of those constructions.

4.2 Distributed Exponentiation

In this section we will consider solutions to the the distributed exponentiation problem based on LISS. The set-up is as follows: we have n servers $P_1, ..., P_n$, an access structure Γ with an ISP $\mathcal{M} = (M, \psi, \varepsilon)$, and an adversary $\mathcal{A}$ who may corrupt any subset of servers not in Γ. Denote the corresponding adversary structure Δ. Finally, we have a special party C called the client who may also be corrupted, independently of which servers are corrupt

In this first solution we consider non-adaptive corruption in the semihonest model, i.e., the adversary must choose which parties to corrupt before the protocol starts, he sees all internal data and communication of corrupt parties, he may cause them to stop playing at any time, but all parties follow the protocol as long as they participate. In order to solve the problem in this model, we must assume that the adversary structure is Q2, i.e., any set of form $A \cup B$, $A, B \in \Delta$ is strictly smaller than $\mathcal{P} = \{P_1, ..., P_n\}$. This ensures that the set of honest servers is in Γ.

We will use Canetti's Universal Composability (UC) framework to state and prove our protocols. For details on this framework, refer to Section 2.1 and [19]. In order to focus on the actual protocol for exponentiation, we will assume a trusted dealer who chooses the group to use and secret-shares the exponent. In the UC framework, this means we assume a functionality representing the dealer is given, as detailed below. We assume for simplicity synchronous communication and also that the client C can broadcast information to all servers. However, we do not assume any private channels so all communication between parties is seen by the adversary.

Functionality $\mathcal{F}_{\texttt{Deal}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving (Start, sid) from all honest parties:
   (a) Choose the group G to use and an exponent s (in principle any efficient algorithm for this could be used here).
   (b) Generate the distribution vector $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$ and calculate the shares from
   $$M \cdot \boldsymbol{\rho} = (s_1, \ldots, s_d)^T,$$
   finally distributes the shares, such that s_i is sent privately to $P_{\psi(i)}$ for $1 \leq i \leq d$. Finally, send a description of G to all parties and the adversary (information allowing to represent group elements and computing the group operation).

Figure 4.1: Functionality $\mathcal{F}_{\texttt{Deal}}$.

The $\mathcal{F}_{\texttt{Deal}}$ functionality given in figure 4.1 together with the protocol we give below will implement the $\mathcal{F}_{\texttt{Exp}}$ functionality given in figure 4.2

Functionality $\mathcal{F}_{\texttt{Exp}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving (Start, sid) from all honest parties:
   (a) Choose the group G to use and an exponent s (same algorithm as used in $\mathcal{F}_{\texttt{Deal}}$).
   (b) Send a description of G (information allowing to represent group elements and computing the group operation) to all parties and the adversary.
2. At any later time, upon receiving $(\mathsf{Exponentiate}, sid, a, G)$ for from the client:
   (a) Send $(\mathsf{Exponentiate}, a, G)$ to all parties and the adversary.
   (b) In the next round, send (Result, a^s) to the client and the adversary.

Figure 4.2: Functionality $\mathcal{F}_{\texttt{Exp}}$.

The protocol proceeds as follows:

Protocol $\mathsf{Exponentiate}(a)$

Initially, each party sends (Start, sid) to $\mathcal{F}_{\texttt{Deal}}$, and stores the description of G and shares of s received from $\mathcal{F}_{\texttt{Deal}}$.

1. On input $a \in G$, C broadcasts a to the servers.
2. Each P_j sends to C $a_i = a^{s_i}$ for each component s_i of the share held by P_j.
3. Since Δ is Q2, C is guaranteed to receive valid contributions from a qualified set of parties $A \in \Gamma$. C uses the entries in the reconstruction vector for A $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{d_A})^T$ together with the contributions $(a_1 = a^{s_1}, \ldots, a_{d_A} = a^{s_{d_A}})$ to construct

$$a^s = \Pi_{i=1}^{d_A} a_i^{\lambda_i}.$$

Theorem 4.1. *The Exponentiate protocol when given access to $\mathcal{F}_{\texttt{Deal}}$, and a broadcast channel from C to the servers, securely implements $\mathcal{F}_{\texttt{Exp}}$. The adversary is assumed to non-adaptively corrupt any set in Q2 structure Δ in a semi-honest fashion.*

Proof. Security is proved by constructing an ideal model adversary (also called simulator) S that works in a setting where it may communicate with the ideal functionality $\mathcal{F}_{\texttt{Exp}}$ and must simulate everything the real life adversary $\mathcal{A}$ would see in a real attack. This works by running internally a copy of $\mathcal{A}$ and proceeds as follows:

1. Let B be the set of servers corrupted by $\mathcal{A}$. Having received the description of G from $\mathcal{F}_{\texttt{Exp}}$, compute a sharing of 0 to simulate the action of $\mathcal{F}_{\texttt{Deal}}$, i.e., the distribution vector is $\boldsymbol{\rho} = (0, \rho_2, \ldots, \rho_e)^T$ and the shares are
$$\boldsymbol{s} = (s_1, \ldots, s_d)^T = M \cdot \boldsymbol{\rho} \tag{4.1}$$
Give to the $\mathcal{A}$ the shares from (4.1) belonging to the servers in B.

2. Upon receiving $(\mathsf{Exponentiate}, a, G)$ and (Result, a^s) from $\mathcal{F}_{\texttt{Exp}}$, we must simulate the contributions that honest parties send to C. To this end, note that if we had used $\boldsymbol{\rho}' = \boldsymbol{\rho} + s\boldsymbol{\kappa}_B$ as distribution vector in (4.1), then the corrupted servers in B would get the same shares, but the secret value would be s instead of 0.

   Now, let R be a row in the distribution matrix M belonging to honest server P_j, say the i'th row, and let s_i be the share component we computed from this row in (4.1). Had we used $\boldsymbol{\rho}'$ instead of $\boldsymbol{\rho}$, then the share component coming from R would have been $s_i' = (\boldsymbol{\rho}+s\boldsymbol{\kappa}_B)\cdot R = s_i + s\boldsymbol{\kappa}_B \cdot R$ instead. The observation is now that because we know a^s and s_i, we can compute $a^{s_i'}$ even though we do not know s. Concretely, we simulate the contribution from P_j by
$$\begin{aligned} a^{s_i}(a^s)^{\boldsymbol{\kappa}_B \cdot R} &= a^{s_i + s\boldsymbol{\kappa}_B \cdot R} \\ &= a^{s_i'} \end{aligned}$$
   Give all simulated contributions to $\mathcal{A}$.

We now need to prove that no environment $\mathcal{Z}$ can distinguish between the real process and the ideal process. This is straightforward: First, the shares computed in step 1 of the simulation are statistically indistinguishable from the shares computed by $\mathcal{F}_{\texttt{Deal}}$ by the privacy of the LISS scheme and since B is unqualified. Second, in both the ideal and real process, honest parties always output the correct value a^s, by definition of $\mathcal{F}_{\texttt{Exp}}$, respectively correctness of the LISS scheme. Finally, given a, a^s, the simulated and real contributions from honest parties are statistically indistinguishable, since the vector we use for the simulated sharing is $\boldsymbol{\rho}' = \boldsymbol{\rho} + s\boldsymbol{\kappa}_B$ which is statistically close to a uniformly chosen sharing vector for s. □

4.3 Active Adversaries and Distributed RSA

If we are not guaranteed that corrupted parties follow the protocol, we can expand the Exponentiate protocol in a natural way by having parties prove in zero-knowledge that their contributions are correct. Given any appropriate scheme for proving correctness of contributions, a corrupt party must either give correct information or be disqualified. Since this is equivalent to the semihonest model, security essentially follows from security of the zero-knowledge proofs and the proof we already gave above.

Depending on the structure of the group and the assumptions we are willing to make, there are many different ways to do the zero-knowledge proofs, see

for instance [28, 35, 36, 39, 56, 80, 81, 83]. Most of the techniques can be made non-interactive in the random oracle model, or are already non-interactive given some set-up assumption. If all else fails, generic zero-knowledge techniques can be used [58].

However, a detailed account of all possibilities is out of scope of this thesis. We concentrate instead on distributed RSA as a particularly interesting special case. The functionality for initial set-up and the functionality we want to implement are modified from the general case to the $\mathcal{F}_{\texttt{RSA,Deal}}$ functionality given in figure 4.3 and the $\mathcal{F}_{\texttt{RSA}}$ functionality given in figure 4.4, respectively.

Functionality $\mathcal{F}_{\texttt{RSA,Deal}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving (Start, sid) from all honest parties:
   (a) Choose $N = pq$ for primes p and q, secret and public exponent (s, e), and a random square $v \in \mathbb{Z}_N^*$
   (b) Generate the distribution vector $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^T$ and calculate the shares from
   $$M \cdot \boldsymbol{\rho} = (s_1, \ldots, s_d)^T,$$
   finally distributes the shares, such that s_i is sent privately to $P_{\psi(i)}$ for $1 \leq i \leq d$. Finally, send $((N, e), v)$ and $v_i = v^{s_i} \bmod N$ for every share component s_i to all parties and the adversary .

Figure 4.3: Functionality $\mathcal{F}_{\texttt{RSA,Deal}}$.

The protocol we will use is the Exponentiate protocol from the previous section, with the extension that C will check each contribution $a_i = a^{s_i} \bmod N$ from server P_j. We want to show that a sufficient check can be done in constant-round without using random oracles to ensure soundness and zero-knowledge, and regardless of which modulus and public exponent is used. To do this, we generalize the results from [35]. Concretely, we use the following well known protocol, which we will repeat in parallel $\lceil 2 + 2\log_2 n \rceil$ times:

Protocol ZK − proof

Initially, each party sends (Start, sid) to $\mathcal{F}_{\texttt{RSA,Deal}}$, and stores the description of $((N, e), v)$ and shares of s along with the v_i's received from $\mathcal{F}_{\texttt{RSA,Deal}}$.

1. P_j chooses a random $k +$ max-bit number r and sends to C $u_1 = a^r \bmod N, u_2 = v^r \bmod N$. Here, max is the maximal bit-length of any s_i that can occur.
2. C sends a random bit b to P_j.
3. P_j sends $z = r + bs_i$, and C checks that $a^z = u_1 a_i^b \bmod N, v^z = u_2 v_i^b \bmod N$

Functionality $\mathcal{F}_{\texttt{RSA}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving (Start, sid) from all honest parties:
   (a) Choose the modulus N to use, secret and public exponents s, e.
   (b) Send N, e to all parties and the adversary.
2. At any later time, upon receiving $(\mathsf{Exponentiate}, sid, a)$ for all $a \in \mathbb{Z}_N^*$ from the client:
   (a) Send $(\mathsf{Exponentiate}, a)$ to all parties and $(\mathsf{Exponentiate}, a, a^s \bmod N)$ to the adversary.
   (b) Wait two rounds and send $(\mathsf{Result}, a^s \bmod N)$ to the client and all parties.

Figure 4.4: Functionality $\mathcal{F}_{\texttt{RSA}}$.

The following Lemma is an easy consequence of corresponding results in [35]:

Lemma 4.1. *The above protocol is statistical zero-knowledge. Furthermore, if $a_i \neq a^s \bmod N$ then a polynomial time prover who can make C accept with probability more than $1/(4n^2)$ can compute efficiently a multiple of the order of v.*

Note that the last result in the lemma implies that if an adversary can cheat the protocol on input a random v, he can factor N by a standard reduction and hence also break RSA.

Even though the soundness error for this protocol is not negligible, we can show that checking the contributions in this way is sufficient to allow C to reconstruct the correct result efficiently. This is done by a generalization of the results from [35]. There it was observed that as long as the expected number of accepted incorrect contributions is small enough, C can reconstruct efficiently by searching exhaustively for a set of correct contribution. In [35], this was done for the case of a threshold access structure. Here we have to be more careful with the search algorithm and the analysis because we have no lower bound on the number of honest parties for a general access structure.

Algorithm Reconstruct

On input public key N, e, $a \in Z_N^*$ and a set of contributions to finding $a^s \bmod N$, execute the $\mathsf{ZK-proof}$ protocol with each server to check the correctness of each contribution.

1. Let the set of accepted contributions be Acc. Do the following for $j = 0, ..., |Acc|$:
2. For each subset $B \subset Acc$ of size $|Acc| - j$, run the reconstruction algorithm from the $\mathsf{Exponentiate}$ protocol on the contributions in B,

attempting to compute $a^s \bmod N$. Let z be the result. If $z^e = a \bmod N$, output z and stop.

Lemma 4.2. *The expected number of subsets considered by Reconstruct is at most 2.*

Proof. Let m be the number of incorrect contributions submitted by corrupt parties. Clearly, the worst case is if all corrupt parties submit bad contributions, so we may assume that the number of honest parties is $n - m$. Let p be the probability that an incorrect contribution is accepted. Then

$$p_i = Pr(i \text{ incorrect shares accepted}) = p^i(1-p)^i\binom{m}{i} \leq p^i m^i$$

Given that i incorrect shares are accepted, we have $n-m+i$ contributions, and we finish at the latest when we have searched all subsets of size $n-m$. This means checking a total of

$$\begin{aligned}\binom{n-m+i}{n-m+i} + \binom{n-m+i}{n-m+i-1} + ... + \binom{n-m+i}{n-m} &\leq (i+1)\binom{n-m+i}{n-m} \\ &= (i+1)\binom{n-m+i}{i} \\ &\leq (i+1)(n-m+i)^i\end{aligned}$$

subsets. It follows that the expected number of subsets we check is at most

$$\sum_{i=0}^{m} p^i m^i \cdot (i+1)(n-m+i)^i \leq \sum_{i=0}^{m} p^i m^i 2^i n^i \leq \sum_{i=0}^{m} (2pn^2)^i \leq \sum_{i=0}^{m} 2^{-i} \leq 2$$

using the above and the fact that $p \leq 1/(4n^2)$. □

A final observation is that by choosing z at random in Z_N^*, and setting $v = z^{2e} \bmod N$, a simulator can easily create a random square v for which $v^s \bmod N$ is known (namely $z^2 \bmod N$). It is then easy to simulate the information $\mathcal{F}_{\texttt{RSA,Deal}}$ sends to corrupt parties. Using this, the proof of Theorem 4.1, Lemma 4.2 and Lemma 4.1, it is straightforward to show:

Theorem 4.2. *Under the RSA assumption, the* Exponentiate *protocol expanded with the above Reconstruction algorithm and given access to the $\mathcal{F}_{\texttt{RSA,Deal}}$ functionality implements the $\mathcal{F}_{\texttt{RSA}}$ functionality. The adversary may non-adaptively and actively corrupt any set in Q2 structure Δ.*

We believe that the interest of this result is that it buys us full generality in access structure and choice of keys and no dependency on extra set-up or complexity assumptions. Since the number of servers n can be expected to be quite small in practice, the overhead compared to more standard solutions is moderate: a factor of $\log n$ in complexity and potentially 2 extra moves. However, in practice, faults are usually rare, so if the the client attempts to get the result from all contributions first and only asks to have the proofs completed if this fails, then the scheme will be non-interactive "almost always".

Chapter 5

Non-Interactive VSS and Multiplication Proofs

5.1 Introduction

Applications such as auctions, elections or benchmarking analysis all involve computing on confidential data from several parties who do not trust each other a priori. This means that solutions involving a single trusted party are typically unsatisfactory. In principle, all such problems can be solved using general secure multiparty computation [9, 24, 57], where all parties take part in computing the desired results. However, in practice, this is often not realistic: in auctions or elections, for instance, the number of parties holding inputs can be very large, they cannot be assumed to be expert users nor can their machines be assumed to be on-line at particular times. Hence assuming that all such parties can reliably take part in a multi-round protocol is unrealistic.

It is therefore often suggested that a smaller number of servers are assigned to do the computation, acting effectively as representatives for the clients supplying inputs. Of course, this makes sense only if the complexity of supplying inputs is much smaller than the complexity of taking part in the actual computation. In particular, we would want that supplying inputs is non-interactive. This problem can be solved using a non-interactive verifiable secret sharing (VSS) scheme. Having done the VSS's, the servers hold shares of all inputs and can do the computation using any of the (numerous) known multiparty computation techniques. Several non-interactive VSS protocols are known see, e.g., [77].

However, many applications require that the inputs supplied satisfy certain constraints. These constraints are typically phrased in a natural way as relations over the integers, because the underlying application is a computation on integers. This is the case for auctions, elections and many statistical applications such as benchmarking. For instance, an auction might specify that bids have to be in a certain interval. In other types of auctions (so called double auctions [11]), a bid consists of a sequence of numbers that must be monotonely increasing.

Standard efficient techniques for handling this would have a client commit to his input and prove in zero-knowledge that his numbers satisfy the required

relations. However, this solution requires interaction in its basic form. The interaction can typically be removed following the Fiat-Shamir heuristic if we are willing to assume the random oracle model. However, it is well known that the security guarantee provided by a proof in the random oracle model leaves something to be desired: we cannot instantiate the oracle with a concrete function and be sure that this always works. Hence, our goal is to avoid random oracles and still have an efficient solution.

In [14], Boudot presents an efficient technique to prove relations, as outlined above, given a primitive to prove that a committed integer is a square. Furthermore, in [1], Abe, Cramer and Fehr propose efficient and non-interactive techniques for proving multiplicative relations on secret-shared values, using distributed-verifier proofs. Unfortunately, the protocols and definition from [1] are not directly useful in the scenarios outlined above, for several reasons: First, the relations that can be proved only hold modulo some (public) prime number, and not necessarily over the integers. Second, for the case of honest majority, the protocols in [1] are only "non-interactive with complaints", that is, if a server is unhappy with the data he received privately from the dealer, he will complain, and the dealer must intervene in a second round to resolve these conflicts. It is clear that we have to avoid this in our scenario. Third, the definition of distributed verifier proofs used in [1] works with only one prover. In our scenario, we will have many provers, some of which may be corrupted. In contrast to the single-prover case, a corrupt prover may now try to exploit the information sent by honest provers in order to cheat.

In this chapter, we propose two protocols that allow a client to non-interactively VSS integers among the servers, and prove in zero-knowledge, by a distributed verifier proof, that shared integers a, b, c satisfy $ab = c$. Using known reductions [14], this implies non-interactive proofs that a shared integer is in a given interval, or that shared numbers a, b satisfy $a \geq b$. Both protocols require one broadcast from the prover and one round of messages between the verifiers (servers), which is a minimal amount of interaction for a distributed verifier proof. Details on the communication complexity of the protocols follow below. We prove our protocols secure in the Universal Composability model (with static adversary), this automatically gives us a definition handling the multiple prover case.

For the first solution, we take the protocol of [1] as the point of departure, introducing new techniques to solve the problems mentioned above. We obtain our solution by replacing in the protocol from [1] Shamir secret-sharing by Linear Integer Secret Sharing (LISS) – which exists for any access structure (see Chapter 3). LISS schemes are basically secret sharing schemes where the secret is reconstructed by taking a integral linear combination of the shares. Also, we replace Pedersen commitments [77] by the integer commitments from [51].

While this is quite straightforward, it is not so trivial to solve the problem of handling complaints without interaction. We first observe that the reason why the dealer must resolve conflicts in the protocol by Abe et al. is that only point-to-point channels between dealer and each server are assumed, and hence servers are not a priori committed to what they received. On the other hand, a typical implementation would realize the channels using public-key

encryption, so we propose to include this encryption explicitly in the protocol. One might now hope that a server can prove it received bad data by "opening" the ciphertexts it received. However, while the sender of a ciphertext can always "open" it convincingly (simply by revealing the coins used to create it), we need that the *receiver* can do so. Since ciphertexts can be adversarially generated, and unopened ciphertexts must remain secure, it is not immediately clear how this can be done in a non-interactive and efficient way. We propose an efficient solution to the problem based on Identity-Based Encryption (IBE). To our knowledge, this is a new application of IBE, and we believe the idea is of independent interest, as the possibility of "complaining convincingly" is often useful in protocol design.

For the case of honest majority, the VSS we obtain requires the dealer to send a total of $O(n \log n(\kappa + l + k + n))$ bits, where κ is the security parameter for the public-key and commitment schemes used, n is the number of parties, l is the bit length of the numbers we share and k is an "information theoretic" security parameter, controlling the statistical leakage of information.

The protocol can handle any Q2 adversary structure (honest majority in the threshold case), which is optimal in terms of the number of corruptions that can be handled at all. However, for realistic values of the parameters, the efficiency is not what we might hope for. This is because the numbers we will be computing on will be numbers specifying bids, prices, productions costs, etc., that is, numbers that are typically much smaller than those used for public-key cryptography. Realistic parameter values might be $n = 7$, $l = 32$, $k = 60$ and $\kappa = 1024$. In such a case, each 32 bit number we share is expanded to about 25.000 bits, which hardly seems desirable.

We therefore propose another solution, where we make the stronger assumption that the adversary structure is Q3 (less than $n/3$ corruptions in the threshold case). We build a solution using a generalization of the pseudorandom secret-sharing technique from [29] to the case of linear *integer* secret sharing. In the threshold case, the protocol requires the dealer to send, once and for all, $O(T(\kappa + nk))$ bits to the servers, where T is the number of maximal unqualified sets in the adversary structure. After this, any number of VSS's can be done by sending $O(l + k)$ bits to the servers for each value to be shared. Each multiplication proof requires 3 VSS invocations and in addition $O((l + k + n)n)$ bits should be sent.

The initial step is not always efficient as a function of n because T may be exponential in n, depending on the adversary structure. In the typical threshold case, T would be about $\binom{n}{n/3}$. However, for a small number of servers, T is moderate. On the other hand, for fixed n and for a large number of VSS invocations we come very close to sending only $l + k$ bits for every l-bit number we share - where of course sending l bits is necessary. It is therefore ideally suited for cases, where a large number of clients need to supply large amounts of data to a small number of servers. For the example parameter values above and assuming we share, say 200 numbers, the dealer needs to send about 230 bits per number to share.

Both our protocols use a common reference string, and assume that the verifiers have public/secret key pairs set up in advance. Note that if we do

not assume random oracles, we cannot get non-interactive protocols without some sort of set-up assumption. Of course, using set-up assumptions, our problem could also be solved using standard techniques for non-interactive zero-knowledge. However, with current state of the art, this approach can only prove the type of statements we are after using generic techniques. This would give non-interactive proofs of size $\Omega(l\kappa|C|)$ where $|C|$ is the size of a Boolean circuit C checking the relation in question. For realistic parameter values, this will be several orders of magnitude larger than our complexity. To our knowledge, our solutions are the first non-interactive protocols for integer relations that do not use random oracles, and have communication complexity independent of the circuit complexity of the relation.

5.2 Verifiable Secret Sharing (VSS) and Distributed Verifier Proofs

5.2.1 Model and Definition

We have a set of m dealers $\{D_1, \ldots, D_m\}$ and a set of n parties or verifiers $\mathcal{P} = \{P_1, \ldots, P_n\}$. We assume an active and static adversary who may corrupt any number of dealers and a set of parties in a given adversary structure Δ. All parties, dealers and the adversary are polynomially bounded. We assume (for simplicity) synchronous communication. We use the Universal Composability framework [19] and define ideal functionalities as described in Figure 5.1 and 5.2.

Consider the $\mathcal{F}_{\mathsf{vss}}$ functionality first that consists of two commands. The Input command takes the value s to be VSS among the parties as argument. Note, that the Input command can take the special symbol $\perp$ as argument, which models that the dealer D_j is corrupt. The Input command takes one round, which models the fact that the implementation takes one round to finish, after the prover has spoken. The Open command simply opens the VSS value s, if all honest parties agree or the dealer D_j chooses to do so.

The $\mathcal{F}_{ab=c}$ functionality consists of two commands, an Input and an Open command. The Input command verifies that the given input a, b, c fulfills the constraint $ab = c$. If this is not the case, the functionality rejects the input after one round, otherwise it accepts. Finally, the Open command can open the c value if all honest parties or the dealer chooses so.

Note 5.1. The $\mathcal{F}_{ab=c}$ functionality could just as well have commands to open both the a and the b value, but we tailored it to our needs, where we only need to show that a given secret shared value is the product of two others, we do *not* need the actual product for later use. Finally note, that there is nothing in the proofs which precludes this extension of the $\mathcal{F}_{ab=c}$ functionality.

For our protocols, we will need a set-up assumption, namely $D_1, \ldots, D_m$ and $P_1, \ldots, P_n$ get common input $k, pk, pk_1, \ldots, pk_n$, where k is the security parameter, pk_i is the public key of P_i, and pk is a common reference string. As private input, P_i has a secret key sk_i corresponding to pk_i. For simplicity, we assume here that the public and secret keys are generated and given to parties initially by an ideal functionality T. However, we stress that T can

Functionality $\mathcal{F}_{\mathsf{VSS}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$, dealers $D_1, \ldots, D_m$, and an adversary S.

1. Upon receiving (Input, sid, s) from a dealer D_j:
   (a) Send (Input, D_j) to all parties and the adversary.
   (b) Wait one round, then send (Reject, D_j) if $s = \perp$, otherwise send (Accept, D_j) to all parties and the adversary.
2. At any later time, upon receiving (Open, sid) from all honest parties or the dealer D_j:
   (a) Send (Value, s) to all parties and the adversary.

Figure 5.1: Functionality $\mathcal{F}_{\mathsf{VSS}}$.

Functionality $\mathcal{F}_{ab=c}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$, dealers $D_1, \ldots, D_m$, and an adversary S.

1. Upon receiving $(\mathsf{Input}, sid, a, b, c)$ from a dealer D_j:
   (a) Send (Input, D_j) to all parties and the adversary.
   (b) Wait one round, then send (Reject, D_j) if $ab \neq c$, otherwise send (Accept, D_j) to all parties and the adversary.
2. At any later time, upon receiving (Open, sid) from all honest parties or the dealer D_j:
   (a) Send (Value, c) to all parties and the adversary.

Figure 5.2: Functionality $\mathcal{F}_{ab=c}$.

be implemented by a once-and-for-all preprocessing among the parties (it is well known that any UC functionality can be securely implemented if we have honest majority, or in general $Q2$). In Section 5.2.4, it is even sufficient that parties generate their own key pairs and broadcast the public keys. We also assume a functionality $\mathcal{F}_{\mathsf{BC}}$, allowing any dealer to broadcast information to the verifiers[1]. Communication between verifiers uses standard authenticated but non-secret channels. Note that the UC framework incorporates, in addition to the adversary $\mathcal{A}$ attacking the protocol, an environment $\mathcal{Z}$ that chooses inputs for and receives outputs from honest parties. We will only consider

[1]Note, that even if we implement the broadcast via a subprotocol, this can be done such that we maintain the non-interactive nature of our proofs, namely the dealer sends a single (signed) message to all parties, who then internally agree on what he said.

environments that give integers (and not $\perp$) as input to honest parties. This models the assumption that honest parties would only attempt to VSS valid integers.

Another issue, which is abstracted away from the functionality, is that the integer to be committed, should be numerical bounded. We go into deeper detail of the problem in Chapter 8 and 9. Shortly described, when secret sharing an integer $a \in \mathbb{Z}$, then a needs to be numerical bounded by some limit (e.g., $|a| < 2^l$) in order to provide privacy. The functionality does not tell the bit-size of the secret, which would compromise the privacy, it only tells that a secret has been shared. Hence, we must assume that the integer a, which is secret shared, is bounded, otherwise the privacy of LISS does not hold and the simulation might fail. Hence, we only quantify over environments that use the functionality on integers that are bounded (the bound is not written explicitly in the functionality).

5.2.2 An Integer Commitment Scheme

A *commitment scheme* for domain $\mathcal{S}$ is given by a family of functions $\mathsf{com}_{pk} : \mathcal{S} \times \mathcal{R}_{pk} \to \mathcal{C}_{pk}$, indexed by a *public key* pk. One commits by publishing $C = \mathsf{com}_{pk}(s, r)$, where $s \in \mathcal{S}$ is the committed value and $r \in \mathcal{R}_{pk}$ is a random value. A *homomorphic* commitment scheme is a scheme where we assume that $\mathcal{S}$ is an additive group and that for any two commitments C and C' and any number λ, anyone can compute commitments S and P such that being able to open C and C' to s and s', respectively, allows to open S to the sum $s + s'$ and P to the product λs.

We use a modified version of the Pedersen commitment scheme [77], based on a multiplicative group G of order unknown to the parties. This commitment scheme first appeared in [50] and later in [51]. We will need primes p, q where $p = 2p' + 1$ and $q = 2q' + 1$ and p', q' are also prime. The computations are done in $\mathbb{Z}_N^*$, where $N = pq$, and the public key is $pk = (N, g, h)$ where g, h are chosen at random in Q_N, the set of squares modulo N. Then we use $\mathsf{com}_{pk} : (s, r) \mapsto g^s h^r \bmod N$. The scheme is *homomorphic*, since given commitments $C = \mathsf{com}_{pk}(s, r)$ and $C' = \mathsf{com}_{pk}(s', r')$ then $CC' = \mathsf{com}_{pk}(s + s', r + r')$ and $C^\lambda = \mathsf{com}_{pk}(\lambda s, \lambda r)$. Note that if we choose r uniformly random from $[0..N2^k]$, then $r \bmod \mathrm{ord}(h)$ is statistically close to being uniformly random in $[0..\mathrm{ord}(h)-1]$.

An important advantage of this scheme is that it allows commitment to *integers*. This follows since the commitment is done in a group G of unknown order. More specifically, the following proposition holds for the above commitment scheme.

Proposition 5.1 ([50])**.** $\mathsf{com}_{pk}(s, r)$ *is a statistically hiding and computationally binding commitment scheme, i.e.:*

- *If factoring is infeasible, then given $pk = (N, g, h)$ it is infeasible to compute $s, s', r, r' \in \mathbb{Z}$ where $s \neq s'$ such that $\mathsf{com}_{pk}(s, r) = \mathsf{com}_{pk}(s', r')$.*
- *For any two values $s, s' \in \mathbb{Z}$, the distributions of $(pk, \mathsf{com}_{pk}(s, r))$ and $(pk, \mathsf{com}_{pk}(s', r'))$ are statistically indistinguishable.*

5.2.3 Public-key Encryption with Verifiable Opening

We introduce here a tool that we will need later. Suppose a party P has a public/secret key pair (pk, sk), and receives ciphertext from various senders, some of whom may be corrupt. We want that the cryptosystem is chosen ciphertext (CCA) secure and has the additional property that for any received ciphertext c, P can reveal the decryption result $x = D_{sk}(c)$ and prove non-interactively and efficiently that x is correct. We want, of course, that "unopened" ciphertexts remain secure, which excludes the trivial solution of revealing the secret key.

Note that if c is a valid ciphertext, the random coins used to generate c can serve as proof of what the plaintext was. However, even if the receiver could compute these coins efficiently, there is still a problem if the sender is corrupt. Then c may be invalid, and "the coins used to generate c" is not even a well-defined notion.

A formal definition of the notion we are after is given below.

A public-key encryption scheme with non-interactive opening is a tuple $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ of algorithms such that:

- The key generation algorithm Gen takes as input a security parameter 1^k and outputs a public key pk and a secret key sk. We write $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$. The public key pk specifies the message space $\mathcal{M}_{pk} \leftarrow \mathsf{MSpc}(pk)$ by a mapping MSpc.

- The encryption algorithm Enc takes as input a public key pk and a message $m \in \mathcal{M}_{pk}$ and outputs a ciphertext C. We write $C \leftarrow_{\mathrm{R}} \mathsf{Enc}_{pk}(m)$.

- The deterministic decryption algorithm Dec takes as input a ciphertext C and a secret key sk. It returns a message $m \in \mathcal{M}_{pk}$ or the distinguished symbol $\perp \notin \mathcal{M}_{pk}$. We write $m \leftarrow \mathsf{Dec}_{sk}(C)$.

- The proving algorithm Prove takes as input a ciphertext C and a secret key sk. It returns a proof π. We write $\pi \leftarrow_{\mathrm{R}} \mathsf{Prove}_{sk}(C)$.

- The deterministic verification algorithm Ver takes as input a tuple (C, m, π, pk), consisting of a ciphertext C, a plaintext m, a proof π, and a public key pk. It returns a result $res \in \{\mathtt{accept}, \mathtt{reject}\}$. We write $res \leftarrow \mathsf{Ver}_{pk}(C, m, \pi)$.

We require that with probability overwhelming in the security parameter k, an honestly generated keypair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$ satisfies the following:

- **Correctness.** For all messages $m \in \mathcal{M}_{pk}$, we have

$$\mathsf{Pr}\left[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m\right] = 1.$$

- **Completeness.** For all ciphertexts C and all possible $\pi \leftarrow \mathsf{Prove}_{sk}(C)$, we have that for $m \leftarrow \mathsf{Dec}_{sk}(C)$, algorithm $\mathsf{Ver}_{pk}(C, m, \pi)$ accepts.[2]

[2] Note that m may be $\perp$.

Definition 5.1 (PKENO security)**.** *A scheme* PKENO *is* PKENO*-secure if it is* IND-CCPA *secure and satisfies computational proof soundness. We define those two below:*

IND-CCPA SECURITY. *For an adversary* A*, consider the following game:*

1. $\mathsf{Gen}(1^k)$ *outputs* (pk, sk)*. Adversary* A *is given* 1^k *and* pk*.*
2. *The adversary may make polynomially many queries to a decryption oracle* $\mathsf{Dec}_{sk}(\cdot)$ *and a proof oracle* $\mathsf{Prove}_{sk}(\cdot)$*.*
3. *At some point,* A *outputs two equal-length messages* m_0, m_1*. A bit* b *is randomly chosen and the adversary is given the challenge ciphertext* $C^* \leftarrow \mathsf{Enc}_{pk}(m_b)$*.*
4. A *may continue to query its decryption and its proof oracle, except that it may not query either with* C^**.*
5. *Finally,* A *outputs a guess* b'*.*

Denote A*'s advantage in guessing* b' *by*

$$\mathcal{A}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k) := \left|\mathsf{Pr}\left[b = b'\right] - 1/2\right|.$$

Scheme PKENO *is called* indistinguishable against chosen-ciphertext and prove attacks (IND-CCPA secure) *if for every adversary* A*,* $\mathcal{A}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(\cdot)$ *is negligible.*

PROOF SOUNDNESS. *For an adversary* A*, consider the following game:*

1. $\mathsf{Gen}(1^k)$ *outputs* (pk, sk)*. Adversary* A *is given* 1^k *and* (pk, sk)*.*
2. *The adversary chooses a message* $m \in \{0,1\}^*$ *and gives it to an encryption oracle that returns* $C \leftarrow_{\mathrm{R}} Enc_{pk}(m)$*.*
3. *The adversary returns* (m', π')*.*

Denote A*'s probability to forge a proof by*

$$\mathcal{A}^{\text{snd}}_{\mathsf{PKENO},\mathsf{A}}(k) := \mathsf{Pr}\left[\mathtt{accept} \leftarrow \mathsf{Ver}_{pk}(C, m', \pi') \wedge m' \neq m\right].$$

Scheme PKENO *is said to satisfy* computational proof soundness *if for every adversary* A*,* $\mathcal{A}^{\text{snd}}_{\mathsf{PKENO},\mathsf{A}}(\cdot)$ *is negligible.*

To construct a PKENO scheme we need an identity-based cryptosystem (IBE) [13]. Note that, under reasonable assumptions, efficient IBE's exist that do not use random oracles [87]. For the IBE we use, we need that given identity t and pk, one can easily verify if a secret key sk_t is the secret key for identity t. This can indeed be done for all known efficient IBE's, we call this IBE with verifiable secret keys (IBE-VSK).

An IBE-VSK scheme is a tuple $\mathsf{IBE} = (\mathsf{Gen}_{\mathsf{ibe}}, \mathsf{KeyGen}_{\mathsf{ibe}}, \mathsf{Enc}_{\mathsf{ibe}}, \mathsf{Dec}_{\mathsf{ibe}}, \mathsf{Ver}_{\mathsf{ibe}})$, of algorithms along with a family $\mathcal{M} = (\mathcal{M}_k)_k$ of message spaces such that:

- The key generation algorithm $\mathsf{Gen}_{\mathsf{ibe}}$ takes as input a security parameter 1^k and outputs a public key pk and a secret key sk. We write $(pk, sk) \leftarrow_R \mathsf{Gen}_{\mathsf{ibe}}(1^k)$.

- The encryption algorithm $\mathsf{Enc}_{\mathsf{ibe}}$ takes as input a public key pk, in identity $id \in \{0,1\}^*$ and a message $m \in \mathcal{M}_k$ and outputs a ciphertext c. We write $c \leftarrow_R \mathsf{Enc}_{\mathsf{ibe},pk}(id, m)$.

- The deterministic decryption algorithm $\mathsf{Dec}_{\mathsf{ibe}}$ takes as input a ciphertext c, an identity $id \in \{0,1\}^*$ and a user secret key $usk[id]$. It returns a message $m \in \mathcal{M}_k$ or the distinguished symbol $\perp \notin \mathcal{M}_k$. We write $m \leftarrow \mathsf{Dec}_{\mathsf{ibe},usk[id]}(c)$.

- The deterministic user secret key algorithm $\mathsf{KeyGen}_{\mathsf{ibe}}$ takes as input an identity $id \in \{0,1\}^*$ and a secret key sk. It returns a user secret key $usk[id]$. We write $usk[id] \leftarrow \mathsf{KeyGen}_{\mathsf{ibe},sk}(id)$.

- The deterministic user secret key verification algorithm $\mathsf{Ver}_{\mathsf{ibe}}$ takes as input an user secret key $usk[id]$, an identity $id \in \{0,1\}^*$, and a public key. It returns either `accept` or `reject`. We write $\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Ver}_{\mathsf{ibe},pk}(usk[id], id)$.

We require that for every honestly generated keypair $(pk, sk) \leftarrow_R \mathsf{Gen}_{\mathsf{ibe}}(1^k)$, for all identities $id \in \{0,1\}^*$ and messages $m \in \mathcal{M}_k$ we have

$$\mathsf{Dec}_{\mathsf{ibe},\mathsf{KeyGen}(sk,id)}(\mathsf{Enc}_{\mathsf{ibe},pk}(id, m)) = m,$$

with probability one.

Here we also require a non-standard soundness property that it is efficiently verifiable if a given user secret key $usk[id]$ was properly generated for identity id. We require for all honestly generated keypair $(pk, sk) \leftarrow_R \mathsf{Gen}_{\mathsf{ibe}}(1^k)$ satisfies the following: For all identities $id \in \{0,1\}^*$ and strings $s \in \{0,1\}^*$ we have $\mathsf{Ver}_{\mathsf{ibe},pk}(id, s) = \texttt{accept}$ if and only if $s = usk[id]$, where $usk[id] \leftarrow \mathsf{KeyGen}_{\mathsf{ibe},sk}(id)$.

We only require a relatively weak security property, namely indistinguishability against selective-ID chosen-plaintext attacks (IND-sID-CPA) [12]. Formally, for an adversary A, consider the following game:

1. Adversary A is given 1^k and outputs a target identity id^*

2. $\mathsf{Gen}_{\mathsf{ibe}}(1^k)$ outputs (pk, sk). Adversary A is given 1^k and pk.

3. The adversary may make polynomially many queries to a user secret-key oracle $\mathsf{KeyGen}_{\mathsf{ibe},sk}(\cdot)$, except that it may not query for id^*

4. At some point, A outputs two equal-length messages m_0, m_1. A bit b is randomly chosen and the adversary is given the challenge ciphertext $c^* \leftarrow_{\mathrm{R}} \mathsf{Enc}_{\mathsf{ibe},pk}(id^*, m_b)$.

5. A may continue to query its user secret-key oracle, except that it may not query for id^*.

6. Finally, A outputs a guess b'.

Denote A's advantage in guessing b' by

$$\mathcal{A}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{A}}(k) := \left|\mathsf{Pr}\left[b = b'\right] - 1/2\right|.$$

Scheme IBE is called IND-sID-CPA secure if $\mathcal{A}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{A}}(\cdot)$ is negligible for every PPT adversary A. We remark that there exist efficient IND-sID-CPA secure IBE schemes without random oracle [12].

We use an adaptation of the IBE-to-PKE transformation by Canetti, Halevi and Katz [22]. Let $\mathsf{OTS} = (\mathsf{SGen}, \mathsf{SSign}, \mathsf{SVer})$ be a one-time signature scheme that we require to be strongly unforgeable against one-time attacks (Syntax and security properties of OTS are recalled in Appendix A). We construct a PKENO scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ as follows.

- $\mathsf{Gen}(1^k)$ runs $(pk, sk) \leftarrow_R \mathsf{Gen}_{\mathsf{ibe}}(1^k)$ and returns (pk, sk).
- $\mathsf{Enc}_{pk}(m)$ first generates a key-pair for the one-time signature scheme by running $(vk, sigk) \leftarrow_R \mathsf{SGen}(1^k)$. Next, it IBE encrypts m with "identity" vk to obtain $c \leftarrow_R \mathsf{Enc}_{\mathsf{ibe},pk}(vk, m)$. Finally, it signs the IBE ciphertext $\sigma \leftarrow \mathsf{SSign}_{sigk}(c)$, and returns the PKENO ciphertext $C = (vk, c, \sigma)$.
- $\mathsf{Dec}_{sk}(C)$ parses C as the tuple (vk, c, σ). Next, it verifies if σ is a correct signature on c by running $\mathsf{SVer}_{vk}(c, \sigma)$. If not, it returns $\perp$. Otherwise, it computes $usk[id] \leftarrow \mathsf{KeyGen}_{sk}(vk)$ and IBE decrypts c by running $m \leftarrow \mathsf{Dec}_{\mathsf{ibe},usk[id]}(c)$. Finally, it returns $m \in \mathcal{M}_k \cup \{\perp\}$.
- $\mathsf{Prove}_{sk}(C)$ parses C as the tuple (vk, c, σ). Next, it verifies if σ is a correct signature on c by running $\mathsf{SVer}_{vk}(c, \sigma)$. If not, it returns $\perp$. Otherwise, it computes $usk[id] \leftarrow \mathsf{KeyGen}_{sk}(vk)$ and returns $\pi \leftarrow usk[id]$ as the proof.
- $\mathsf{Ver}_{pk}(C, m, \pi)$ parses C as the tuple (vk, c, σ). Next, it verifies if σ is a corrects signature on c with respect to verification key vk by running $\mathsf{SVer}_{vk}(c, \sigma)$. If not, it returns `reject`. Otherwise, it checks if π is a properly generated user secret-key for "identity" vk by running $\mathsf{Ver}_{\mathsf{ibe},pk}(vk, \pi)$. If not, it returns `reject`. Otherwise, it IBE decrypts c by running $\hat{m} \leftarrow \mathsf{Dec}_{\mathsf{ibe},\pi=usk[id]}(c)$, where $\hat{m} \in \mathcal{M}_k \cup \{\perp\}$. If $\hat{m} \neq m$, it returns `reject`. Otherwise, it returns `accept`.

Theorem 5.1. *Assume* IBE *is* IND-sID-CPA *secure and* OTS *is* SUF-OT *secure. Then* PKENO *constructed above is* PKENO *secure.*

Proof. Let A be an IND-CCPA adversary against PKENO. We show how to construct an IND-sID-CPA adversary B against IBE and a SUF-OT adversary F against OTS such that

$$\text{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k) \leq \text{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{B}}(k) + \text{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(k).$$

We start by describing B that interacts with the IND-sID-CPA security game against IBE as follows.

1. On input 1^k, B picks $(sigk^*, vk^*) \leftarrow_R \mathsf{SGen}(1^k)$ and returns vk^* as the challenge identity.

2. Adversary B inputs pk and runs A on pk.

3. As already pointed out it is sufficient for B to answer A's proof queries. Let $C = (vk, c, \sigma)$ be such a proof query made by A. First, B checks if the signature σ is correct and rejects if not. Next, if $vk \neq vk^*$ then B terminates and reports failure. Next, B queries its $\mathsf{KeyGen}_{sk}(\cdot)$ oracle for the user secret key $usk[id]$ of "identity vk" and returns the proof $\pi = usk[id]$ to A.

4. B inputs the two messages (m_0, m_1) from A and submits them to its own experiment obtaining a challenge ciphertext c^* for identity vk^*. It returns $C = (vk^*, c^*, \sigma^*)$ to A, where $\sigma^* \leftarrow_{sigk} (c^*)$.

5. B continues answering A's proof queries, as above.

6. Finally A returns a bit b' that B outputs to its own IBE experiment.

We denote Fail the event the B reports failure in the above simulation. Let WinA and WinB be the events that A and B win their security experiment in the above simulation. Note, that B perfectly simulates A's view in the IND-CCPA experiment unless Fail happens and we have $\Pr[\mathsf{WinB} \wedge \neg\mathsf{Fail}] = \Pr[\mathsf{WinA} \wedge \neg\mathsf{Fail}]$. Hence,

$$|\Pr[\mathsf{WinA}] - \Pr[\mathsf{WinB}] \leq \Pr[\mathsf{Fail}],$$

and we get

$$\begin{aligned} \mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}} &= |\Pr[\mathsf{WinA}] - 1/2| \\ &\leq |\Pr[\mathsf{WinB}] - 1/2| + \Pr[\mathsf{Fail}] \\ &= \mathrm{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{B}} + \Pr[\mathsf{Fail}]. \end{aligned}$$

We now give an adversary F such that

$$\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}} \geq \Pr[\mathsf{Fail}].$$

Adversary F carries out the above simulation with the following tow changes. First, it generates $(pk, sk) \leftarrow_R \mathsf{Gen}_{\mathsf{ibe}}(1^k)$ itself, hence being able to simulate the surrounding IBE IND-sID-CPA security experiment. Second, instead of generating $(vk^*, sigk^*)$ itself, it only inputs vk^* from the SUF-OT experiment against OTS. Adversary F only has to make one call to the signing oracle to obtain the signature σ^* on c^* with respect to vk^*. Consider the case that Fail happens and let $(vk = vk^*, C, \sigma)$ be the corresponding proof query. Since $(C, \sigma) \neq (C^*, \sigma^*)$, σ is a valid strong forgery on C with respect to vk^*. This implies,

$$\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}} \geq \Pr[\mathsf{Fail}],$$

and completes the proof. □

5.2.4 VSS using Integer Commitments

In this section we construct a non-interactive *verifiable secret sharing* [26] (VSS) scheme based on LISS. In VSS the object is to resist malicious parties, such as

(i) a dealer sending incorrect shares to some or all of the participants, and

(ii) participants submitting incorrect shares during the reconstruction protocol,

as defined in [26]. In the following when we say a value is *shared* among the parties, then it means that the value is verifiable secret shared among the parties.

We use the model described in the previous sections. Specifically, the common reference string will be a public key $pk = (g, h, n)$ for the integer commitment scheme described above. Moreover, each party P_j has a key pair (pk_j, sk_j) for a PKENO scheme as described above.

Protocol $\mathsf{Share}_{pk}(s)$

On input $s \in [-2^l..2^l]$, the dealer D makes a commitment $C = \mathsf{com}_{pk}(s, r)$ to s, and then executes the following protocol to prove that he knows how to open C to value s, and to secret share s:

Protocol $\mathsf{Proof}_{pk}(C)$

1. Given an ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$, the dealer D chooses a random *distribution vector* $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]^e$ with $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = s$, and commits to $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_e)^T$ by commitments $R_1, \ldots, R_e$ to $\rho_1, \ldots, \rho_e$, respectively, where $R_1 = C$ and all commitments use (g, h) as public parameter. The commitments $R_2, .., R_e$ to the additional randomness are included in the proof π. D computes the shares of s: $\boldsymbol{s} = (s_1, \ldots, s_d)^T = M\boldsymbol{\rho}$, and for each $i = 1, \ldots, d$ he computes the opening information o_i for the corresponding commitment

$$C_i = \prod_{j=1}^{e} R_j^{m_{ij}}$$

using the homomorphic property, where m_{ij} is defined by $M = [m_{ij}]$. Finally, he includes $c_i = E_{pk_{\psi(i)}}(o_i)$ in his proof π, where all these ciphertexts are assigned a tag consisting of C concatenated with the name of D (see Section 5.2.3). Finally, D broadcasts C, π.

2. For each i, $P_{\psi(i)}$ decrypts c_i. If he finds that the resulting opening information o_i is incorrect w.r.t. C_i, then he sends o_i to all other parties, along with a proof that o_i is indeed the result of decrypting c_i, this counts as an accusation against D. Otherwise he sends Accept.

3. For any accusation from $P_{\psi(i)}$, each party verifies that any o_i received is indeed the value that c_i decrypts to. If this is not the case this o_i is discarded.
4. Each party looks at all (non-discarded) o_i-values he knows. If any such o_i is inconsistent with C_i, then he rejects. Otherwise he accepts.

We need the following protocol to open a verifiable secret shared value.

Protocol $\mathsf{Open}(C)$

All parties know commitments $R_1 = C, R_2, \ldots, R_e$, and for $i = 1, \ldots, d$ party $P_{\psi(i)}$ has opening $o_i = (s_i, r_i)$ to commitment,

$$\mathsf{com}_{pk}(s_i, r_i) = C_i = \prod_{j=1}^{e} R_j^{m_{ij}},$$

where m_{ij} is defined by $M = [m_{ij}]$.

1. For $i = 1, \ldots, d$ party $P_{\psi(i)}$ broadcasts o_i.
2. Upon receiving $o_i = (s_i, r_i)$, check whether $\mathsf{com}_{pk}(s_i, r_i) = C_i$, if it holds add o_i to an initially empty collection O.
3. When O consists of share components of a qualified set of parties $A \in \Gamma$, then take the corresponding reconstruction vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T$ and let $s = \sum_{i=1}^{d} \lambda_i s_i$.

Note 5.2. The reconstruction vector $\boldsymbol{\lambda}$ used in step 3 in the above Open protocol only has non-zero entries for the entries corresponding to the accepted openings in O.

Theorem 5.2. *Given a secure PKENO and a binding commitment scheme, then the protocols* $\mathsf{Share}_{pk}(s)$ *and* $\mathsf{Open}(C)$ *securely implement* $\mathcal{F}_{\mathsf{vss}}$*, assuming any* $Q2$ *access structure* Γ*.*

Proof. To show that $\mathsf{Share}_{pk}(s)$ securely implements $\mathcal{F}_{\mathsf{vss}}$, we are given an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, and we need to construct a simulator S. The simulator interacts with $\mathcal{A}$ to simulate its view of attacking the protocol, and on the other hand interacts with $\mathcal{F}_{\mathsf{vss}}$ on behalf of corrupt parties. This game is called the *ideal process*. This is compared to the *real process*, where $\mathbb{Z}$ and $\mathcal{A}$ are interacting with a real instance of the protocol. In both processes, $\mathcal{Z}$ and $\mathcal{A}$ may communicate at any time. The goal is now to show that $\mathcal{Z}$ cannot distinguish the real from the ideal process. Our simulator works as follows:

1. The simulator generates the keys $pk = (N, g, h), \{(pk_j, sk_j)\}$ following T's algorithm, and sends all public keys to $\mathcal{A}$, along with secret keys for corrupted parties.

2. The simulator S now acts whenever required, as follows:

- If $\mathcal{A}$ sends C and a proof π to the broadcast functionality on behalf of corrupt dealer D_j, the simulator does the following: using its secret keys, it can decrypt ciphertext in π intended for honest parties and follow their algorithm to compute what they would send in the second round. This also lets it decide if the proof would be accepted. If not, the simulator sends $\perp$ to $\mathcal{F}_{\mathsf{vss}}$. If the proof is acceptable, observe first that since Γ is $Q2$, the set of honest parties, A, is qualified, and that every honest party can open his commitment to s_i. Let $\boldsymbol{\lambda}$ be a reconstruction vector for A, that is, $\langle \boldsymbol{s}, \boldsymbol{\lambda} \rangle = s$ and $\boldsymbol{\lambda}_{A^{\complement}} = \mathbf{0}$, i.e., if $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T$ then

$$\sum_{i=1}^{d} \lambda_i s_i = \sum_{i=1}^{d} \lambda_i \sum_{j=1}^{e} m_{ij} \rho_j = \rho_1 = s,$$

  where $\lambda_j = 0$ for $\psi(j) \notin A$. Hence, the above equation implies that $\sum_{i=1}^{d} \lambda_i m_{ij} = \delta_{1j}$, where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Therefore, by the homomorphic property, the simulator can open commitment $C' = \prod_{i=1}^{d} C_i^{\lambda_i}$ to $s' = \sum_{i=1}^{d} \lambda_i s_i$. Now, since

$$C' = \prod_{i=1}^{d} C_i^{\lambda_i} = \prod_{i=1}^{d} \left(\prod_{j=1}^{e} R_j^{m_{ij}} \right)^{\lambda_i} = \prod_{j=1}^{e} R_j^{\sum_i \lambda_i m_{ij}} = R_1 = C,$$

  we see that the simulator can extract from the proof a way to open commitment C to a value s. The simulator sends (Input, sid, s) to $\mathcal{F}_{\mathsf{vss}}$ on behalf of $\mathcal{A}$.

- On input (Input, D_j) from $\mathcal{F}_{\mathsf{vss}}$, where D_j is honest, the simulator simulates what D_j would send in the protocol, as follows: First, create a commitment to 0, i.e., $C = \mathsf{com}_{pk}(0, r)$. Since S generated $pk = (N, g, h)$ it follows that S knows $\mathrm{ord}(g)$ and $\mathrm{ord}(h)$, and a b such that $g = h^b \bmod N$. The simulator S can at a later point open C to any value, simply by noting that,

$$\mathsf{com}_{pk}(0, r) = \mathsf{com}_{pk}(s, r - bs).$$

  We therefore proceed, assuming implicitly that C "contains" s. Now, let A be the set of corrupted parties. Then there exists a sweeping vector $\boldsymbol{\kappa}$ such that $M_A \boldsymbol{\kappa} = \mathbf{0}$ and $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon} \rangle = 1$. Let $\boldsymbol{\rho}_0 = (r_1, \ldots, r_e)^T$ be a random distribution vector such that $\langle \boldsymbol{\rho}_0, \boldsymbol{\varepsilon} \rangle = 0$, i.e., a distribution vector to a random sharing of 0. Construct $R'_1, \ldots, R'_e$ as random commitments of $r_1, \ldots, r_e$, respectively, with the exception that $R'_1 = 1$ (or the commitment of $r_1 = 0$ using randomness 0). Then, by the homomorphic property of the commitment scheme, compute commitments

$$C'_i = \prod_{j=1}^{e} {R'_j}^{m_{ij}},$$

to shares s_i, which determines the secret 0. Now, given the commitment C for the secret s, we modify the commitments so they become consistent with s: Compute the public commitments $R_i = R'_i C^{\kappa_i}$ where $\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_e)^T$ is the sweeping vector for A. Note that $R_1 = R'_1 C^{\kappa_1} = 1C^1 = C$ as required, since $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon} \rangle = 1$ (i.e., $\kappa_1 = 1$). The commitments to the shares in s will be as follows:

$$C_i = \prod_{j=1}^{e} R_j^{m_{ij}} = \prod_{j=1}^{e} (R'_j C^{\kappa_j})^{m_{ij}} = \prod_{j=1}^{e} {R'_j}^{m_{ij}} C^{\kappa_j m_{ij}}.$$

For the parties in A we have that,

$$\prod_{j=1}^{e} C^{\kappa_j m_{ij}} = C^{\sum_j \kappa_j m_{ij}} = C^0 = 1,$$

since the inner product of $\boldsymbol{\kappa}$ and a row in M that is owned by a party in A is 0. So for a corrupt $P_{\psi(i)}$ we have $C'_i = C_i$, and we know how to open these commitments. The simulated proof therefore consists of the commitments $R_1, \ldots, R_e$, encryptions of correct opening information for C_i when $P_{\psi(i)}$ is corrupt, and encryptions of random values for honest parties.

- On input (Value, s) from $\mathcal{F}_{\mathsf{vss}}$, where the value comes from an honest party, the simulator calculates an opening to commitment $C = \mathsf{com}_{pk}(0, r)$ from the input phase, such that $C = \mathsf{com}_{pk}(s, r')$, which can be done since S generated the commitment scheme and knows the factorization. The simulator calculates the opening o_i for all,

$$C_i = \prod_{j=1}^{e} R_j^{m_{ij}} = \prod_{j=1}^{e} (R'_j C^{\kappa_j})^{m_{ij}} = \prod_{j=1}^{e} {R'_j}^{m_{ij}} C^{\kappa_j m_{ij}},$$

  owned by an honest party by using the opening information from C. Finally, S broadcasts the opening o_i on behalf on the honest party $P_{\psi(i)}$.
- On input (Value, s) from $\mathcal{F}_{\mathsf{vss}}$, where the value comes from a corrupt party, then just follow the opening protocol.
- If a corrupt party chooses to open her commitment, then send (Open, *sid*) to $\mathcal{F}_{\mathsf{vss}}$ on behalf of the corrupted party.

To see that this simulation works, note the following: First, the simulation of the initial set-up stage and of the case where a corrupt dealer gives a proof is perfect. In particular, when a corrupt dealer does a Share that would be accepted in the real protocol, the simulator can *always* extract the correct secret, and honest parties will therefore output accept also in the ideal process.

In the case where an honest dealer does a Share, this will in the ideal process simply mean that it sends integer s to $\mathcal{F}_{\mathsf{vss}}$. The functionality will send accept to everyone, so all honest parties output accept. This is also the case in the real protocol: correct opening information for each C_i is uniquely determined

from the ciphertext c_i, hence no honest party will accuse D and every other accusation will be rejected by the honest parties.

When the honest parties open a shared value by a corrupt party, the simulation is perfect. On the other hand, if the corrupted party opens the shared value s by himself, we need to argue, that he cannot open it another value $s' \neq s$ without being rejected. Let Pr denote the probability that $\mathcal{A}$ opens to $s' \neq s$. If Pr is non-negligible, then $\mathcal{A}$ can distinguish between the real and the ideal process, since in the real process she will open to s', whereas in the ideal process the functionality $\mathcal{F}_{\mathsf{vss}}$ will ensure that it is opened to s. Let Adv' denote such an adversary, then the goal is to show that we can break the binding property of the commitment scheme. However, we cannot argue that the system $Adv', Z, \mathcal{F}_{\mathsf{vss}}, S$ can break the binding property, since S has generated the commitment scheme and knows the factorization of N, the modulo used in the commitment. Instead we make a system Adv', Z, S', where we have removed the functionality $\mathcal{F}_{\mathsf{vss}}$. The idea is, that S' receives a public key (N, g, h) of the commitment scheme generated by some oracle. The environment $\mathcal{Z}$ is wired with S' instead of $\mathcal{F}_{\mathsf{vss}}$, i.e., S' will receives the values to be shared by honest parties. Hence, S' can generate a view that is the same as the real process, and therefore Adv' will still break the binding property with the same probability Pr. However, if Adv' breaks the binding, then S' will have two distinct openings to the commitment and can feed them to the oracle, hence breaking the binding property of the commitment scheme.

When an honest party opens a shared value, or more precisely, when the simulator receives a (Value, s) message from the $\mathcal{F}_{\mathsf{vss}}$ functionality, the simulator uses the knowledge from the generation of the commitment scheme to make an opening to s. This leads to the final thing to argue, which is the distribution of commitments and encryptions.

Hence the only possible difference between the ideal and real process is in the simulated commitment C and proof π that is shown to $\mathcal{A}$. By the statistical hiding property of the commitment scheme and privacy of the LISS scheme, it follows that the opening information sent to corrupt parties, as well as the commitments $R_1, \ldots, R_e$ have distribution statistically close to the one seen in the real protocol. So the only difference is the fact that the ciphertexts intended for honest parties are random in the simulation, and contain valid openings of commitments in the real protocol.

We cannot argue that the two sets of encryptions are indistinguishable based directly on the ideal process because S knows all secret keys. Instead, we construct a machine S' that acts as an adversary breaking the underlying PKENO scheme. S' will run the algorithms of Z, Adv and S, with the following modifications to S: S' receives public keys for the honest parties from an oracle. Whenever S needs to decrypt a ciphertext sent to an honest party with tag t (see Section 5.2.3), S' will ask the oracle for the secret key for that tag, and can then decrypt. When S wants to create ciphertext for honest parties in a simulated proof, S' will ask the oracle to encrypt either 1) random data or 2) genuine opening information for the relevant commitments. The latter is possible because S' also runs $\mathcal{Z}$ and therefore knows each secret that is shared, this allows it to create the commitment C as a genuine commitment containing the

right value, and from this it can compute how to open all the other commitments in that Share. In the case 1), we produce exactly what we get in the ideal process, in case 2) we produce something statistically close to what we get in the real process. Hence, if $\mathcal{Z}$ could distinguish the two processes, S' can use the output from $\mathcal{Z}$ to break the underlying IND-CCPA security in the PKENO scheme. □

5.2.5 Verifiable Commitment Multiplication Proof

We now show a (distributed verifier) proof that issued integers a, b, c satisfy that $c = ab$:

Protocol $\mathsf{MultProof}_{pk}(a, b, c)$

1. The prover makes commitments A, B, C to a, b, c and then executes $\mathsf{Proof}_{pk}(A)$, $\mathsf{Proof}_{pk}(B)$, and $\mathsf{Proof}_{pk}(C)$.
2. The prover executes $\mathsf{Proof}_{B,h}(C)$ using the *same* distribution vector $\boldsymbol{\rho}_a$ as in step 1 (but with new independent randomness for the commitments).
3. Every party verifies whether his shares obtained from $\mathsf{Proof}_{pk}(A)$ (from step 1.) and $\mathsf{Proof}_{B,h}(C)$ (from step 2.) coincide. If this does not hold, he accuses the dealer by opening the ciphertexts he received in $\mathsf{Proof}_{pk}(A)$ and $\mathsf{Proof}_{B,h}(C)$. Each party verifies any accusation made.
4. The proof is accepted if all subproofs were accepted, and no valid accusations were made.

Note that the four executions of the Proof protocol can be run in parallel. A similar protocol appeared in [1], but we have here added $\mathsf{Proof}_{pk}(B)$[3].

Theorem 5.3. *Assuming the integer commitment scheme is binding and given a secure PKENO scheme, then* $\mathsf{MultProof}_{pk}(a, b, c)$ *securely implements* $\mathcal{F}_{ab=c}$ *assuming any* $Q2$ *adversary structure* Γ.

Proof. Note that making commitments A, B, C and then executing the first 3 instances of Proof is equivalent to executing 3 instances of Share_{pk}. Therefore, to simulate this, we run the simulator from the previous theorem 3 times (in parallel). To simulate the execution of $\mathsf{Proof}_{B,h}(C)$, we run the same simulator again, with the following changes: when simulating the actions of an honest dealer, the simulator will not create its own commitment to play the role of the commitment to the secret, instead it will use C. Also, it will use the same distribution vector that was used in the simulation of $\mathsf{Proof}_{pk}(A)$.

To show that this simulation works, we only need to check that when we extract opening information from an acceptable proof given by a corrupt prover, we will get values a, b, c such that $ab = c$. Note, that if the proof is accepted, it

[3]This is necessary since the order of the group of the commitments is unknown and we can therefore not prove soundness the same way as in [1] (Lemma 1).

follows from the proof of Theorem 5.2 that we can extract from step 1. pairs $(a, r_a), (b, r_b)$ and (c, r_c) such that $A = \mathsf{com}_{g,h}(a, r_a)$, $B = \mathsf{com}_{g,h}(b, r_b)$ and $C = \mathsf{com}_{g,h}(c, r_c)$. Furthermore, steps 2. and 3. ensure that we can extract (c, r^*) such that $C = \mathsf{com}_{B,h}(c, r^*) = B^a h^{r^*}$ [4]. Combining this with the expression for $B = \mathsf{com}_{g,h}(b, r_b) = g^b h^{r_b}$ we get $C = B^s h^{r^*} = (g^b h^{r_b})^s h^{r^*} = g^{ab} h^{r_b a + r^*}$ In other words, we can now open C to both c and ab, which contradicts the binding property unless $c = ab$. □

5.3 Verifiable Multiplication Proof Based on Pseudo-Random Sharing

5.3.1 Replicated Integer Secret-Sharing and Share Conversion

In this section we first introduce the *Replicated Integer Secret-Sharing* (RISS), an integer version of Replicated Secret-Sharing [64], where we share an integer over a monotone access structure. Then we define share conversion, and show that shares generated by a RISS scheme can be locally converted to shares in the same secret generated by LISS schemes.

Scheme Replicated Integer Secret-Sharing (RISS)

Let Δ be an adversary structure. For each set $B \in \Delta^+$ choose a uniformly random r_B integer from the interval $[-2^{l+k}..2^{l+k}]$ and send privately r_B to each party $P_i \notin B$. Furthermore, publish $r = s + \sum_{B \in \Delta^+} r_B$, where s is the secret from the interval $[-2^l..2^l]$.

A RISS sharing is represented by $(r, \{r_B\}_{B \in \Delta^+})$, where $r = s + \sum_{B \in \Delta^+} r_B$ and s is the secret. The share of party P_i is given by $(r, \{r_B\}_{B \in \Delta^+, P_i \notin B})$. The parties can reconstruct the secret simply by publishing their shares, then the secret is given by $s = r - \sum_{B \in \Delta^+} r_B$.

For notational convenience we simply denote a RISS share by $\{r_B\}$, where we exclude the subscript and the public value. The adversary structure of the RISS scheme should always be clear from the context.

Lemma 5.1. *The RISS scheme is correct and (statistically) private for any $Q2$ adversary structure Δ.*

Proof. Since the adversary structure Δ is $Q2$, it implies that for all $B \in \Delta$ it holds that $B^{\complement}$ is a qualified subset of parties. Hence, any qualified set A must contain at least one party that is not in B, for any $B \in \Delta^+$. Hence A knows all the shares and correctness follows. On the other hand, any set $B \in \Delta^+$ has no information on r_B. Hence if s was shared, we can replace r_B by $r_B + s' - s$ for any other secret s', and this new value will be in range except with negligible probability in k. Privacy follows. □

[4] Note that the proof in step 2. uses B, which might have been adversarially generated, in place of g which comes from the common reference string. However, this is not a problem since the extraction will work for any set of values.

We proceed to define what is meant by a local convertible secret sharing scheme. Basically, it means that a secret sharing scheme $\mathcal{S}$ can locally be converted without communication into another secret sharing scheme $\mathcal{S}'$ sharing the same secret. The formal definition follows.

Definition 5.2. *Let $\mathcal{S}$ and $\mathcal{S}'$ be two secret-sharing schemes. We say that $\mathcal{S}$ is* locally convertible *to $\mathcal{S}'$ if there exist local conversion functions $g_1, \ldots, g_n$ such that the following holds. If $(s_1, \ldots, s_n)$ are valid shares of a secret s in $\mathcal{S}$, then $(g_1(s_1), \ldots, g_n(s_n))$ are valid shares of the same secret s in $\mathcal{S}'$. We denote by g the concatenation of all g_i, namely $g(s_1, \ldots, s_n) = (g_1(s_1), \ldots, g_n(s_n))$, and refer to g as a share conversion function.*

Note 5.3. By the locality feature of the conversion, that converted shares cannot reveal more information about s than the original shares.

Let $\mathbf{1} = (1, \ldots, 1)^T$ denote the all one vector. The size of $\mathbf{1}$ should always be clear from the context. In [66] Krachmer and Wigderson introduced the notion of *canonic span program*, we use the following modification for ISP.

Definition 5.3 (Canonic ISP)**.** *Let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ be an ISP for $\Gamma_{\mathcal{M}}$. We define a* canonic ISP *$\hat{\mathcal{M}} = (\hat{M}, \psi, \mathbf{1})$ as follows. $\hat{M}$ has the same size and row labeling as $\mathcal{M}$, but possibly a different number of columns. Let $\mathcal{T} = \mathcal{T}_{\mathcal{M}}$ be the collection of maximal forbidden sets of $\Gamma_{\mathcal{M}}$. For every $B \in \mathcal{T}$, let $\boldsymbol{\kappa}^B$ be the vector satisfying $M_B \boldsymbol{\kappa}^B = \mathbf{0}$ and $\langle \boldsymbol{\varepsilon}, \boldsymbol{\kappa}^B \rangle = 1$. For each maximal forbidden set $B \in \mathcal{T}$, the matrix $\hat{M}$ will include a corresponding column $\boldsymbol{c}^B = M \cdot \boldsymbol{\kappa}^B$ (so that altogether $\hat{M}$ has as many columns as sets in $\mathcal{T}_{\mathcal{M}}$).*

Observe the following properties of a canonic ISP. For every $B \in \Delta^+$ it holds that $\boldsymbol{c}^B_B = (0, \ldots, 0)^T$, i.e., the restriction of $\boldsymbol{c}^B$ to the entries jointly owned by the parties in B is the zero-vector. For each $A \in \Gamma$ it holds that $\hat{M}_A^T \boldsymbol{\lambda}_A = \mathbf{1}$, where $\boldsymbol{\lambda}_A$ is the reconstruction vector for M_A. This holds because, let $\boldsymbol{c}^B$ be an arbitrary column in $\hat{M}_A$, then

$$(\boldsymbol{c}^B)^T_A \boldsymbol{\lambda}_A = (M_A \boldsymbol{\kappa}^B)^T \boldsymbol{\lambda}_A = (\boldsymbol{\kappa}^B)^T (M_A^T \boldsymbol{\lambda}_A) = (\boldsymbol{\kappa}^B)^T \boldsymbol{\varepsilon} = 1,$$

where $\boldsymbol{\kappa}^B$ is corresponding sweeping vector to the column $\boldsymbol{c}^B$. Since this holds for any column in $\hat{M}$, this implies that we use an additive sharing to share in the canonic ISP, i.e., the target vector is $\mathbf{1}$.

Lemma 5.2. *Using the above notation, the scheme $\mathcal{S}_{\hat{\mathcal{M}}}$ is locally convertible to $\mathcal{S}_{\mathcal{M}}$ via the identity function $g(\boldsymbol{s}) = \boldsymbol{s}$.*

Remark 5.1. The proof uses ideas similar to what was used in [29].

Proof. (Lemma 5.2) Let $\hat{\boldsymbol{\rho}}$ be some integer additive sharing, that is $\langle \mathbf{1}, \hat{\boldsymbol{\rho}} \rangle = s$, which is induced by the dealer's randomness in $\mathcal{S}_{\hat{\mathcal{M}}}$. Let $\boldsymbol{\rho} = W \hat{\boldsymbol{\rho}}$ where W is a concatenation of all column vectors $\boldsymbol{\kappa}^B$ in the order used for constructing $\hat{M}$. By the construction of $\hat{M}$ we have $\hat{M} = MW$ and so $\hat{M}\hat{\boldsymbol{\rho}} = MW\hat{\boldsymbol{\rho}} = M\boldsymbol{\rho}$. Thus, $\boldsymbol{\rho}$ produces the same shares in $\mathcal{S}_{\mathcal{M}}$ as $\hat{\boldsymbol{\rho}}$ produces in $\mathcal{S}_{\hat{\mathcal{M}}}$. Finally, since every $\boldsymbol{\kappa}^B$ must satisfy $\langle \boldsymbol{\varepsilon}, \boldsymbol{\kappa}^B \rangle = 1$, we have $\langle \boldsymbol{\varepsilon}, \boldsymbol{\rho} \rangle = \boldsymbol{\varepsilon}^T W \hat{\boldsymbol{\rho}} = \langle \mathbf{1}, \hat{\boldsymbol{\rho}} \rangle$, and thus $\boldsymbol{\rho}$ is consistent with the same secret s. □

Lemma 5.2 states, that each legal sharing of s under a canonic ISP, is also a sharing of s under the corresponding ISP.

Lemma 5.3. *Let $\mathcal{R}_\Gamma$ be a RISS scheme realizing Γ over the integers, $\mathcal{M}' = (M', \psi', \boldsymbol{\varepsilon})$ an ISP for Γ' such that $\Gamma' \subseteq \Gamma$, and $\hat{M}'$ a canonic ISP of $\mathcal{M}'$. Then $\mathcal{R}_\Gamma$ is locally convertible to $\mathcal{S}_{\hat{M}'}$.*

Remark 5.2. The proof uses ideas similar to what was used in [29].

Proof. (Lemma 5.3) Suppose first that $\Gamma' = \Gamma$. Denote the collection maximal forbidden sets by Δ^+. The R_Γ-share viewed by party P_i is $\{r_B\}_{B\in\Delta^+ : P_i \notin B}$ and the public value $r = s - \sum_{B\in\Delta^+} r_B$. Fix some $B' \in \Delta^+$ and let c_i^B denote the i-th entry in $\boldsymbol{c}^B$. Define

$$g_i(s_i) = (r - r_{B'})c_i^{B'} - \sum_{B\in\Delta^+, B\neq B'} r_B c_i^B,$$

since each column $\boldsymbol{c}^B$ only has zeros in the entries corresponding to the parties in B, the g_i is a local conversion function. Furthermore, it the sharing of s as required.

In the general case, where $\Gamma' \subseteq \Gamma$, is only slightly more involved. For each $B \in \Delta'^+$ assign some set $B' \in \Delta^+$ containing it. For each $B' \in \Delta$, define $r_{B'}$ to be the sum of all r_B such that B is assigned to B', or 0 ir there is no B assigned to B'. Then the same local conversion function can be used. □

The following theorem follows immediately from the above two lemmas.

Theorem 5.4. *The RISS scheme $\mathcal{R}_\Gamma$, realizing Γ, is locally convertible to any LISS realizing an access structure $\Gamma' \subseteq \Gamma$.*

Clearly, for any prime p, a RISS sharing of integer s can be thought of as a replicated sharing over $\mathbb{Z}_p$ of $s \bmod p$, by reducing all shares modulo p. Furthermore, in [29] it was shown how to locally convert a replicated sharing over $\mathbb{Z}_p$ to any linear secret sharing (LSS) scheme over $\mathbb{Z}_p$ (such as Shamir's scheme). From these two observations, we immediately get.

Proposition 5.2. *The RISS scheme $\mathcal{R}_\Gamma$, realizing Γ, is locally convertible to any LSS over $\mathbb{Z}_p$ realizing an access structure $\Gamma' \subseteq \Gamma$, where the original secret s after conversion will be $s \bmod p$.*

Example 5.1. In the following we show how the above proposition is implemented for the threshold case, which we will use in the following sections before we generalize the result to arbitrary access structures.

Hence, the goal is to locally convert a RISS share for some threshold access structure over the integers to a Shamir share over a finite field $\mathbb{Z}_p$, for some prime $p > n$.

Consider a threshold-t access structure $T_{t,n}$. Assume the parties have values $\{r_B\}_{|B|=t}$ and r among them, such that all parties P_i with $P_i \notin B$ has r_B. It follows,

$$s = r - \sum_{B\subseteq\mathcal{P}, |B|=t} r_B,$$

where s is the secret value.

For every set $B \subseteq \mathcal{P}$ with $|B| = t$ let f_B be the uniquely defined degree-t polynomial over $\mathbb{Z}_p$ as follows,

1. $f_B(0) = 1$.
2. $f_B(i) = 0$ for all $i \in B$.

Each party P_i computes his share s_i as follows,

$$s_i = r - \sum_{B \subseteq \mathcal{P}, |B|=t, P_i \notin B} r_B \cdot f_B(i) \bmod p.$$

Consider the polynomial,

$$f = r - \sum_{B \subseteq \mathcal{P}, |B|=t} r_B \cdot f_B \bmod p,$$

which obviously has degree t, $f(0) = s \bmod p$, and $f(i) = s_i$ as required.

5.3.2 Application to VSS

We now show how the results from the previous subsection can be used to generate a series of verifiably shared secrets by broadcasting only two values per secret, at the initial cost of distributing a set of random seeds to the parties. We use the model defined earlier, where each party P_i has a public and a secret key. In this case, we assume that there is a public key pk_B defined for each $B \in \Delta^+$, and P_i's public key consists of all pk_B for those B in which P_i is *not* a member. The secret key consists of all secret keys corresponding to relevant pk_B's. As before, we assume these are keys for a PKENO scheme.

The following protocol does the initial distribution of seeds.

Protocol $\mathsf{Random}_{\{r_B\}}(\Delta^+)$

1. For each $B \in \Delta^+$ the dealer D choose an uniformly random r_B from the interval $[0..2^k[$.
2. For each $B \in \Delta^+$ D broadcasts r_B encrypted under pk_B. The dealer's name is used as tag for this ciphertext. Each party decrypts all the ciphertexts for which he has the secret key.

The protocol clearly ensures that parties have mutually consistent shares, i.e., all honest parties not in B agree on the value of r_B, for any $B \in \Delta^+$. In the following $\mathsf{Random}_{\{r_B^1,\ldots,r_B^t\}}(\Delta^+)$ will denote the protocol where the dealer distributes the seeds $r_B^1, \ldots, r_B^t$ to all parties not in B encrypted under pk_B.

Given a pseudorandom function (PRF) $\varphi_\cdot(\cdot)$ with k-bit keys,

$$\varphi : [0..2^k[\times \mathbb{Z} \to [-2^{l+k}..2^{l+k}],$$

then the following protocol is realizable.

Protocol $\mathsf{Share}_{\{r_B\}}(s)$

It is assumed that the dealer D has run $\mathsf{Random}_{\{r_B\}}(\Delta^+)$ on some adversary structure, Δ.

1. D broadcasts a value a, to serve as a "label" for this instance of the protocol. The only demand is that a can be used as input to φ, and that D never reuses an a-value. D computes, with his knowledge of $\{r_B\}$, $r = s + \sum_B \varphi_{r_B}(a)$ and broadcasts r.
2. Each party P_i checks that $r \in [-(|\Delta^+|+1)2^{l+k}..(|\Delta^+|+1)2^{l+k}]$, and rejects if this is not the case. Otherwise, he computes $\varphi_{r_B}(a)$, for every B where $P_i \notin B$.

This lemma follows easily by inspection of the protocol:

Lemma 5.4. *If D is honest, no honest party will reject in* $\mathsf{Share}_{\{r_B\}}(s)$*. No matter what the dealer does, if honest parties accept, the set of values $r, \{\varphi_{r_B}(a)| B \in \Delta^+\}$ form a RISS sharing of some value s'. If D is honest, $s' = s$, otherwise $s' \in [-(2|\Delta^+|+2)2^{l+k}..(2|\Delta^+|+2)2^{l+k}]$.*

It is also quite straightforward to see that if D is honest, and the PRF is secure, a polynomially bounded adversary does not learn anything about the secret involved. A proof of this is implicit in the proof of Theorem 5.5 below.

Clearly, if an adversary structure Δ is Q2, every complement of a set $B \in \Delta^+$ contains an honest party, so the VSS we constructed via protocol Share can always be opened by having parties open the encryptions of seeds they received in the initial phase. This is somewhat unsatisfactory, since this will open any secret that was shared using the relevant instance of $\mathsf{Random}_{\{r_B\}}(\Delta^+)$.

However if the adversary structure is $Q3$, in particular threshold-t with $t < n/3$, we can do better: namely, by choosing a prime p that is guaranteed to be larger than the secret s. Parties then locally convert their shares to a Shamir-sharing of s, reveal the shares and use standard error correction to find s.

In the general case of an $Q3$ adversary structure Δ, we can do the following. Note, that any linear secret sharing scheme over a field $\mathbb{F}$ can be expressed by a monotone span program [66], hence, given a monotone span program $(M \in \mathbb{F}^{d\times e}, \psi\colon \{1,\ldots,d\} \to \mathcal{P})$, then the shares are computed by choosing random vector $\boldsymbol{\rho} \in \mathbb{F}^e$ and simply computing $\boldsymbol{s} = M\boldsymbol{\rho}$. Let $\boldsymbol{s}$ be the shares, where at most $\boldsymbol{s}_A$ shares are corrupted for any $A \in \Delta$. Let $M \in \mathbb{F}^{d\times e}$ be a matrix (or the monotone span program), then the linear system $M\boldsymbol{\rho} = \boldsymbol{s}$ over the field $\mathbb{F}$ is *consistent*, if and only if the the vector $\boldsymbol{s}$ in $\mathbb{F}^d$ is in the span of the column vectors of M. Note, that if the linear system is consistent, then obviously the shares can be reconstructed uniquely to a value s. A linear system can by the *Gauss-Jordan* method (see, e.g., [6]) be efficiently checked whether it is consistent or not. The open protocol is as follows in the general $Q3$ case.

Protocol $\mathsf{Open}(\boldsymbol{s})$

The parties have shared $\boldsymbol{s}$ among them.

1. For $i = 1,\ldots,n$ party P_i publishes his share $\boldsymbol{s}_{\{P_i\}}$.

2. For each $A \in \Delta^+$ do the following.
   - Use the Gauss-Jordan method to check whether the linear system $M_{A^{\complement}}\boldsymbol{\rho}' = \boldsymbol{s}_{A^{\complement}}$ is consistent for any $\boldsymbol{\rho}'$.
   - If so, then reconstruct the secret from $\boldsymbol{s}_{A^{\complement}}$, and stop.
   - Proceed at step 2.

Remark 5.3. Since the shares of at most one subset in Δ^+ are corrupted and the structure is $Q3$, we know that for any $A, B \in \Delta^+$ the remaining set of parties $(A \cup B)^{\complement}$ is qualified. Hence, in step 2 of the Open protocol, we know that there is at least a qualified subset of parties in the remaining shares $\boldsymbol{s}_{A^{\complement}}$. This ensures that the shares $\boldsymbol{s}_{A^{\complement}}$ can only be consistent if no tampered corrupt shares are involved, which argues the correctness of the protocol.

The Open protocol is not very efficient, since the collection of sets in Δ^+ might be exponential. However, one can argue, that the adversary does not gain anything from cheating except slowing down the procedure, hence, she does not benefit from it in most cases.

However, in some cases we can do better than described in the above general case. If there exists an efficient strongly multiplicative monotone span program for the given adversary structure Δ we can use the method proposed by Cramer et al. in [31]. The parties choose a prime as described above, convert the shares to the strongly multiplicative monotone span program, and finally use the protocol proposed by Cramer et al. from [31].

5.3.3 Multiplication Proof

In this section we describe a protocol that non-interactively proves that a shared value is the product of two other shared values. For simplicity, we will only consider the case of a threshold adversary who corrupts $t < n/3$ of the parties, so the adversary structure Δ will in this section consist of all set of cardinality at most t. In Section 5.3.4 we describe a generalization to all Q3 adversary structures.

We will need a tool from [29], called Pseudorandom Zero Sharing (PRZS). This protocol assumes that for all $B \in \Delta^+$, parties not in B have been given t random seeds $r_B^1, \ldots, r_B^t$ and a prime $p > n$ is agreed in advance. Based on this, the protocol generates (by local computation only) a pseudorandom polynomial f over $\mathbb{Z}_p$ of degree at most $2t$ such that $f(0) = 0$ and each party P_i knows $f(i)$. The protocol is a simple generalization of the share conversion technique.

For each $B \in T_{t,n}^+$ consider the following set,

$$F_B = \{f \mid \deg(f) \leq 2t, f(0) = 0, j \notin B \Rightarrow f(j) = 0\}.$$

Let F denote the set of all polynomials of degree at most $2t$. Consider F as a vector space, then F_B is a subspace of F of dimension t. For each $B \in T_{t,n}^+$ choose once and for all $f_B^1, \ldots, f_B^t$ to be a basis for F_B and use it in the following protocol.

Protocol Pseudorandom Zero-Sharing (PRZS)

It is assumed that $\mathsf{Random}_{\{r_B^1,\ldots,r_B^t\}}(T_{t,n})$ has been run, where t is the threshold. Furthermore, they all have a common value a.

1. For $= 1, \ldots, n$ party P_i lets

$$s_i = \sum_{B \in T^+_{t,n}, P_i \in B} \sum_{j=1}^{t} \varphi_{r_B^j}(a) f_B^j(i)$$

Note 5.4. It follows from straight forward verification, that the shares from the above protocol are consistent with a polynomial f_0 of degree at most $2t$ and $f_0(0) = 0$.

We will choose a fixed prime p, such $p > 2(4|\Delta^+| + 2)^2 2^{2(l+k)}$.

Protocol $\mathsf{MultProof}_{\{r_B, r_B^1, \ldots, r_B^t\}}(a, b, c)$

1. The dealer D executes $\mathsf{Random}_{\{r_B, r_B^1, \ldots, r_B^t\}}(\Delta^+)$.
2. D executes $\mathsf{Share}_{\{r_B\}}(a)$, $\mathsf{Share}_{\{r_B\}}(b)$ and $\mathsf{Share}_{\{r_B\}}(c)$.
3. The parties use Proposition 5.2 to locally convert the RISS sharings we now have of a, b, c to Shamir sharings of $a \bmod p$,$b \bmod p$ and $c \bmod p$, consistent with polynomials f_a, f_b and f_c of degree at most t, t and $2t$ respectively. The parties use PRZS to generate shares in a polynomial f of degree at most $2t$ with $f(0) = 0$.
4. D uses his knowledge of all seeds to compute the polynomial $h = f + f_a f_b - f_c$ and broadcasts h.
5. Each party P_i verifies that $h(i) = f(i) + f_a(i) f_b(i) - f_c(i)$. If the verification fails then P_i broadcast "Accusation" and opens all encrypted values $r_B, r_B^1, .., r_B^t$ known by him.
6. The proof is rejected if one of the following situations happen: one of the Share protocols in Step 2 was rejected, the broadcasted polynomial h is not of degree at most $2t$, $h(0) \neq 0$, or broadcasted values by a party are consistent with the encrypted values but inconsistent with the broadcasted values by D.

Theorem 5.5. *When based on a secure PKENO scheme and PRF, then the protocol* $\mathsf{MultProof}_{\{r_B, r_B^1, \ldots, r_B^t\}}(a, b, c)$ *securely implements* $\mathcal{F}_{ab=c}$*, for any threshold-t adversary structure where* $t < n/3$.

Proof. We construct a simulator S that works as follows:

1. S generates the keys $pk, \{(pk_B, sk_B)\}$ following T's algorithm, and sends all public keys to $\mathcal{A}$, along with secret keys for corrupted parties.

2. S now acts whenever required, as follows:

   - When $\mathcal{A}$ does a proof on behalf of a corrupt dealer, S can simply decrypt everything sent by the adversary, and decide if the proof would be accepted in the real process. If so, it reconstructs values a, b and c and sends them to the ideal functionality. Otherwise, it sends $\perp$ to the ideal functionality and uses the honest parties' algorithm to compute the messages (complaints) they would send to corrupt parties, and sends these to $\mathcal{A}$.

- When an honest dealer does a proof, S will generate a simulated proof by simply following the prover's algorithm, using $a = b = c = 0$.
- When $\mathcal{A}$ does an opening on behalf of a corrupted party, then just follow the protocol.
- When receiving (Value, c) and let B denote the set of corrupted parties, then generate the shares of the honest parties using $v = \varphi_{r_B}(a) - c$ instead of $\varphi_{r_B}(a)$ in the conversion to the Shamir polynomial f_c. Otherwise follow the protocol and open the value.

To see that this simulation works as required, note first that the simulation of the set-up phase and proofs by corrupt dealers is perfect. This is because the simulator follows the honest parties algorithm to compute their reaction to the proof, so we just need to check that when the proof is accepted, the simulator can send a correct witness to the functionality. By Lemma 5.4, the values a, b, c that the simulator reconstructs from the proof will be in the interval $[-(2|\Delta^+| + 2)2^{l+k} \ldots (2|\Delta^+| + 2)2^{l+k}]$, so we know that $|ab|, |c|$ are less than $p/2$. Now, from Step 5, we know that h agrees with $f + f_a f_b - f_c$ in all points owned by honest parties, of which there are at least $2t + 1$. This implies that $h = f + f_a f_b - f_c$, and therefore that $ab = c \bmod p$. However, if $ab \neq c$, it would have to be the case that $|ab - c| \geq p$, while on the other hand we already know that $|ab - c| \leq |ab| + |c| < p$. So indeed $ab = c$.

It remains to show that the simulation of an honest dealer's proof shown to the adversary is indistinguishable from a real proof. For this, consider the real process *Real*, and assume the worst case where the adversary has corrupted a maximal set B of parties. This means that when an honest dealer does a proof, the key sk_B is the only secret key the adversary does not know. We then define a new "hybrid" process Hyb_1, where we replace the broadcasted encryptions of $r_B, r_B^1, \ldots, r_B^t$ (under pk_B) by encryptions of independent random values. By an argument similar to the proof of Theorem 5.2, *Real* is indistinguishable from Hyb_1 if the underlying PKENO scheme is secure. Note that in Hyb_1, we can replace evaluations of the PRF using seeds $r_B, r_B^1, \ldots, r_B^t$ by oracle access to the PRF with the same seeds, and all messages sent will remain unchanged. We define Hyb_2 by replacing the PRF oracles by oracles for truly random functions. By security of the PRF, Hyb_2 is indistinguishable from Hyb_1. Finally, we define Hyb_3 as follows: we first replace the dealer's inputs (a, b, c) to the $\mathsf{Share}_{\{r_B\}}(\cdot)$-protocols by random values in the legal interval, and second, we choose the polynomial h to broadcast as a uniformly random polynomial, subject to $h(0) = 0$, $deg(h) \leq 2t$, and that $h(i)$ agrees with the adversary's information for all corrupt parties P_i. Now, Hyb_3 is statistically indistinguishable from Hyb_2: consider, for instance, the execution of $\mathsf{Share}_{\{r_B\}}(a)$ in Hyb_2. If we subtract the randomness that the adversary already knows, we see that he can compute $R + a$, where R is a truly random value in $I_r = [-2^{l+k}..2^{l+k}]$. This is statistically indistinguishable from $R + r$ where r is a random value in $I_s = [-2^l..2^l]$, which is what the adversary would see in Hyb_3. The polynomial h is easily seen to have exactly the same distribution in Hyb_2 and Hyb_3. It follows that *Real* is indistinguishable from Hyb_3.

Note, that in the argument we just gave, we did not use anything special about the inputs a, b, c, other than $ab = c$. Therefore, essentially the same argument shows that the ideal process is also indistinguishable from Hyb_3 since the simulator uses $a = b = c = 0$ and otherwise follows the protocol. The argument now follows from transitivity of indistinguishability.

When a value of a corrupted party is opened, then it happens exactly like in the real process, hence, the simulation is perfect.

Finally, we need to argue, that when opening honest shared values, the simulation is indistinguishable. First note, that for the corrupted set of parties B, the value $v = \varphi_{r_B}(a) - c$ is in the same range as $\varphi_{r_B}(a)$ with all but negligible probability. Using v instead of $\varphi_{r_B}(a)$ will in the conversion generate a set of Shamir shares for the value c, since we now have,

$$r - v - \sum_{A \subseteq \mathcal{P}, |A|=t, A \neq B} \varphi_{r_A}(a) = c,$$

see, e.g., Example 5.1.

Finally, we need to argue that the published polynomial h in step 4 in the MultProof protocol is consistent with the published shares of f_c. We have that the adversary knows the polynomial f_c and,

$$h(i) = f(i) + f_a(i) f_b(i) + f_c(i),$$

for all $i \in B$, where f is the polynomial from the PRZS protocol. In the following we will argue, that even if the adversary knows f_a, f_b and f_c, then f could be generated such that it is consistent with h.

Assume without loss of generality that $|B| = t$. Then note, that even before the f_c polynomial is opened, the adversary knows t points on all polynomials h, f, f_a, f_b and f_c. That is, there are only t unknown points to the adversary on in the f polynomial.

The values $r_B^1, \ldots, r_B^t$ are unknown to the adversary, and F_B is given by,

$$F_B = \{f \mid \deg(f) \leq 2t, f(0) = 0, j \notin B \Rightarrow f(j) = 0\}.$$

That is, F_B is a subspace of F, the set of all polynomials of degree at most $2t$, which spans the subspace of f which is unknown by the adversary. F_B has basis $f_B^1, \ldots, f_B^t$, and for any given set of t points $\{\beta_j \mid P_j \notin B\}$, there exists a linear combination $\alpha_1, \ldots, \alpha_t$, such that $f' = f + \sum_{i=1}^{t} \alpha_i f_B^i$ has the following property, that $f'(0) = 0$, $f'(i) = s_i$ for all $P_i \in B$, where s_i is the share of P_i, and $f'(j) = \beta_j$ for t parties $P_j \notin B$.

By similar arguments as above, the $\varphi_{r_B^1}(a), \ldots, \varphi_{r_B^t}(a)$ values used in the PRZS protocol, can be exchanged to a random values. Hence, the polynomial f could be chosen to fit an arbitrary set of points, as described above. Hence, the indistinguishability follows. □

5.3.4 RISS-based protocols for General Adversary Structures

To generalize all our protocols based on RISS to general Q3 access structures, one simply replaces the polynomials over $\mathbb{Z}_p$ of degree at most t by sharings

using a monotone span program (MSP) $\mathcal{M}$ over $\mathbb{Z}_p$ (same as ISP's, but defined over $\mathbb{Z}_p$ and not $\mathbb{Z}$). These being linear, we can still convert RISS shares to this type using Proposition 5.2. $\mathcal{M}$ must be strongly multiplicative and realize the adversary structure Δ such an MSP always exists (see [30]). The polynomials of degree at most $2t$ are replaced by sharings using $\mathcal{M}^\diamond$, which is an MSP constructed from $\mathcal{M}$ by computing the tensor product of each row in $\mathcal{M}$ by itself and letting the results be the rows in $\mathcal{M}^\diamond$. Finally, the results on PRZS generalize in a straightforward way to any linear secret sharing scheme.

5.4 Interval Proofs and Application to Secure Computing

Boudot [14] observes that to prove that a number x lies in an interval $[a, b]$ it is sufficient to prove that $x - a \geq 0$ and $b - x \geq 0$. By using a homomorphic commitments scheme and a primitive to prove that a committed integer is a square, he constructs an efficient proof that a committed number is non-negative. Only a small constant number of calls to the primitive is required.

Boudot's protocols can be run in our settings by using one of the VSS protocols we have presented to play the role of commitments in Boudot's protocols. Note that both types of VSS's we construct are linear and so we have the homomorphic properties needed. In this way, we get a non-interactive proof that a shared number is in a given interval, using a constant number of invocations of our VSS protocol.

Unfortunately, Boudot's protocols use the random oracle, which we have been avoiding to use. However, using that every non-negative integer is the sum of four squares, i.e., for all $x \in \mathbb{Z}, x \geq 0$ there exists $x_1, x_2, x_3, x_4 \in \mathbb{Z}$ such that $x = x_1^2 + x_2^2 + x_3^2 + x_4^2$. Then we can use Boudot's observation to prove that a secret shared integer x lies in a public known interval $[a..b]$.

Furthermore, each number x we prove something about is verifiably shared among the parties, using a LISS scheme (a RISS scheme in case of the second protocol). If we consider the shares as numbers mod q for any prime q, we obtain a linear sharing over $\mathbb{Z}_q$ of x mod q. We can now, possibly after local conversion using [29], do secure computing on such numbers using, e.g., the protocols from [11, 37, 55]. If what we really want is secure addition and multiplication over the integers, we can use the initial interval proofs to make sure the numbers are small enough to avoid modular reductions.

Chapter 6

Efficiency of the Non-Interactive Multiplication Proofs

6.1 Introduction

Secure multi-party computation (MPC) is the problem where n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ compute some joint functionality $f\colon (x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$, where party P_i provides input x_i and receives output y_i, but without compromising the privacy of the their inputs and the correctness of their outputs. The most efficient solutions of MPC are done over a finite field. On the other hand, in many scenarios the actual computations should simulate computations over the integers, e.g., an online auction. The standard way to handle this, is to restrict the inputs of the parties to some public known interval $[a..b]$, then do the MPC over a field with big enough size to avoid any modular reduction. This is the common way to solve the problem. While it does *not* solve all the problems (see, e.g., Chapter 1), it is still sufficient in many situations. Note, that this only works *if* all parties input fulfill the constraint. However, in most scenarios the parties do *not* trust each other a priori, and hence, do *not* trust that the the other parties' input fulfill the constraint.

One way to get around this problem is to include a check in the MPC on the inputs, but this is an expensive matter and requires a lot of interaction and increases the round complexity. While doing MPC over the LISS scheme is more expensive than over a finite field, we proposed in Chapter 5 two non-interactive protocols which provide a zero-knowledge proof that a secret shared integer over LISS is contained in a given public interval $[a..b]$. Furthermore, the LISS scheme can be locally converted to a linear secret sharing scheme over any field.

Hence, we propose to let the parties provide their inputs as shares over the LISS scheme, use one of the two non-interactive zero-knowledge proofs from Chapter 5, locally convert the LISS shares to a linear secret sharing scheme over a field of reasonable size, and finally do the actual MPC. In this way we avoid the expensive interactive MPC check on their inputs.

In this chapter we compare this strategy to the general MPC strategy in terms of bit complexity. A standard way to compare protocols is by the multiplication complexity, since the multiplications require interaction. However,

in this case we need to compare an interactive with a non-interactive protocol. Therefore we compare on the total number of bit sent during the protocol execution.

6.2 Preliminaries

Let the n parties involved in the MPC be denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$. In a secret sharing scheme the subsets $A \subseteq \mathcal{P}$ that are jointly allowed to reconstruct the secret are called *qualified*, while other subsets are called *forbidden*. Note, that we require that the subsets of qualified subsets to be monotone.

The parties and the adversary attacking the protocol are all PPT interactive Turing machines. We assume non-authenticated channels between the parties in a synchronous communication network that runs in rounds. As a set-up assumption we assume that $P_1, \ldots, P_n$ get common input $pk_1, \ldots, pk_n$, where pk_i is the public key of P_i. Each party P_i gets private input sk_i, the corresponding secret key to pk_i.

Definition 6.1. *By the* bit complexity *of a protocol, we mean the total number of bits sent during a successful protocol execution by all parties involved, where all parties follow the protocol.*

Let $[x]$ denote a secret sharing of x.

Definition 6.2. *Let D denote a dealer that secret shares a value x among the set of parties $\mathcal{P}$, resulting in $[x]$. The* interval proof problem *is the following. D wants to publically prove that x is contained in the public interval $[a..b]$ without compromising the privacy of $[x]$ if $x \in [a..b]$.*

By a *comparison protocol* we mean a protocol that takes $[x]$ and $[y]$ as input and outputs $[x < y]$, where $[x < y]$ is a seccret sharing of 1 if and only if $x < y$ and 0 otherwise.

Given a comparison protocol it is obvious how to solve the interval proof problem. While this sounds straightforward the bit complexities are quite expensive. Nishide and Ohta [75] proposed a more direct approach where they exploit that they compare against some public values, but it only saves them a constant factor of the bit complexity.

In the previous chapter (Chapter 5) we proposed two protocols that solve the interval proof problem non-interactively. Both protocol are in the cryptographic model with a static active adversary. A *static active adversary* is an adversary that chooses a set of parties to corrupt before protocol execution, but that may deviate arbitrary from the protocol.

In the cryptographic model broadcast can basically be implemented by letting the dealer D send the message m signed $sig_{sk_D}(m)$ to all parties. Then the parties internally agree on what the dealer sent. This requires that each party sends the message once to everybody.

Hence, broadcasting m bits has bit complexity mn^2, where we assume that the overhead of the signature is not significant.

6.3 Comparing the Interval Proofs Approaches

In this section we compare the bit complexities of the different approaches of the interval proof problem.

The bit complexity is defined as the number of bits transmitted by all parties during an successful protocol execution, see Definition 6.1. First note, that the bit complexity of secret sharing an integer z over LISS and secret sharing a field element y over a linear secret sharing scheme is not the same, even though that z and y have the same bit-length. Secondly note, that we need to assume some access structure to compare the bit complexities. In the comparisons below we assume that the computations are done over a threshold-t access structure $T_{t,n}$ with $t < n/3$. For the linear secret sharing scheme $T_{t,n}$ can be realized using Shamir secret sharing [82], where each share consists of one field element. The LISS scheme can realize the threshold structure such that each share consists of $\log(n)$ integers of bounded size [32, 41].

In the following comparisons n will denote the number of parties involved, t the threshold, l the bit size of the element to be secret shared, k the statistical security parameter of the LISS scheme, and κ the security parameter for the commitment scheme and the public-key scheme.

6.3.1 Non-Interactive Interval Proofs (I)

In the first approach from Chapter 5 the cryptographic security relies on the integer commitment scheme and an identity-based encryption scheme. This approach can handle a $Q2$ static active adversary, but we only consider it in the $Q3$ scenario in order to able to compare it to the other approaches.

The dealer needs to broadcast

$$8n\log(n)(k + l + \kappa + n\log(n))$$

bits. Hence, the bit complexity is

$$8n^3\log(n)(k + l + \kappa + n\log(n))$$

bits.

Example 6.1. Let $n = 7$, $l = 64$, $k = 60$, and $\kappa = 1024$. The bit complexity of this example is

$$1 \cdot 10^7$$

bits.

Example 6.2. Let $n = 13$, $l = 128$, $k = 120$, and $\kappa = 2048$. The bit complexity of this example is

$$1.6 \cdot 10^8$$

bits.

6.3.2 Non-Interactive Interval Proofs (II)

In the second approach from Chapter 5 the cryptographic security relies on an identity based encryption scheme and a pseudorandom function. The protocol can handle a $Q3$ static active adversary.

This protocol is a bit more subtle to give a bit complexity for, since there is a setup-phase. The bit complexity of the setup phase is,

$$n^2\binom{n}{t}\kappa(t+1)$$

bits, if $\kappa > k$, which it is in all normal cases.

Furthermore, the protocol needs to broadcast three values of size,

$$\log\binom{n}{t} + l + k + 1$$

bits and three labels of undefined size, they should just not be re-used.

Finally, a polynomial is broadcast, which has size

$$2t(1 + 2(2 + \log\binom{n}{t}) + 2l + 2k)$$

bits

Example 6.3. Let $n = 7$, $t = 2$, $l = 64$, $k = 60$, and $\kappa = 1024$. The bit complexity of this example is

$$3.2 \cdot 10^6$$

bits.

Example 6.4. Let $n = 13$, $t = 4$, $l = 128$, $k = 120$, and $\kappa = 2048$. The bit complexity of this example is

$$1.2 \cdot 10^9$$

bits.

Note 6.1. The setup phase dominates the overhead of the protocol. If a party uses the protocol, say, 100 times, then the amortized cost is approximately cut by a factor 100. Hence, this protocol is very desirable if each party uses the protocol many times.

6.3.3 Standard Interval Proof

Nishide and Ohta [75] proposed an efficient protocol for interval test with bit complexity $110l + 1$ multiplications, where l is the bit-length of the prime field where the computations are done in.

We assume that a multiplication costs the transmission of n^3 field elements in total. Amortized better results can be achieved, see e.g. [40], but the number of multiplications needs to be so much greater in order to take advantage of the better amortized bound.

That is, the bit complexity is,

$$(110l + 1)ln^3$$

bits.

Example 6.5. For our example, that is, for $l = 64$ and $n = 7$, the bit complexity is

$$1.5 \cdot 10^8$$

bits.

Example 6.6. For our second example, that is, $l = 128$ and $n = 13$, the bit complexity is

$$4 \cdot 10^9$$

bits.

6.4 Discussion

In Example 6.1, 6.3, and 6.5 the parameters are $n = 7$, $t = 2$, $l = 64$, $k = 60$, and $\kappa = 1024$. The second approach from Section 6.3.2 has an slight advantage in the bit complexity. As pointed out in Note 6.1 the protocol amortized bit complexity drops almost by the factor of times it is used.

In Example 6.2, 6.4, and 6.6 the parameters are $n = 13$, $t = 4$, $l = 128$, $k = 120$, and $\kappa = 2048$. Here the first approach from Section 6.3.1 has the sligth advantage, but note that if the approach from section 6.3.2 is used at least 8 times, then it has the slight advantage on the amortized bit complexity.

If we compare the bit complexities as expressions of the parameters, then first note that they only have l and n in common. However, if we need, e.g., to prove that the number is at most a 32 bits in a 64 bit field, then obviously $l = 64$ in the protocol from section 6.3.3, but note that in the two approaches from section 6.3.1 and 6.3.2 we only need $l = 32$. Hence, the exact bit-complexities also depend on the specific scenario.

Furthermore, note that the n parameter is the dominant factor in all of the approaches, especially in the approach from section 6.3.2. However, in most likely scenarios n is not the dominant parameter.

One thing that is ignored in the comparisons above is the round complexity. The protocols proposed in Chapter 5 are non-interactive, hence optimal in the round complexity. The protocol proposed by Nishide and Ohta runs in 13 rounds.

Consider an online auction system that uses MPC distributed on some servers of different organizations with independent interests to ensure the privacy of the bids. Hence, each bidder secret shares their bid among the servers and lets them do the MPC. However, the auction system needs to verify that the input of the bidders are in some specified interval in order to make the computations in a field but avoiding any modular reductions. Hence, they need a protocol like the ones above. Say, that they use the one proposed by Nishide and Ohta in [75] and there are a lot of bidders, then the 13 rounds becomes a big factor compared to the non-interactive approaches from Chapter 5.

6.5 Further Considerations

Say that a dealer D needs to secret share $x_1, \ldots, x_\nu$ in increasing order $x_1 \leq \ldots \leq x_\nu$ among a set of parties which use this as input to some MPC. If D is not trusted and the increasing order is a requirement for the following MPC to succeed or be realizable, then the parties need to verify it. One way to solve the problem is to use a comparison protocol on $[x_i]$ and $[x_{i+1}]$ as input for $i = 1, \ldots, \nu - 1$ to verify the constraint.

Unfortunately the comparison protocol is quite expensive. The protocol proposed by Nishide and Ohta [75] requires $279l + 5$ multiplications and runs in 15 rounds.

The proposed protocols in Chapter 5 can also be used to solve the above problem. If using the parameters of the given examples in this note, the bit complexities are also lower than the approach in [75]. Furthermore, the protocols presented in Chapter 5 are non-interactive opposed to the 15 round comparison protocol by Nishide and Ohta [75].

Chapter 7

Conversion between Fields

7.1 Introduction

Secure multi-party computation (MPC) is the problem where n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ distributed compute some pre-determined functionality $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ privately, such that party P_i provides input x_i and receives output y_i, but without compromising the privacy of their input or the correctness of the outputs. While doing MPC it is often beneficial to do some computations on bits. For instance, Damgård et al. showed in [37] how some operations can be efficiently done over bit-sharings.

When efficiency is considered in bit-arithmetic the operation *xor* is crucial. If this operation is done on secret shared bits over a field of characteristic $p \neq 2$ it costs one multiplication. While on the other hand, if it is done in a field of characteristic 2 it can be done without interaction. Most often, the obtained bits are in a prime field $\mathbb{Z}_p$. Converting a shared bit b over $\mathbb{Z}_p$ to a field of characteristic 2 is therefore very useful. We denote this problem in the following by the *conversion problem*.

To our knowledge, the previous best solution to solve the conversion problem [85], is to let each party secret share a random number r_i over $\mathbb{Z}_p$ and share the least significant bit of r_i over a field of characteristic 2, called the *target field*. Furthermore, the random number r_i needs to be bounded by $\lfloor \log(p) \rfloor - \lfloor \log(n) \rfloor$ bits. This ensures, that the sum of all the parties random shared values $r = \sum_i r_i$ will not lead to a modular reduction. Hence, the sum of all the least significant bits in the target field will be the least significant bit of r. When this is done, the actual conversion of the secret shared bit b over the prime field, is done by opening $r + b$ and use the opened value to adjust the shared bit in target field. Observe that this protocol only handles the case of a passive adversary, which follows the protocol. For an active adversary, which may deviate from the protocol, each party needs to prove that the least significant bit is correct and that the random value is not too big.

In this chapter we present a novel technique to solve the conversion problem. Under the assumption that the parties have secret shared a random replicated value, the conversion problem can be solved by opening only one value. In the threshold-t case with no more than t corrupted parties the predistributed

replicated share consists of $\binom{n}{t}$ elements, where n is the number of parties. This might seem as a big value, but for small n this is still acceptable. In the case of a $t < n/3$ threshold, the protocol is secure against an active adversary.

Furthermore, the costs of the initial replicated secret share can be further reduced if assuming that pseudo-random functions (PRF) are secure. If the parties jointly generate one public random value, used as a seed for PRF, then the same random replicated secret share can be used.

7.2 Preliminaries

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote the n involved parties and let Γ be a monotone access structure over $\mathcal{P}$. Furthermore, let Δ^+ denote all the *maximal* unqualified sets of Δ, the corresponding adversary structure of Γ. In a replicated secret-sharing scheme [64] a random number r_T for each set $T \in \Delta^+$ is chosen, with the restriction that $s = \sum_{T \in \Delta^+} r_T$ equals the secret-shared value. Then each share r_T is given to all parties $P_j \notin T$. That is, the share of party P_j is $\{r_T\}_{P_j \notin T}$.

Privacy follows from the fact that members of every maximal unqualified set $T \in \Delta^+$ jointly miss exactly one additive share, namely the share r_T. Since Γ is monotone, a qualified set $Q \in \Gamma$ will jointly view all shares r_T and can thus reconstruct s.

Definition 7.1 (Share Conversion [29])**.** *Let S, S' be two secret-sharing schemes over the same secret-domain K. We say that S is* locally convertible *to S' if there exist local conversion functions $g_1, \ldots, g_n$ such that the following holds: if $(s_1, \ldots, s_n)$ are valid shares of a secret s in S, then $(g_1(s_1), \ldots, g_n(s_n))$ are valid shares of the same secret s in S'.*

In [29] Cramer et al. show how to locally convert a replicated secret-share s to a Shamir shares [82]. First note that the maximal unqualified sets of the $T_{t,n}$ threshold structure are given by $\Delta^+ = \{A \subseteq \mathcal{P} \mid |A| = t\}$. Thus, we have

$$s = \sum_{A \in \Delta^+} r_A,$$

where r_A has been given to all parties not in A.

Furthermore, let x_i be a unique point assigned to P_i. For each $A \in \Delta^+$, let f_A be the degree-t polynomial such that:

$$f_A(0) = 1,$$

and

$$f_A(x_i) = 0 \quad \forall P_i \in A.$$

Every party P_j can compute a share s_j as follows:

$$s_j = \sum_{A \in \Delta^+} r_A \cdot f_A(x_j),$$

which defines consistent shares of the polynomial

$$f = \sum_{A \in \Delta^+} r_A \cdot f_A.$$

First observe that by the locality of the above conversion, converted shares cannot reveal more information about the secret than the original shares.

Furthermore observe that the functions $f_A \in \mathbb{F}[X]$ and the values r_A might not be from the same domain. Say, if $r_A \in \mathbb{Z}$ and $f_A \in \mathbb{Z}_p[X]$, then the above conversions still is meaningful, since the map $\cdot : \mathbb{Z} \times \mathbb{Z}_p[X] \to \mathbb{Z}_p[X]$ can be seen as a $\mathbb{Z}$-module. In this case, the replicated share $s = \sum r_A$ will be locally converted to a Shamir sharing of $s \bmod p$. Furthermore, note if the functions were chosen from $GF(2^l)[X]$, i.e., polynomials over a field of characteristic 2, the locally conversion will result in a Shamir sharing of $s \bmod 2$.

The replicated secret sharing scheme from [64] is done over some field. In Chapter 5 we showed how the replicated secret-sharing scheme from [64] can be realized over the integers and proved secure.

First we formally define the scheme over the integers.

Scheme Replicated Integer Secret-Sharing (RISS) R_Γ

> Let Δ^+ be the maximal unqualified sets of an access structure Γ. For each set $T \in \Delta^+$ choose an uniformly random r_T integer from the interval $[-2^{l+k}..2^{l+k}]$ and send privately r_T to each party $P_i \notin T$. Furthermore, publish $r = s + \sum_{T \in \Delta^+} r_T$, where s is the secret from the interval $[-2^l..2^l]$.

Note that the public value $r = s + \sum_{T \in \Delta^+} r_T$ does not further complicate the conversion. Simply follow the protocol above, resulting in a polynomial $f \in \mathbb{F}[X]$ such that $f(0) = \sum r_T \in \mathbb{F}$. Then using the constant polynomial $g(x) = s + \sum r_T$ to locally obtain sharings of $F = g - f$, i.e., $F(0) = s$.

Given a RISS share $(r, \{r_T\})$ we denote each r_T a *share element*, not to confuse them with the share component of the LISS scheme. The *share* of party P_i consists of share elements $\{r_T\}_{P_i \notin T}$.

Note that even though the obtained Shamir sharing is well known to be secure, it does not necessarily mean that the RISS sharing is secure, or more precisely, correct and private. However, the RISS scheme was shown correct and private in Chapter 5.

As stated above, given a replicated share for a threshold structure it can be locally converted to a Shamir share, where the conversion is done as a $\mathbb{Z}$-module. In Chapter 5 a more general result was shown, but we only need the above conversion. Note, that by the locality of the conversion, no information is leaked in the conversion.

Furthermore, as noted above, if the target field of the share conversion has characteristic 2, then the resulting share will be the least significant bit of the RISS share.

7.3 Bit Conversion

In this section we will show how the replicated integer secret-sharing (RISS) can be used to convert a secret shared bit over a prime-field to a field of characteristic

2 as well as to an arbitrary field. Furthermore, we extend the conversion to be elements of bounded size from one field to another.

First some note on the usage of RISS shares in the following. A RISS share is denoted by $(r, \{r_A\})$, where the secret is given by $s = r - \sum r_A$, where $r = s + \sum r_A$. In the following we need a RISS share of a random value. This could obviously be done by choosing random $\{r_A\}$ and a random value r' and let $r = r' + \sum r_A$. Then $(r, \{r_A\})$ would be a random RISS share of random value r'.

However, consider the scenario where the dealer is corrupt. To deal a *consistent* RISS share, that is, that all parties P_i not in A agree on r_A for all $A \in \Delta^+$, can be done like described in Chapter 5. However, the problem is if we need to bound the RISS shared value. Then we can only upper bound it by,

$$|r| + |\Delta^+|2^k,$$

where we assume that all $|r_A| \leq 2^k$, which can be verified by the honest parties. Hence, the point of having the public r value is somewhat gone. Therefore, in the following when we consider a RISS share, it only denotes the replicated values $\{r_A\}$.

We introduce some notation. Let $[a]$ denote a RISS sharing of a, that is, it represents $\{r_A\}$ such that $a = \sum r_A$. Let $[a]_p$ denote a Shamir share of a over $\mathbb{Z}_p$, and finally we denote $[b]_2$ as a Shamir share over a field of characteristic 2 of a bit b.

7.3.1 Conversion to a Field of Characteristic 2

Assume that the parties have shared a RISS share $\{r_A\}$ for a threshold structure among them, with the restriction that all $0 \leq r_A < 2^k$ for all $A \in \Delta^+$. Choose p such that $p > r = \sum r_A$.

Then the protocol is as follows. Given $[b]_p$ for some bit b the goal is to convert it to $[b]_2$. By assumption we are given $[r]$ which can be locally converted to $[r]_p$ and $[r_0]_2$, where r_0 denotes the least significant bit of r. This is possible since $0 \leq r = \sum r_A < p$. First note, that the polynomials in the conversion to $[r_0]_2$ are taken from a field of characteristic 2, hence the share will be either 0 or 1, as earlier noted. Furthermore, since the least significant bit of $r \bmod p$ equals the least significant bit of $r \in \mathbb{Z}$.

Then open the value $[r+b]_p = [r]_p + [b]_p$ to reveal $r+b$. Note that the least significant bit of $(r+b)$ is $r_0 \oplus b$. Use this value to modify $[r_0]_2$ to obtain $[b]_2$.

Note that the only information leaked in the protocol is $(r+b)$, which statistically hides the bit b in the parameter k.

7.3.2 Conversion Between Two Prime Fields

Note, that there is nothing in the protocol from the above section that makes it necessary to restrict the target field to have characteristic 2.

Starting from $[r]$ obtain $[r]_p$ and $[r]_q$, where we have that $q, p > r$ and the replicated shares elements of $[r]$ are positive and bounded by 2^k. Then open the value $[r+b]_p$ and use $r + b \bmod q$ to modify $[r]_q$ to $[b]_q$.

Finally note, that we do not need to restrict the conversion to a bit b, we only need to require that the converted value is statistically hidden by the random value r. That is, if we want to convert a value $a \in [0..2^l]$ then we need r to be uniformly random in $[0..2^{l+k}]$ for the statistical security parameter k.

7.3.3 Multiple Conversions

Given random keys $\{r_A\}$ for a RISS share $[r]$ and a pseudo-random function (PRF) $\psi : \mathbb{Z}\times\mathbb{Z} \to [0..2^l-1]$ it is possible to make arbitrary many conversions as follows. The parties distributed agree upon a value a and use the keys $\{\psi_{r_A}(a)\}$ as the initial RISS share. Otherwise proceed as the protocol above.

A security proof that this can be done securely is given indirectly in the proof of Theorem 5.5 in Chapter 5, where the security is based on the PRF.

7.3.4 The Initial RISS Share

In the previous sections we have assumed that an initial random RISS share has been dealt among the parties. In this section we look at how this initial share can be distributed among the parties.

First note that, if we do *not* use the strategy with the PRF, we need an initial random RISS share for each conversion. If we are in the scenario where one party P_i knows the value of the bit to be converted and he has interest in keeping it secret, then obviously party P_i can do the random RISS share.

On the other hand, if we use the strategy based on the PRF or the bit to be converted is unknown, then we need an initial random RISS share that no party knows.

This can be solved by letting each party share a random value in RISS and using the sum of those. This ensures that an adversary that corrupts all but one party, will still not be able to determine the secret, since at least one k-bit value remains unknown to him. Furthermore, the one k-bit value unknown to the adversary is enough to statistically hide the converted bit or value.

If all parties share an RISS value with k-bit share elements, the resulting share elements will be $k + \log(n)$-bits, where n is the number of parties. This is not a problem, but should just be noted when choosing the size of the field to convert from.

If necessary, the parties can discard all but the k least significant bits of all the resulting share elements. This way, they avoid the potential $\log(n)$ bits overhead.

Part III

The Information Theoretic Model

Chapter 8

Information Theoretic Integer Commitment Scheme

8.1 Introduction

One of the most useful tools in cryptology is the *commitment* [58]. By a commitment we mean a tool com that the *committer* can use to secretly "store" a value, i.e., he simply uses $\mathsf{com}(v)$. We say that the committer *commits* to v, or that he has *committed* to v in $\mathsf{com}(v)$.

The commitment $\mathsf{com}(v)$ should be *binding* and *hiding*, that is, once the committer has committed the binding property should ensure, that he cannot *open* it to any other value $v' \neq v$, whereas the hiding property should ensure that the other involved parties are not able to extract any information about v from the commitment tool com.

The commitment can be based on some computational assumption, like the one presented in Chapter 5, that was based on the assumption that factoring is hard. Most commitment schemes based on some computational assumption are realized by some one-way function f. A committer commits to a value v by publishing $f(v, r)$, where r is some randomness. A major drawback of commitment schemes based on one-way functions is that they cannot be used in the information theoretic model, where the adversary has unbounded computing power. Assume that a commitment scheme based on some one-way function f was unconditional binding and hiding. Then let a dealer publish $f(v, r)$ as a commitment. By the hiding property there must be some r' such that $f(v, r) = f(v', r')$ for any $v' \neq v$, otherwise the adversary could use her unbounded computing power to exclude v' as a possible committed value. On the other hand, this implies that the malicious committer could use her unbounded computing power to generate an r'' for any v'' such that $f(v'', r'') = f(v, r)$, i.e., contradicting the binding property in the information theoretic model.

Hence, if we need an information theoretic secure commitment scheme, we need some other means than a one-way functionality to implement the scheme. One way to solve this is to let the committer secret share the value v among the set of parties. If the secret sharing is done correctly, then when the committer opens the commitment the parties can verify that it is done correctly. This way, the binding property follows, since the committer cannot open to another

value without being detected. On the other hand, if only an unqualified subset of parties cooperate they cannot extract any information about the committed value, hence, the hiding property is fulfilled.

The big challenge in this is to ensure that the secret shared value is shared "correctly" among the set of parties. In this chapter we use ideas inspired by Damgård and Nielsen [40] to construct a integer commitment scheme in the information theoretic model. Basically, their idea is as follows: in order to commit to ν values the committer secret shares values r, $y_1, \ldots, y_\nu$, gets a challenge x, and opens the linear combination $r + \sum_{i=1}^{\nu} x^i y_i$. They show that this ensures that the values $y_1, \ldots, y_\nu$ are correctly shared among the honest parties, which is the best we can hope for. In this chapter we modify their strategy to work over the integers and prove it secure in the UC framework.

8.2 Model and Definitions

In this section we define the UC functionality to capture the properties of a commitment scheme. Since we use the UC model to prove security, we first shortly describe the setting.

First of all we have a dealer (or committer) that we denote by D. There are $n > 2$ parties involved that are denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$. The adversary attacking the protocol is static and active, i.e., the adversary chooses the set of parties to corrupt before protocol execution, but may deviate arbitrary from the protocol. The dealer and all the parties are polynomial time bounded, while the environment and the adversary have unbounded computing power, and the simulator should be polynomial in the complexity of the adversary. We assume synchronous communication in rounds with secure authenticated point-to-point channels. Finally, we assume there is a broadcast channel available for the dealer and all the parties.

A commitment scheme consists of a commit protocol and an open protocol. The commitment scheme should be *binding*, that is, once the dealer has committed the value, it is *not* possible to open the commitment to any other value. Furthermore, the scheme should be *hiding*, that is, the commitment should not reveal any information about the committed value.

Note, that a commitment scheme cannot be both unconditional hiding and binding if there are only two parties involved, but it is possible here, since the protocols involve more than two parties and we assume secure authenticated point-to-point channels.

The commitment functionality is captured in Figure 8.1, which involves a dealer, the parties which the value is committed to, and an adversary attacking the protocol. Note, that the functionality captures that the commitment scheme should be additive homomorphic. That is, if D has any number of commitments, he can construct a commitment to any given linear combination of them, which can be verifiably opened by him.

We need to restrict the behavior of the environment $\mathcal{Z}$ on $\mathcal{F}_{\texttt{commit}}$. We need to restrict $\mathcal{Z}$ only to use the $(\mathsf{Input}, sid, a_1, \ldots, a_\nu, l)$ command if $|a_i| \leq 2^l$ for $i = 1, \ldots, \nu$.

Functionality $\mathcal{F}_{\texttt{commit}}$

The functionality proceeds as follows, running with the dealer D, parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{Input}, sid, a_1, \ldots, a_\nu, l)$ from D:
   (a) Store (v_i, a_i, l) under a unique variable name v_i for all $i = 1, \ldots, \nu$.
   (b) Wait one round. Broadcast $(\mathsf{Input}, D, v_1, \ldots, v_\nu, l)$ to all parties and the adversary.
2. Upon receiving $(\mathsf{Add}, sid, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ from D:
   (a) Verify that there exists stored tuples $(v_1, a_1, l_1), \ldots, (v_\eta, a_\eta, l_\eta)$, otherwise abort.
   (b) Store the tuple $(v, \sum_{i=1}^{\eta} c_i a_i, \max(c_1 + l_1, \ldots, c_\eta + l_\eta) + \eta - 1)$ under a unique variable name v.
   (c) Broadcast $(\mathsf{Add}, D, v, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ to all parties and the adversary.
3. Upon receiving (Open, sid, v) from party D:
   (a) Verify that there exists stored tuple (v, a, l), otherwise abort.
   (b) Verify that $|a| < 2^l$, if not then wait one round and broadcast $(\mathsf{Value}, D, v, \mathsf{Reject})$ and abort.
   (c) Wait one round. Broadcast $(\mathsf{Value}, D, v, a)$ to all parties and the adversary.

Figure 8.1: Functionality $\mathcal{F}_{\texttt{commit}}$.

8.3 Committing to Integers

In this section, we show how a dealer D can commit to integers with information theoretic security, assuming the adversary structure we deal with is $Q3$.

The protocol realizing the functionality needs to maintain a set C_D for the dealer D containing all parties who at some point accused D of cheating. If at any point C_D becomes so large that it is no longer in the adversary structure, D will be disqualified. Since then at least one honest party accused D.

For the moment, we assume an "oracle" that produces the random challenges, that is, integers chosen uniformly in a given interval. We discuss later how we can implement this. We also assume a broadcast channel which can, however, be implemented using a sub-protocol since the adversary structure is $Q3$.

The protocol can commit to any number of integers, $y_1, \ldots, y_\nu$, in one run. This comes in handy since it saves the overhead of the randomness r needed to "hide" the commitments. We assume that all committed values are bounded by the same public interval. This can be generalized such that each commitment

have each a public interval, but to simplify the notation this is neglected.

For notational convenience we use $[x]$ to represent the integer x secret shared over LISS. The protocol is as follows.

Protocol $\mathsf{Commit}(y_1, \ldots, y_\nu, l)$

1. Assume that $y_1, \ldots, y_\nu \in [-2^l..2^l]$, then choose $r \in [-2^{l+k'}..2^{l+k'}]$, where $k' = k + \lceil \log(\nu) \rceil + \ell$ and k is the information theoretic security parameter.
2. D secret shares $r, y_1, \ldots, y_\nu$ resulting in $[r], [y_1], \ldots, [y_\nu]$.
3. Each party verifies that each share component of $[y_1], \ldots, [y_\nu]$ is less than $2^{l_0+k+m_{\max}+\lceil \log(e) \rceil}$, where $m_{\max}$ is the maximal bit length of any entry in M from the ISP used in the protocol. If not he broadcasts (Accuse, D) and is added to C_D.
4. D receives challenges $c_1, \ldots, c_\nu \in [0..2^\ell]$.
5. D publically opens $[r] + \sum_{i=1}^{\nu} c_i[y_i]$, that is, D broadcasts a distribution vector $\boldsymbol{\rho}$ such that $M\boldsymbol{\rho} = [r] + \sum_{i=1}^{\nu} c_i[y_i]$.
6. Each party computes his share of $[r] + \sum_{i=1}^{\nu} c_i[y_i]$ and compares what D broadcasted. If there is disagreement, he broadcasts (Accuse, D) and is added to C_D.
7. If C_D is no longer in the adversary structure, D is disqualified, otherwise the commitment is accepted.

Remark 8.1. The verification in step 3 of the Commit-protocol will ensure that the share components of the honest parties are in the expected range. Furthermore, if the share component exceed this size, we know that the dealer is dealing out either rubbish or share components of a too big secret.

The following protocol lets a dealer D open one of his previously committed values or any linear combination of his commitments.

Protocol $\mathsf{Open}([y])$

1. D broadcasts the distribution vector $\boldsymbol{\rho}$ used to construct $[y]$.
2. If $|\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle| = |y| > 2^l$ then reject the opening and abort.
3. Each party not in C_D computes the shares resulting from D's message and compares to his share obtained at commitment time. If there is a disagreement, he broadcasts (Accuse, D).
4. All accusers are added to C_D. If C_D is now not in the adversary structure, D is disqualified and the opening rejected, else it is accepted.

Definition 8.1. *Let $\boldsymbol{y}$ be a share vector and let H be the set of honest parties not in C_D. Then $\boldsymbol{y}$ is said to be* consistently shared *if there exists an unique value y such that for all qualified $A \subseteq H$ then $\langle \boldsymbol{y}, \boldsymbol{\lambda}^A \rangle = y$, where $\boldsymbol{\lambda}^A$ is the reconstruction vector for A.*

Lemma 8.1. *Suppose D can complete the* $\mathsf{Commit}(y_1, \ldots, y_\nu, l)$ *protocol successfully and that the challenges were chosen uniformly at random, then the committed values are consistently shared with all but negligible probability in the security parameter ℓ.*

Proof. Let H be the set of all honest parties not in C_D and let $\boldsymbol{\rho}$ be the distribution vector that D provided in the protocol. Since the protocol completed successfully we have that $(M\boldsymbol{\rho})_H = \boldsymbol{z}_H$, where $\boldsymbol{z}$ is the share vector of $[z] = [r] + \sum_{i=1}^{\nu} c_i[y_i]$ and M is from the ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ used in the protocol. Let $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = z$, then for all qualified $A \subseteq H$ we have that $\langle \boldsymbol{z}, \boldsymbol{\lambda}^A \rangle = z$, where $\boldsymbol{\lambda}^A$ is the reconstruction vector for A. That is, $\boldsymbol{z}$ is consistently shared.

Let $\boldsymbol{r} = (y_{0,1}, \ldots, y_{0,d})^T$ be the share vector of $[r]$ and for $i = 1, \ldots, \nu$ let $\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,d})^T$ be the share vector of $[y_i]$. Finally, let $\boldsymbol{z} = (z_1, \ldots, z_d)^T$ be the consistent sharing of $[z]$.

Since $[z]$ is consistently shared we know that for all qualified $A \subseteq H$ it holds that $\sum_{i=1}^{d} \lambda_i^A z_i = z$ for a fixed z, where $\boldsymbol{\lambda}^A = (\lambda_1^A, \ldots, \lambda_d^A)^T$ is the reconstruction vector for A.

This also means, that

$$\sum_{i=1}^{d} \lambda_i^A z_i = \sum_{j=0}^{l} c_j \sum_{i=1}^{d} \lambda_i^A y_{j,i}.$$

If at least one of the shares is inconsistent, there must exist j and $A, A' \subseteq H$ such that

$$\sum_{i=1}^{d} \lambda_i^A y_{j,i} \neq \sum_{i=1}^{d} \lambda_i^{A'} y_{j,i}.$$

Define $v_{A,j} = \sum_{i=1}^{d} \lambda_i^A y_{j,i}$. Then we have, that

$$v_{A,j} \neq v_{A',j},$$

but

$$\sum_{j=0}^{l} c_j v_{A,j} = \sum_{j=0}^{l} c_j v_{A',j},$$

or equivalent

$$0 = \sum_{j=0}^{l} c_j (v_{A,j} - v_{A',j}).$$

If we define $\boldsymbol{v} = (v_{A,0} - v_{A',0}, \ldots, v_{A,l} - v_{A',l})^T$ and $\boldsymbol{c} = (c_0, \ldots, c_l)^T$. Then we have that,

$$\langle \boldsymbol{c}, \boldsymbol{v} \rangle = \sum_{j=0}^{l} c_j (v_{A,j} - v_{A',j}).$$

Since $\boldsymbol{c} \in [0..2^\ell]^{l+1}$ is chosen uniformly at random, we only need bound the probability that $\langle \boldsymbol{c}, \boldsymbol{v} \rangle$ is zero for non-zero $\boldsymbol{v}$ and uniformly random $\boldsymbol{c}$.

However, if we were working over a vector space, then we could make an orthogonal basis with $\boldsymbol{v}$ as the first basis vector. Then mapping $\boldsymbol{c}$ over to this

basis, write $\boldsymbol{c}_b$, would still be a random vector. Furthermore, $\langle \boldsymbol{c}, \boldsymbol{v} \rangle = 0$ if and only if $\langle \boldsymbol{c}_b, (1, 0, \ldots, 0)^T \rangle = 0$, which only happens with probability $2^{-\ell}$.

Since we are working over a $\mathbb{Z}$-module the change of basis is not guaranteed to exist. However, this is not a problem, since we only need to observe, that $\langle \boldsymbol{c}, \boldsymbol{v} \rangle = 0$ in the $\mathbb{Z}$-module, if and only if it is zero over a field. Hence, the probability follows. □

Theorem 8.1. *The* Commit *and* Open *protocols securely realize the functionality* $\mathcal{F}_{\texttt{commit}}$ *(Fig. 8.1) with an active adversary for an* $Q3$ *adversary structure.*

Proof. We need to construct a simulator that can generate a view such that any given adversary $\mathcal{A}$ cannot distinguish between the real and the ideal process. Let B denote the set of corrupted parties, which is chosen by $\mathcal{A}$ before the protocol execution is started.

The simulator acts as follows when required.

1. Upon receiving $(\mathsf{Input}, P_j, v_1, \ldots, v_\nu, l)$ from the functionality, then the simulator runs the $\mathsf{Commit}(0, \ldots, 0, l)$ protocol on behalf of honest party P_j with the value 0.

2. Upon receiving shares on behalf of all honest parties from $\mathcal{A}$, follows the protocol. If accepted, then reconstruct the committed values, to say $a_1, \ldots, a_\nu$, and send $(\mathsf{Input}, sid, a_1, \ldots, a_\nu, l)$ to the functionality on behalf of the corrupted party.

3. Upon receiving $(\mathsf{Add}, P_j, x, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ follow the protocol.

4. Upon receiving (Value, v, a) from the functionality, where the variable v refers to an *honest* committed value, i.e., a commitment of 0 in the protocol that the simulator has done. The simulator modifies the honest parties shares by $aM\boldsymbol{\kappa}$, where $\boldsymbol{\kappa}$ is the sweeping vector of the set of corrupted parties B.

5. If $\mathcal{A}$ opens a value, then just follow the protocol and send (Open, sid, v) to the functionality on behalf of the corrupted party.

First we argue that $\mathcal{Z}$ cannot distinguish the real from the ideal process based on case 1 in the simulation. In the real process we have that,

$$\begin{pmatrix} v \\ r_2 \\ \vdots \\ r_e \end{pmatrix} = \begin{pmatrix} r \\ r_{2,0} \\ \vdots \\ r_{e,0} \end{pmatrix} + \sum_{i=1}^{\nu} c_i \begin{pmatrix} y_i \\ r_{2,i} \\ \vdots \\ r_{e,i} \end{pmatrix},$$

for r chosen uniformly at random in $[-2^{l+k'}..2^{l+k'}]$ for $k' = k + \lceil \log(\nu) \rceil + \ell$, $y_i \in [-2^l..2^l]$, and uniformly random $c_i \in [-2^\ell..2^\ell]$. Whereas in the ideal process we have that,

$$\begin{pmatrix} v' \\ r_2 \\ \vdots \\ r_e \end{pmatrix} = \begin{pmatrix} r \\ r_{2,0} \\ \vdots \\ r_{e,0} \end{pmatrix} + \sum_{i=1}^{\nu} c_i \begin{pmatrix} 0 \\ r_{2,i} \\ \vdots \\ r_{e,i} \end{pmatrix},$$

hence, it is only v' that has a different distribution in the revealed distribution vector. Let $y = \sum_{i=1}^{\nu} c_i y_i$, then $|y| \leq 2^{l+\ell+\lceil \log(\nu) \rceil}$. We have that r is chosen uniformly at random from $[-2^{l+k'}..2^{l+k'}]$ where $k' = k + \lceil \log(\nu) \rceil + \ell$. Define v' to be *good* if lies in the same interval as v for a given y. Then the statistical distance of the distributions of v and v' is at most the probability that v' is not good. The probability that v' is not good is given by

$$\Pr(v' \text{ is not good}) = \frac{2^{l+\ell+\lceil \log(\nu) \rceil}}{2^{l+\ell+\lceil \log(\nu) \rceil + k}} = \frac{1}{2^k},$$

which is negligible in the security parameter k. Hence, the environment cannot distinguish with non-negligible probability between the real and ideal process based on case 1.

Note here, that if we did not restrict the environment $\mathcal{Z}$ to let the honest parties use the functionality $\mathcal{F}_{\texttt{commit}}$ as intended, then $\mathcal{Z}$ could distinguish the real from the ideal process as follows: Tell an honest party to commit to an integer a that is greater than 2^l. Say that $a = 2^{l+k+1000}$, then the share components in the real process would obviously reflect this. Whereas in the ideal process the simulator would only know that a value was secret shared and use 0 in its place when doing the secret sharing over LISS, resulting in share components with normal distribution. Hence the environment would easily distinguish the two cases.

If the protocol is accepted in case 2, then we need to argue, that the simulator can reconstruct to one unique value. However, this follows from Lemma 8.1.

There is no communication in case 3. Hence, no simulation is needed in this case.

In case 4 the simulator will receive the value and modifies the shares of the honest parties, as described. By the privacy of LISS this is statistically close to perfect. Note that the opened commitment $[a]$ can be linear dependent on other commitments. Say $[a] = c_1[b_1] + c_2[b_2]$ for known constants c_1 and c_2. Hence, opening $[a]$ makes some restrictions on the possible values of $[b_1]$ and $[b_2]$. This does not cause any problems, since the shares in the commitment of $[a]$ are modified by $aM\boldsymbol{\kappa} = (c_1 b_1 + c_2 b_2)M\boldsymbol{\kappa}$ and if $[b_1]$ is opened afterwards it will be modified by $b_1 M\boldsymbol{\kappa}$ which is consistent with the shares of $[a]$. Note, that this would not be possible for an adaptive adversary where $\boldsymbol{\kappa}$ would change depending on the set of corrupted parties.

In case 5 where a corrupted party opens a commitment, we need to argue, that if the opening is accepted, then he opens to the value that was reconstructed by the simulator. Say, that the simulator reconstructed to y. That the Open protocol was accepted implies that C_D is still in the adversary structure. Since the adversary structure is $Q3$, then we are assured that H, the set of honest parties not in C_D, is qualified. Since the shares of the parties in H agree with what D sends, they must agree on the value he claims is committed. Hence D must claim it was y, otherwise the Open protocol is rejected. □

Remark 8.2. Note that we only need the challenge to be chosen such that the set of challenges that D can answer are chosen with negligible probability. This is the only assumption used above. We can therefore let each party choose a bit

string and concatenate them all to form the challenge. This should be done such that the length of the string chosen by each party times the minimal number of honest parties is at least the security parameter ℓ above. If the actual value of the challenge is longer, this must be taken into account when choosing the range for r in the Commit protocol.

Chapter 9

Multi-Party Computations

9.1 Introduction

Secure *multi-party computation* (MPC) [9,24,57] is the problem where n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ want to compute some joint functionality $f\colon (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, i.e., $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where party P_i provides input x_i and receives output y_i. The goal is to do this joint computation without compromising the privacy of their inputs or the correctness of the outputs. During the execution a subset of parties $B \subset \mathcal{P}$ can be *corrupted* and may cooperate to break the privacy of the inputs and/or the correctness of the outputs. Informally speaking, the protocol is called *secure* if this is not possible for the parties in B.

The MPC problem can be solved in the *cryptographic model* where the corrupted parties are assumed to be poly-time bounded. In this model the security is based on some computational assumption. While this model suffices in many situations, it is sometimes preferable to have a stronger security notion. For instance, if it is very critical to keep the input secret, even years after the MPC computation, it might not be sufficient to base the security on some computational assumption. Hence, the *information theoretic model*, where the corrupted parties have unbounded computation power, is preferable in these situations. Both in the cryptographic and the information theoretic model an *adaptive adversary* can be considered, where the adversary can adaptively choose the parties to be corrupted during the protocol execution, or a *static adversary*, where the set of corrupted parties is chosen before protocol execution. Furthermore, a *passive* and a *active* adversary are considered, where the passive adversary follows the protocol but tries to obtain additional information from the protocol execution, whereas the active adversary may deviate arbitrarily from the protocol.

As pointed out in Chapter 1 secure MPC is in general solved over a finite field, whereas the problem at hand is a computation over the integers. This can be simulated over a prime field $\mathbb{Z}_p$, where p is chosen big enough to avoid any confusion on the interpretation of the result as an integer. However, this strategy has some drawbacks: (i) an upper bound on the input must be known before protocol execution, (ii) the arithmetic is done over $\mathbb{Z}_p$ and not $\mathbb{Z}$. For a

full description of the problems, see Chapter 1.

In [30] Cramer, Damgård, and Mauer showed how the MPC problem can be solved for general linear secret sharing scheme in the information theoretic model for an adaptive adversary, where they use Krachmar and Wigderson result [66] that linear secret sharing scheme over the fields are equivalent to the notion of *monotone span programs* (MSP).

The linear integer secret sharing (LISS) scheme is based on the notion *integer span programs* (ISP) introduced by Cramer and Fehr in [32], which is the equivalent to MSP but over the integers instead of a field. Since the constructions are made over the MSP in [30] we generalize their results to fit the ISP, though, with some differences.

One main difference is that we prove our construction secure in the universal composability (UC) framework proposed by Canetti in [19]. This makes it straightforward to compose our result with other protocols proved secure in the UC framework, due to the composition theorem in the UC model.

A standard way to secure a multiplication protocol against an active adversary is to use a commitment scheme. The commitment scheme from [30] is based on a bi-variable polynomial and can give *unconditional perfect* security. Unconditional perfect security is a special case of information theoretic security, where the simulation should be identical and not just statistically close. Since the LISS secret sharing scheme is a statistical secret sharing scheme, unconditional perfect security can never be achieved. We therefore change the commitment scheme from [30] to the scheme presented in Chapter 8, which is secure in the information theoretic settings but only with statistical security. Furthermore, the scheme from [30] is non-interactive whereas the one presented in Chapter 8 is interactive but has the advantage that it can commit to any number of commitments in the same run.

9.2 Multiplicative Property of LISS

Given two d-vectors $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_d)^T$, let $\boldsymbol{x} \diamond \boldsymbol{y}$ denote the vector containing all entries of the form $x_i y_j$, where $\psi(i) = \psi(j)$. Note that if d_i is the number of rows owned by party P_i, then $\boldsymbol{x} \diamond \boldsymbol{y}$ has $m = \sum_i d_i^2$ entries. Note also that if $\boldsymbol{x}$ and $\boldsymbol{y}$ are share vectors, then $\boldsymbol{x} \diamond \boldsymbol{y}$ can be computed locally by the parties.

Definition 9.1. *A* multiplicative ISP *is an ISP $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ for which there exists a vector $\boldsymbol{r} \in \mathbb{Z}^m$, called a* recombination vector*, such that for any two secrets s, s' and any $\boldsymbol{\rho}$, $\boldsymbol{\rho}'$, such that $\langle \boldsymbol{\varepsilon}, \boldsymbol{\rho} \rangle = s$ and $\langle \boldsymbol{\varepsilon}, \boldsymbol{\rho}' \rangle = s'$, it holds that*

$$ss' = \langle \boldsymbol{r}, M\boldsymbol{\rho} \diamond M\boldsymbol{\rho}' \rangle$$

We say that $\mathcal{M}$ is strongly multiplicative *if for any party subset A that is rejected by $\mathcal{M}$, $\mathcal{M}_{\bar{A}}$ is multiplicative.*

Cramer et al. [33] note that, as in the case of span programs over fields [30], for every access structure Γ there exists a (strongly) multiplicative ISP over Γ

if and only if Γ is $Q2$ ($Q3$). Furthermore, there exists an efficient algorithm to construct a multiplicative ISP from any ISP. The result is given below.

Let E be the matrix with zero everywhere, except in its upper-left corner where the entry is one.

Lemma 9.1. *Let $\mathcal{M}_0 = (M_0, \boldsymbol{\varepsilon}, \psi)$ and $\mathcal{M}_1 = (M_1, \boldsymbol{\varepsilon}, \psi)$ be ISPs that compute Γ_0 and Γ_1, respectively, and where M_0 and M_1 are $d \times e$ matrices, where the mapping ψ is identical for both ISPs. If $M_0^T M_1 = E$, then one can construct a multiplicative ISP computing $\Gamma_0 \cup \Gamma_1$ of size at most $2d$.*

The proof of the lemma only uses basic properties of span programs and is done in [30]. We give it below for completeness.

Lemma 9.1. Before we construct the multiplicative ISP, we make some observations. If we share secret s under $\mathcal{M}_0$ and $\mathcal{M}_1$, i.e., we select random $\boldsymbol{\rho}_0$ and $\boldsymbol{\rho}_1$, such that $\langle \boldsymbol{\varepsilon}, \boldsymbol{\rho}_0 \rangle = \langle \boldsymbol{\varepsilon}, \boldsymbol{\rho}_1 \rangle = s$, and by computing the vector $(\boldsymbol{s}_0, \boldsymbol{s}_1) = (M_0\boldsymbol{\rho}_0, M_1\boldsymbol{\rho}_1)$, and give the i-th entry of $\boldsymbol{s}_0$ and $\boldsymbol{s}_1$ to party $P_{\psi(i)}$. This implies that any subset A that is qualified w.r.t. either Γ_0 or Γ_1 can reconstruct the secret.

If we share another secret s', i.e., we have that $(\boldsymbol{s}'_0, \boldsymbol{s}'_1) = (M_0\boldsymbol{\rho}'_0, M_1\boldsymbol{\rho}'_1)$, such that $\langle \boldsymbol{\varepsilon}, \boldsymbol{\rho}'_0 \rangle = \langle \boldsymbol{\varepsilon}, \boldsymbol{\rho}'_1 \rangle = s'$. Then if we let $\boldsymbol{s}_0 * \boldsymbol{s}'_1$ denote the d-vector obtained by coordinate-wise multiplication of $\boldsymbol{s}_0$ and $\boldsymbol{s}'_1$. Then it follows that

$$\langle \mathbf{1}, \boldsymbol{s}_0 * \boldsymbol{s}'_1 \rangle = \boldsymbol{s}_0^T \boldsymbol{s}'_1 = \boldsymbol{\rho}_0^T M_0^T M_1 \boldsymbol{\rho}_1 = \boldsymbol{\rho}_0^T E \boldsymbol{\rho}_1 = ss'$$

Because the i-th coordinate of $\boldsymbol{s}_0$ and of $\boldsymbol{s}'_1$ are held by the same party, an additive share of the product can be computed locally by the parties.

We now construct a multiplicative ISP $\mathcal{M} = (M, \boldsymbol{\varepsilon}, \psi')$ computing $\Gamma_0 \cup \Gamma_1$, where M is a $2d \times (2e-1)$ matrix. Let M' denote the matrix that has M_0 in the upper left corner, and all but the first column of M_1 in the lower right corner. Then M is the matrix for which the column vector consisting of d zeros and followed by the first column of M_1 is added to the first column of M'. This ISP $\mathcal{M}$ clearly computes $\Gamma_0 \cup \Gamma_1$, and it is also clear that the vector $(\boldsymbol{s}_0, \boldsymbol{s}_1) \diamond (\boldsymbol{s}'_0, \boldsymbol{s}'_1)$ contains the entries of $\boldsymbol{s}_0 * \boldsymbol{s}'_1$. Hence the vector with 1's corresponding to these entries and 0's elsewhere is the recombination vector. □

Let $\Gamma^* = \{A \mid A^c \notin \Gamma\}$ denote the *dual* access structure to Γ.

Remark 9.1. If Γ is $Q2$ then it holds that for each $A \notin \Gamma$ implies $A^c \in \Gamma$ and thus $A \notin \Gamma^*$. Hence $\Gamma = \Gamma \cup \Gamma^*$, if Γ is in $Q2$.

Remark 9.2. If we are given a LISS scheme $\mathcal{M} = (M, \boldsymbol{\varepsilon}, \psi)$ computing any $Q2$ access structure Γ, and we can construct a LISS scheme $\mathcal{M}^* = (M^*, \boldsymbol{\varepsilon}, \psi)$ computing Γ^* such that $M^T M^* = E$. Then Lemma 9.1 gives the construction of a multiplicative ISP that computes $\Gamma = \Gamma \cup \Gamma^*$.

The following proposition holds for any principal ideal domain, but we only need the result for the integers $\mathbb{Z}$.

Proposition 9.1. *If $\mathcal{M} = (M, \boldsymbol{\varepsilon}, \psi)$ computes Γ, and M is a $d \times e$ matrix, then we can assume without loss of generality that $e \leq d$.*

Proof. Let $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_e$ denote the e columns of M. First we show, that any multiple of the first column cannot be linearly dependent on the other columns. Assume for contradiction, that there existed constants $k_1 \ldots, k_e$ such that

$$k_1 \boldsymbol{c}_1 = k_2 \boldsymbol{c}_2 + \cdots + k_2 \boldsymbol{c}_e,$$

then for any $A \in P(\mathcal{P})$ it would hold that

$$k_1 \boldsymbol{c}_{1_A} = k_2 \boldsymbol{c}_{2_A} + \cdots + k_e \boldsymbol{c}_{e_A}.$$

If $A \in \Gamma$ then there exists $\boldsymbol{\lambda}$ such that $M_A^T \boldsymbol{\lambda}_A = \boldsymbol{\varepsilon}$, or more precisely, that $\langle \boldsymbol{c}_{1_A}, \boldsymbol{\lambda}_A \rangle = 1$ and $\langle \boldsymbol{c}_{i_A}, \boldsymbol{\lambda}_A \rangle = 0$ for all $i = 2, \ldots, e$. However, this is a contradiction since we assumed that a multiple of $\boldsymbol{c}_1$ was linearly dependent on the $e-1$ other columns. Hence, the first column must be linearly independent on the other $e-1$ columns.

Assume that $\boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_d$ span the same space as $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$. Assume that $A \in \Gamma$ then there exists $\boldsymbol{\lambda}$ such that $\langle \boldsymbol{c}_{i_A}, \boldsymbol{\lambda}_A \rangle = 0$ for all $i = 2, \ldots, e$. Furthermore, for any $i = 2, \ldots, d$ it must hold that $\langle \boldsymbol{c}'_{i_A}, \boldsymbol{\lambda}_A \rangle = 0$, since $\boldsymbol{c}'_i$ is in the span of $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$, i.e., $\boldsymbol{c}'_i = k_2 \boldsymbol{c}_2 + \cdots + k_e \boldsymbol{c}_e$ for some constants. On the other hand, if $A \notin \Gamma$ then there exists $\boldsymbol{\kappa} = (1, \kappa_2, \ldots, \kappa_e)^T$ such that

$$-\boldsymbol{c}_{1_A} = \kappa_2 \boldsymbol{c}_{2_A} + \cdots + \kappa_e \boldsymbol{c}_{e_A}.$$

Since the columns $\boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_d$ span the same space as $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$ there must exist constants $\kappa'_2, \ldots, \kappa'_d$ such that

$$-\boldsymbol{c}_{1_A} = \kappa'_2 \boldsymbol{c}'_{2_A} + \cdots + \kappa'_e \boldsymbol{c}'_{d_A}.$$

Hence, the 2nd to the e-th column can be changed by any other columns that span the same space.

Finally, we need to show, that we can always find $d-1$ columns that will span the same space as $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$. However, this can be done as follows. Let $c_{i,j}$ denote the j-th entry in column $\boldsymbol{c}_i$. Then let $c_1 = \gcd(c_{2,1}, \ldots, c_{e,1})$ and let $\boldsymbol{c}'_2$ be a column obtained by linear combination of $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$ such that the first entry of $\boldsymbol{c}_2$ is c_1. Now define $\boldsymbol{c}^1_2, \ldots, \boldsymbol{c}^1_e$ to be the columns where the first entry has been added out by $\boldsymbol{c}'_2$, i.e., $\boldsymbol{c}^1_i = \boldsymbol{c}_i - k\boldsymbol{c}'_2$, for some constant k such that the first entry in $\boldsymbol{c}^1_i$ is zero. Then proceed this procedure, but now on the second entry and so forth. This result in d columns, $\boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_{d+1}$. However, since any multiple of the first column is linearly independent of the $e-1$ other columns, and they map into an d dimensional space, the procedure must end with $d-1$ column. Assume for a contradiction, that we end with d columns, $\boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_{d+1}$, which obviously are linearly independent by construction.

Now assume that $\boldsymbol{c}_1, \boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_{d+1}$ all are linearly independent, but then we can proceed with the above strategy and end up with d columns. To complete the argument, we need to show that the resulting d vectors span $\boldsymbol{c}_1$. Furthermore, we need to show that the resulting $d-1$ columns, $\boldsymbol{c}'_2, \ldots, \boldsymbol{c}'_d$, span the same space as $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_e$. We argue both claims at the same time. Obviously, the resulting columns span a subspace of the original columns span. Hence, all we need to show is, that the new column space spans $\boldsymbol{c}_i$ for all i. However, this is clear, since the process which obtained the columns is reversable, hence, we can get back to any column we started with. □

Lemma 9.2. *Given an ISP $\mathcal{M} = (M, \boldsymbol{\varepsilon}, \psi)$ computing some access structure Γ, then an ISP $\mathcal{M}^* = (M^*, \boldsymbol{\varepsilon}^*, \psi)$ of the same size, computing the dual access structure Γ^* that satisfies $M^T M^* = E$ can be efficiently constructed from $\mathcal{M}$.*

This lemma can be proven in various ways [32, 46, 52, 74].

Lemma 9.2 [32]. Let $\mathcal{M} = (M, \boldsymbol{\varepsilon}, \psi)$ be an ISP for Γ. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_l$ be an arbitrary generating set of vectors for $\ker(M^T)$, and choose $\boldsymbol{\lambda}$ such that $M^T\boldsymbol{\lambda} = \boldsymbol{\varepsilon}$. Let M^* be the matrix defined by the $l+1$ columns $(\boldsymbol{\lambda}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_l)$, and use ψ to label M^*. Define $\mathcal{M}^* = (M^*, \psi, \boldsymbol{\varepsilon}^*)$, where $\boldsymbol{\varepsilon}^* = (1, 0, \ldots, 0)^T \in \mathbb{Z}^{l+1}$. Note that $\text{size}(\mathcal{M}^*) = \text{size}(\mathcal{M})$.

If $A^c \notin \Gamma$, then by definition, there exists $\boldsymbol{\kappa}$ such that $M_{A^c}\boldsymbol{\kappa} = \mathbf{0}$ and $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon}^* \rangle = 1$. Define $\boldsymbol{\lambda}^* = M_A\boldsymbol{\kappa}$. Then $(M^*)_A^T\boldsymbol{\lambda}^* = ((M^*)^T \cdot M)\boldsymbol{\kappa} = \boldsymbol{\varepsilon}^*$. On the other hand, if $A^c \in \Gamma$, then there exists $\hat{\boldsymbol{\lambda}}$ such that $M^T\hat{\boldsymbol{\lambda}} = \boldsymbol{\varepsilon}$ and $\hat{\boldsymbol{\lambda}}_A = \mathbf{0}$. By definition of M^*, there exists $\boldsymbol{\kappa}$ such that $M^*\boldsymbol{\kappa} = \hat{\boldsymbol{\lambda}}$ and $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon}^* \rangle = 1$. This follows from the fact, that M^* contains one solution to $M^T\boldsymbol{\lambda} = \boldsymbol{\varepsilon}$ and spans the kernel of M^T. Hence, $M_A^*\boldsymbol{\kappa} = \hat{\boldsymbol{\lambda}}_A = \mathbf{0}$ and $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon}^* \rangle = 1$. Finally, note that M^* obviously satisfies that $M^T M^* = E$. □

Remark 9.3. Note, that Proposition 9.1 ensures, that the dual matrix M^* as well as the original M can be constructed such that they have less than d columns each, where d is the size of M. This comes in handy when calculating shares, since it restricts the number of random values needed to be less or equal to the number of share components.

The following theorem follows immediately from Lemma 9.1 and 9.2 and Remark 9.1 and 9.2.

Theorem 9.1. *There exists an efficient algorithm which, on input of an ISP $\mathcal{M}$ computing a $Q2$ adversary structure, outputs a multiplicative ISP $\mathcal{M}'$ computing the same adversary structure and of size at most twice that of $\mathcal{M}$.*

As in the case of span program over fields, no general efficient construction is known for a strongly multiplicative ISP. In [32] Cramer and Fehr show that the constructed ISP for any threshold-t structure $T_{t,n}$ is (strongly) multiplicative if and only if the threshold $t < n/2$ ($t < n/3$), where n is the number of parties involved.

9.3 Multi-Party Computation for Passive Adversary

9.3.1 Model and Definitions

We use the UC framework to prove security and define a UC functionality for the MPC features. However, first we shortly describe the settings in the UC framework.

There are n parties involved in the computations, that are denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$. The adversary attacking the protocol is static and passive, that is, the adversary chooses a set of parties, from a given adversary structure, to corrupt before protocol execution, but follows the protocol as intended. In Section 9.4 we let the adversary be active, that is, she may deviate

arbitrary from the protocol. All the parties are polynomial time bounded, while the environment and adversary are computational unbounded, and the simulator should be polynomial in the complexity of the adversary. We assume synchronous communication in rounds with secure authenticated point-to-point channels. Finally, we assume there is a broadcast channel available for all parties.

The ideal functionality is defined in Figure 9.1. The functionality has four commands. The Input command takes an integer and a bit-length as input and stores them, which can be done by any party. Note, that the bit-length is a publically known value. The Add command stores a new integer that is any linear combination of previously stored integers. The only requirement is, that all honest parties agree on the linear combination. Furthermore, there is a Mult command that stores an integer that is the product of two previously stored integers. Again here all honest parties should agree on the input values. Finally, there is an Open command that opens any stored integer if all honest parties agree. Note, that if the opened integer has greater bit-length than the bit-length stored with it, the value is rejected.

Remark 9.4. Note, that the bit-length l assigned to each input value should not be thought of as the actual bit-length, since this would reveal some information about the secret. In the normal scenario the bit-length of all input values is agreed on before the protocol begins. The reason not to model the functionality this way is that from time to time some inputs have either smaller or bigger size, e.g., say that all but one input have size 32 bit where the one input deviating from this size has size 128 bit. Hence, this way we avoid having all inputs use randomness to hide a 128 bit integer.

The $(\mathsf{Input}, sid, a, l)$ command in the $\mathcal{F}_{\mathtt{MPC}}$ functionality broadcasts $(\mathsf{Input}, P, v, l)$, where P is the party which it received the command from, v is a variable name under which the secret a is stored, and l is the bit-length of a, i.e., $|a| \leq 2^l$. The bit-length l is the upper bound to statistically hide *any* secret numerically less than 2^l (see Remark 9.4), hence, it is a measure of the randomness needed in the sharing process.

As shortly described in Chapter 2, when proving a protocol secure in the UC [19] model one needs to describe a simulator in the ideal process that provides a view indistinguishable from the real process where the a real execution of the protocol is run. In the ideal process the simulator should provide the view based on the information received from the functionality. If the environment $\mathcal{Z}$ uses the $(\mathsf{Input}, sid, a, l)$ on a value $|a| > 2^l$, then in the real process the secret sharing in LISS of a is not guaranteed to hide a anymore. However, in the ideal process where a is unknown and only the bit-length l is known, then it will obviously not reveal any information about a. Hence, the two processes can be distinguished in this manner by $\mathcal{Z}$.

Hence we restrict the environment $\mathcal{Z}$ only to use the $(\mathsf{Input}, sid, a, l)$ command if $|a| \leq 2^l$.

Note 9.1. Another approach would be to let the functionality only accept input for which $|a| \leq 2^l$. However, this does not model the protocol, since when a party secret shares a value a it is not always decidable whether $|a| > 2^l$ or not.

Functionality $\mathcal{F}_{\texttt{MPC}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{Input}, sid, a, l)$ from some party P_j:
   (a) Store (v, a, l) under a unique variable name v.
   (b) Broadcast $(\mathsf{Input}, P_j, v, l)$ to all parties and the adversary.
2. Upon receiving $(\mathsf{Add}, sid, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ from all honest parties:
   (a) Verify that there exists stored tuples $(v_1, a, l_1), \ldots, (v_\eta, a_\eta, l_\eta)$, otherwise abort.
   (b) Store tuple $(x, \sum_{i=1}^{\eta} c_i a_i, \max\{\lceil \log c_1 \rceil + l_1, \ldots, \lceil \log c_\eta \rceil + l_\eta\} + \eta - 1)$ under a unique variable name x. Broadcast $(\mathsf{Add}, x, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ to all parties and the adversary.
3. Upon receiving $(\mathsf{Mult}, sid, v, w)$ from all honest parties:
   (a) Verify that there exists stored tuples (v, a, l_a) and (w, b, l_b), otherwise abort.
   (b) Wait one round, then store tuple $(x, ab, l_a + l_b)$ under a unique variable name x. Broadcast $(\mathsf{Mult}, x, v, w, l_a, l_b)$ to all parties and the adversary.
4. Upon receiving (Open, sid, v) from all honest parties:
   (a) Verify that there exists stored tuple (v, a, l), otherwise abort.
   (b) Verify that $|a| < 2^l$, if not, broadcast $(\mathsf{Value}, v, \mathsf{Reject})$ and abort.
   (c) Broadcast (Value, v, a) to all parties and the adversary.

Figure 9.1: Functionality $\mathcal{F}_{\texttt{MPC}}$.

Furthermore, this would give problems when simulating the protocol in the ideal process. If a corrupted party secret shares a value a that is out of bound, however, it is not decidable whether it is out of range by the protocol. Hence, the LISS sharing is accepted. However, when the simulator reconstructs the value and on behalf of the corrupted party send a as input to the functionality, the functionality will reject the input, since it is out of bound.

9.3.2 The Protocols and the Indistinguishability Proof

Let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ be a multiplicative ISP for Γ, and let a and b be two secrets from the intervals $[-2^{l_a}..2^{l_a}]$ and $[-2^{l_b}..2^{l_b}]$, respectively. Let $\boldsymbol{a} = (a_1, \ldots, a_d)^T = M\boldsymbol{\rho}_a$ and $\boldsymbol{b} = (b_1, \ldots, b_d)^T = M\boldsymbol{\rho}_b$, where $\boldsymbol{\rho}_a$ and $\boldsymbol{\rho}_b$ are chosen uniformly at random from $[-2^{(l_a)_0+k}..2^{(l_a)_0+k}]^e$ and $[-2^{(l_b)_0+k}..2^{(l_b)_0+k}]^e$, respectively, with the restriction that $\langle \boldsymbol{\rho}_a, \boldsymbol{\varepsilon} \rangle = a$ and $\langle \boldsymbol{\rho}_b, \boldsymbol{\varepsilon} \rangle = b$. The share

components a_i and b_i are given to party $P_{\psi(i)}$ for $i = 1, \ldots, d$. Note, that any linear combination of any secret sharings can be done locally. Say, we want to have a secret sharing of $s = c_a a + c_b b$ for publically known constants c_a and c_b. This is done by letting $s_i = c_a a_i + c_b b_i$ for all $i = 1, \ldots, e$. This is done locally by party $P_{\psi(i)}$. Hence, the challenge is to make a secret sharing of the product of two secret sharings. That is, the goal is to compute $\boldsymbol{c} = (c_1, \ldots, c_d)^T$ such that $\boldsymbol{c}$ is a sharing over $\mathcal{M}$ of the product $c = ab$ in such a way, that no information is leaked about a and b. The protocols are as follows.

Protocol $\mathsf{Add}(a_1, \ldots, a_\nu, c_1, \ldots, c_\nu)$

The input of the protocol are shares $\boldsymbol{a}_1 = (a_{1,1}, \ldots, a_{1,d})^T, \ldots, \boldsymbol{a}_\nu = (a_{\nu,1}, \ldots, a_{\nu,d})^T$, of $a_1, \ldots, a_\nu$, respectively. The protocol should output a secret sharing of $c = \sum_{i=1}^{\nu} c_i a_i$.

1. For $j = 1, \ldots, d$ party $P_{\psi(j)}$ lets $s_j = \sum_{i=1}^{\nu} c_i a_{i,j}$.

Then $\boldsymbol{s} = (s_1, \ldots, s_d)^T$ gives the desired secret sharing.

Protocol $\mathsf{Multiplication}(a, b)$

The input of the protocol is secret shared values of a and b from publically known intervals $[-2^{l_a}..2^{l_a}], [-2^{l_b}..2^{l_b}]$, respectively. The shares are represented by $\boldsymbol{a} = (a_1, \ldots, a_d)^T$ and $\boldsymbol{b} = (b_1, \ldots, b_d)^T$, respectively. The protocol outputs a secret sharing of $c \in [-2^{l_c}..2^{l_c}]$, where $l_c = l_a + l_b$. Define l to be the maximal bit-lenght of any share component used in the protocol.

1. For $\ell = 1, \ldots, n$, P_ℓ computes $a_i \cdot b_j = c_{(i,j)}$ for all $i, j \in \psi^{-1}(\ell)$.
2. *Re-sharing step:* P_ℓ secret shares $c_{(i,j)}$ using a uniformly random distribution vector $\boldsymbol{\rho}_{(i,j)} \in [-2^{l_0+k}..2^{l_0+k}]^d$ with the restriction that $\langle \boldsymbol{\rho}_{(i,j)}, \boldsymbol{\varepsilon} \rangle = c_{(i,j)}$, which results in share components $(c_{(i,j)_1}, \ldots, c_{(i,j)_n})^T = M\boldsymbol{\rho}_{(i,j)}$, and for $\iota = 1, \ldots, d$ P_ℓ sends $c_{(i,j)_\iota}$ to party $P_{\psi(\iota)}$.
3. *Recombination step:* For $\iota = 1, \ldots, d$, party $P_{\psi(\iota)}$ computes $c_\iota = \sum_{i,j} r_{(i,j)} c_{(i,j)_\iota}$, where $\boldsymbol{r} = (r_{(1,1)}, \ldots, r_{(d,d)})^T$ is the recombination vector of $\mathcal{M}$.

First we need the following lemma.

Lemma 9.3. *The recombination vector for a multiplicative ISP has at least one non-zero entry corresponding to an honest re-sharing.*

Proof. Let $R = \{P_{\psi(i)} \mid r_{(i,j)} \neq 0\}$, where $\boldsymbol{r} = (r_{(1,1)}, \ldots, r_{(d,d)})^T$ is the recombination vector for the multiplicative ISP $\mathcal{M}$. We claim that R is not in Δ the adversary structure. Suppose for a contradiction that $R \in \Delta$. If $R \in \Delta$ then by definition there exists a sweeping vector $\boldsymbol{\kappa}$ such that $M_R\boldsymbol{\kappa} = \mathbf{0}$ with $\langle \boldsymbol{\kappa}, \boldsymbol{\varepsilon} \rangle = 1$. Then by the multiplication property, we have that $1 = \langle \boldsymbol{r}, M\boldsymbol{\kappa} \diamond M\boldsymbol{\kappa} \rangle$. However, we also have that, $M_R\boldsymbol{\kappa} \diamond M_R\boldsymbol{\kappa} = \mathbf{0}$ and $\boldsymbol{r}_{R^c} = \mathbf{0}$, which implies that $\langle \boldsymbol{r}, M\boldsymbol{\kappa} \diamond M\boldsymbol{\kappa} \rangle = 0$, a contradiction. □

Note 9.2. Another way to argue Lemma 9.3 is the following. If it did not hold, the adversary could simply make a secret sharing of 1 and run the multiplication protocol on some unknown secret sharing a, which would not involve the honest parties. Under the assumption that the protocol is correct the corrupted parties would then have enough information to extract the value a. Hence, breaking the privacy of the LISS scheme.

Note 9.3. If a party P_ℓ needs to re-share more than one share component, he can recombine them before sending the components to the other parties, i.e., each party only needs to re-share one value.

Protocol Open(a)

Let the shares of a be represented by $\boldsymbol{a} = (a_1, \ldots, a_d)^T$. Let o be a secret sharing of 1 represented by $\boldsymbol{o} = (o_1, \ldots, o_d)^T$.

1. Run the Multiplication protocol on $\boldsymbol{a}$ and $\boldsymbol{o}$ resulting in $\boldsymbol{a}'$.
2. All parties reveal their part of $\boldsymbol{a}'$.

The above protocol is only needed to simplify the proof of the following theorem (Theorem 9.2) significantly. Note, that the theorem can be proved without the Open protocol, but since the full proof without the Open protocol is not technical hard, but just long and cumbersome, we chose to add the Open protocol for the great simplification.

Lemma 9.4. *Using the notation from the* Multiplication *protocol. Let* $\boldsymbol{\delta}$ *be chosen such that* $\sum_{(i,j)} r_{(i,j)}\delta_{(i,j)} = 0$*, where* $\boldsymbol{r}$ *is the recombination vector. If* $c'_{(i,j)} = c_{(i,j)} + \delta_{(i,j)}$*, then the final shares after the recombination step are the same.*

Proof. The claim is that after the protocol has finished the $c_{(i,j)}$'s can be arbitrary with the restriction that they still sum to c, the value of the multiplication. Let $c'_{(i,j)} = c_{(i,j)} + \delta_{(i,j)}$ such that $\sum_{(i,j)} r_{(i,j)}\delta_{(i,j)} = 0$, i.e., $\sum_{(i,j)} r_{(i,j)}c'_{(i,j)} = c$. Then observe that

$$\begin{aligned}
\boldsymbol{\rho} &= \sum_{(i,j)} r_{(i,j)} \left(\boldsymbol{\rho}_{(i,j)} + \delta_{(i,j)}\boldsymbol{\kappa}\right) \\
&= \sum_{(i,j)} r_{(i,j)}\boldsymbol{\rho}_{(i,j)} + \boldsymbol{\kappa}\sum_{(i,j)} r_{(i,j)}\delta_{(i,j)} \\
&= \sum_{(i,j)} r_{(i,j)}\boldsymbol{\rho}_{(i,j)}.
\end{aligned}$$

Hence, the result follows. □

Note 9.4. In order to prove the following theorem we need to restrict the recombination vector $\boldsymbol{r}$ from Definition 9.1 to have entries from $\{-1, 0, 1\}$. This is not a problem by the construction given for multiplicative ISP, since it only contains entries from $\{0, 1\}$.

Theorem 9.2. *The* Add, Multiplication, *and* Open *protocol securely realize* $\mathcal{F}_{\texttt{MPC}}$ *(Fig. 9.1) for a static passive* $Q2$ *adversary.*

Proof. In order to prove the protocols secure in the UC-framework, or more precisely, to show that the protocols Add, Multiplication, and Open securely implement $\mathcal{F}_{\texttt{MPC}}$. That is, given an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$, then we need to construct a simulator S. The simulator provides $\mathcal{A}$ with a simulated view of the protocol, which is based on the information provided from the functionality $\mathcal{F}_{\texttt{MPC}}$. This game is called the *ideal process*. This is compared to the *real process*, where $\mathcal{Z}$ and $\mathcal{A}$ are interacting with a real instance of the protocol. In both processes, $\mathcal{Z}$ and $\mathcal{A}$ may communicate at any time. The goal is now to show that $\mathcal{Z}$ cannot distinguish the real from the ideal process.

The simulator proceeds as follows whenever required in the cases below, where B denotes the set of corrupted parties.

1. Upon receiving $(\mathsf{Input}, P_j, a, l)$ from the functionality, then the simulator chooses random vector $\boldsymbol{\rho} \in [-2^{l_0+k}..2^{l_0+k}]^e$, with the restriction that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = 0$, computes $\boldsymbol{s} = M\boldsymbol{\rho}$ and provides the adversary $\boldsymbol{s}_B$.

2. Upon receiving shares to all honest parties from the adversary, then follow the protocol. Using that the simulator has all honest shares, reconstruct the value to, say, a. Then send $(\mathsf{Input}, P_{\mathcal{A}}, a, l)$ to the functionality.

3. Upon receiving $(\mathsf{Add}, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$, update the simulated sharings of the honest parties accordingly.

4. Upon receiving $(\mathsf{Mult}, v, w, x, l_a, l_b)$, the simulator follows the protocol. Note, if the adversary does not know both multiplied values it will result in a secret sharing of 0, since all unknown values of the adversary are secret sharings of 0.

5. Upon receiving (Value, v, c) the simulator does one of the following things.

   - If the variable v refers to a secret shared value that is known by the adversary, then just follow the protocol.
   - Otherwise, do the following. The simulator modifies the unknown (to the adversary) shares of the honest parties by $cM\boldsymbol{\kappa}$, where $\boldsymbol{\kappa}$ is the sweeping vector for B. Note, that if the opened value is a linear combination of some known and some unknown to $\mathcal{A}$ secret shares, then the value c needs to be modified accordingly such that the opened value will be c.

First of all we need to argue that the Multiplication protocol is correct, that is, given input $\boldsymbol{a}$ and $\boldsymbol{b}$ of secret sharings of a and b, respectively, the protocol should output $\boldsymbol{c}$, which should be a secret sharing of $c = ab$. However, this follows by inspection of the protocol and since the used ISP is multiplicative.

Note, by the privacy of the LISS scheme $\mathcal{Z}$ cannot distinguish between the ideal and real process based on secret sharing of values (case 1).

Since we are in the passive case, the adversary will secret share a consistent share in case 2, and hence, the simulator can uniquely reconstruct this value and provide it to the functionality.

Note, that there is no interaction in case 3. Hence, there is nothing to argue in this case.

Case 4 is a bit more subtle to argue. First we introduce some notation. Let $\boldsymbol{a} = (a_1, \ldots, a_d)^T$ and $\boldsymbol{b} = (b_1, \ldots, b_d)^T$ be the share components of the two secret sharings to be multiplied by the protocol. That is, share components a_i and b_i are owned by party $P_{\psi(i)}$. Let $c_{(i,j)} = a_i b_j$, where $i, j \in \psi^{-1}(\iota)$ for some ι. Let $\boldsymbol{\rho} = \sum_{(i,j)} r_{(i,j)} \boldsymbol{\rho}_{(i,j)}$, where $\boldsymbol{\rho}_{(i,j)}$ is the distribution vector used in the re-sharing step of the protocol for the product of share components a_i and b_j, where $i, j \in \psi^{-1}(\iota)$ for some ι. If the result of the multiplication is c, then in the real process it would follow that $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = c$. Furthermore, note that $c_{(i,j)} = \langle \boldsymbol{\rho}_{(i,j)}, \boldsymbol{\varepsilon} \rangle$ for all $i, j \in \psi^{-1}(\iota)$ for some ι.

Let $\boldsymbol{\rho}^{(\text{ideal})}_{(i,j)}$ be the distribution vectors used in the ideal process. Then we have

$$\boldsymbol{\rho}^{(\text{ideal})} = \sum_{(i,j)} r_{(i,j)} \boldsymbol{\rho}^{(\text{ideal})}_{(i,j)},$$

and we assume that $\langle \boldsymbol{\rho}^{(\text{ideal})}, \boldsymbol{\varepsilon} \rangle = 0$, otherwise there is nothing to show. Furthermore, let $\langle \boldsymbol{\rho}^{(\text{ideal})}_{(i,j)}, \boldsymbol{\varepsilon} \rangle = c^{(\text{ideal})}_{(i,j)}$, i.e., $\sum_{(i,j)} r_{(i,j)} c^{(\text{ideal})}_{(i,j)} = 0$.

In the real process we have

$$\boldsymbol{\rho}^{(\text{real})} = \sum_{(i,j)} r_{(i,j)} \boldsymbol{\rho}^{(\text{real})}_{(i,j)}$$

and $\langle \boldsymbol{\rho}^{(\text{real})}, \boldsymbol{\varepsilon} \rangle = c$, where c is the product of the two shares multiplied. And $\langle \boldsymbol{\rho}^{(\text{real})}_{(i,j)}, \boldsymbol{\varepsilon} \rangle = c^{(\text{real})}_{(i,j)}$, i.e., $\sum_{(i,j)} r_{(i,j)} c^{(\text{real})}_{(i,j)} = c$.

By Lemma 9.4 it follows that the distribution of the final shares only depend upon the actual value of the shares. That is, in the ideal process, the value is 0 and the real process it is c.

By Lemma 9.3 there exists (i, j) owned by an honest party, with $r_{(i',j')} \neq 0$. Let $(\boldsymbol{\rho}')^{(\text{ideal})}_{(i',j')} = \boldsymbol{\rho}^{(\text{ideal})}_{(i',j')} + c\boldsymbol{\kappa} r_{(i',j')}$. Hence,

$$(\boldsymbol{\rho}')^{(\text{ideal})} = r_{(i',j')} \left((\boldsymbol{\rho}')^{(\text{ideal})}_{(i',j')} + c\boldsymbol{\kappa} r_{(i',j')} \right) + \sum_{(i,j) \neq (i',j')} r_{(i,j)} \boldsymbol{\rho}^{(\text{ideal})}_{(i,j)},$$

which fulfills $\langle (\boldsymbol{\rho}')^{(\text{ideal})}, \boldsymbol{\varepsilon} \rangle = c$.

Since $|c| < 2^l$ it follows by the privacy of LISS that the above modification can be done with only changing the statistical distribution at most negligible in the security parameter.

Hence, the statistical distance between the distributions of $\boldsymbol{\rho}^{(\text{real})}$ and $\boldsymbol{\rho}^{(\text{ideal})}$ is negligible close.

Note, that all $|c_{(i,j)}| < 2^l$ it follows by the privacy of LISS, that only negligible statistical information is leaked about the $c_{(i,j)}$'s in the re-sharing step.

In case 5 the value to be opened is the output from a multiplication, see the Open protocol.

That is, the value to be opened is the result of the Mult command. Let $\boldsymbol{c}^{(\text{real})}$ be the share components in the real process. Since we assume that all values that are to be opened are unknown to $\mathcal{A}$ and therefore also unknown

for the simulator. That is, the simulator will hold a set of share components $\boldsymbol{c}^{(\text{ideal})}$ such that it is a secret sharing of 0 and $\boldsymbol{c}_B^{(\text{ideal})}$ is given to $\mathcal{A}$, where B is the set of corrupted parties. We have already argued that $\boldsymbol{c}^{(\text{real})}$ and $\boldsymbol{c}^{(\text{ideal})}$ are indistinguishable for $\mathcal{Z}$ by the privacy of LISS. However, that also means, that the set of share components $\boldsymbol{c}^{(\text{ideal})}$ can be modified to $\boldsymbol{c}^{(\text{ideal})} + c\boldsymbol{\kappa}$, where $\boldsymbol{\kappa}$ is the sweeping vector for B, undetectable with all but negligible probability. That is, the opening of $\boldsymbol{c}^{(\text{ideal})} + c\boldsymbol{\kappa}$ can be done as in the real process with all but negligible probability, i.e., it is indistinguishable for $\mathcal{Z}$.

We have now argued all possible scenarios $\mathcal{Z}$ can use to distinguish the real from the ideal process cannot be used to distinguish the two processes, hence, the Add and the Multiplication protocols securely realize $\mathcal{F}_{\texttt{MPC}}$. □

Consider the following scenario. Let $\boldsymbol{a} = (a_1, \ldots, a_d)^T$ and $\boldsymbol{b} = (b_1, \ldots, b_d)^T$ be initial secret sharing over LISS for values $a \in [-2^{l_a}..2^{l_a}]$ and $b \in [-2^{l_b}..2^{l_b}]$. Using the above Multiplication-protocol will result in a secret sharing of $\boldsymbol{c} = (c_1, \ldots, c_d)^T$ of $c = ab \in [-2^{l_c}..2^{l_c}]$ for $l_c = l_a + l_b$. Note here, that the share components of $\boldsymbol{c}$ are of greater size than if it was an initial sharing of c with the same security parameter k. Hence, the shares of c are bigger than necessary.

The following protocol can be used to reduce the size of the share components. In order to do the reduction we use a protocol from [2] by Algesheimer et al. The protocol takes an additive sharing over some prime-field $\mathbb{Z}_p$ and converts it to an additive sharing over the integers $\mathbb{Z}$, where each share will be of size $l + k + 2$, where l is the bit length of the actual secret value and k is the security parameter. Note, that the resulting share sizes only depend on the l and k, and *not* of the field size of $\mathbb{Z}_p$.

We cannot use their protocol directly, since we do not start with an additive sharing over some prime field $\mathbb{Z}_p$. We denote their protocol by SQ2SI.

Protocol Reduce($\boldsymbol{c}$)

For $i = 1, \ldots, n$ party P_i has share $\boldsymbol{c}_{\{P_i\}}$. The size of the secret in the secret sharing $\boldsymbol{c}$ is l, and k is the security parameter.

1. Choose prime $p > 2^{l+k+\lceil \log(n) \rceil + 4}$.
2. For $i = 1, \ldots, d$ party $P_{\psi(i)}$ lets $\hat{c}_i = c_i \bmod p$.
3. For $i = 1, \ldots, d$ party $P_{\psi(i)}$ makes an additive sharing of $\hat{c}_i$, resulting in $a_{(j,1)}, \ldots, a_{(j,n)}$ such that $\hat{c}_i = \sum_i a_i \bmod p$.
   - For each $j = 1, \ldots, n$ party $P_{\psi(i)}$ sends $a_{(i,j)}$ privately to party P_j.
4. For $i = 1, \ldots n$ party P_i lets $\bar{c}_i = \sum_{j=1}^{d} \lambda_j a_{(j,i)}$, where $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_d)^T$ is a reconstruction vector.
5. Use SQ2SI on the additive sharing consisting of $\bar{\boldsymbol{c}} = (\bar{c}_1, \ldots, \bar{c}_n)^T$. This results in an additive sharing over the integers, we denote it $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_n)^T$, where party P_i has share $\tilde{c}_i$.
6. For $i = 1, \ldots, n$ party P_i secret share share $\tilde{c}_i$ over LISS, resulting in $(\tilde{c}_{(i,1)}, \ldots \tilde{c}_{(i,d)})^T$.

- For each $j = 1, \ldots, d$ party P_i sends $\tilde{c}_{(i,j)}$ privately to party $P_{\psi(j)}$.

7. For $i = 1, \ldots, d$ party $P_{\psi(i)}$ lets $c_i^{(\text{new})} = \sum_j \tilde{c}_{(j,i)}$ be his new share.

Remark 9.5. Consider the following scenario. Let P_T denote a trusted third party, which all parties involved in the protocol trust. Then if all parties send their shares to P_T, then P_T could re-share the value with appropriate share sizes. Denote this strategy by Ideal. If we compare the share component sizes from Reduce to Ideal, then they are at most $k + \log(n)$ bits larger in the Reduce protocol.

9.3.3 Share Sizes and Communication Complexity

Consider the following experiment. Let a and b be two integers from the interval $[-2^l..2^l]$. Choose $\boldsymbol{\rho}_a$ and $\boldsymbol{\rho}_b$ uniformly at random from the intervals $[-2^{l_0+k}..2^{l_0+k}]$, with the restriction that $\langle \boldsymbol{\rho}_a, \boldsymbol{\varepsilon} \rangle = a$ and $\langle \boldsymbol{\rho}_b, \boldsymbol{\varepsilon} \rangle = b$. Let $\boldsymbol{a} = M\boldsymbol{\rho}_a$ and $\boldsymbol{b} = M\boldsymbol{\rho}_b$. Use the Multiplication protocol on $\boldsymbol{a}$ and $\boldsymbol{b}$ as input, then it will result in $\boldsymbol{c} = M\boldsymbol{\rho}$ where $\boldsymbol{\rho} = \sum_{i,j} r_{(i,j)} \boldsymbol{\rho}_{(i,j)}$. By Note 9.3 we can simplify to $\boldsymbol{\rho} = \sum_{i=1}^{n} r_1 \boldsymbol{\rho}_i$. Where each $\boldsymbol{\rho}_i$ is chosen from $[-2^{l'_0+k}..2^{l'_0+k}]$, where $l' = 2(l_0 + k + m_{\max} + \lceil \log(e) \rceil)$. That is, the share components of $\boldsymbol{c}$ have size bounded by $l'_0 + k + m_{\max} + \lceil \log(e) \rceil + \lceil \log(n) \rceil$, which is $2(k + m_{\max} + 2\lceil \log(e) \rceil + \lceil \log(\kappa_{\max}) \rceil) + \lceil \log(n) \rceil$ bits greater than if $\boldsymbol{c}$ was an initial share.

In order to give a communication complexity it is necessary to consider a specific access structure. As a standard example we use a threshold-t access structure. In this case, the share component sizes of a secret bounded numerical by 2^l is $O(l + k + n)$ bits, were we use that the ISP constructed in [34] has $m_{\max} = O(n)$. Each share consists of $\log(n)$ share components, see [34].

If given two secret sharings $\boldsymbol{a}$ and $\boldsymbol{b}$ of integers a and b from the interval $[-2^l..2^l]$, respectively. Then using the Multiplication protocol on these sharings will result in that each party needs to send $O\left(n \log(n)(l + k + n)\right)$ bits, since each party needs to make one re-sharing (see Note 9.3). Let the communication complexity be defined to be all bits send during the protocol execution. Then the communication complexity of the Multiplication protocol is,

$$O\Big(n^2 \log(n)(l + k + n)\Big)$$

bits.

9.4 Multi-Party Computation for Active Adversary

In order to secure the multiplication protocol against an active adversary we will use the information theoretic secure commitment scheme from Chapter 8. A commitment scheme itself does not solve all the problems, we need some additional tools. First of all, we will need to be able to "transfer" a commitment. That is, if party P_i commits to a value v, then it should be able for him to transfer the commitment to party P_j, but in a way such that party P_j is ensured

that he receives the correct opening to the commitment and can prove otherwise if not.

To model this in the UC framework, we extend the functionality $\mathcal{F}_{\texttt{commit}}$ given in Chapter 8. First we simply extend the functionality to let each party involved in the computation be able to commit. We give the redefined $\mathcal{F}_{\texttt{commit}'}$ functionality in Figure 9.2. Note, that the only difference is, that each party can use the commit command, whereas in $\mathcal{F}_{\texttt{commit}}$ it was restricted to one party, the dealer. The $\mathcal{F}_{\texttt{commit}'}$ functionality can obviously be implemented in the $\mathcal{F}_{\texttt{commit}}$-hybrid model. That is, $\mathcal{F}_{\texttt{commit}'}$ can be realized by using the same protocols as for $\mathcal{F}_{\texttt{commit}}$.

Functionality $\mathcal{F}_{\texttt{commit}'}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{Input}, sid, a_1, \ldots, a_\nu, l)$ from party P_j:
   (a) Store (P_j, v_i, a_i, l) under a unique variable name v_i for all $i = 1, \ldots, \nu$.
   (b) Wait one round. Broadcast $(\mathsf{Input}, P_j, v_1, \ldots, v_\nu, l)$ to all parties and the adversary.
2. Upon receiving $(\mathsf{Add}, sid, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ from P_j:
   (a) Verify that there exists stored tuples $(P_j, v_1, a_1, l_1), \ldots, (P_j, v_\eta, a_\eta, l_\eta)$, otherwise abort.
   (b) Store the tuple $(P_j, x, \sum_{i=1}^{\eta} c_i a_i, \max(c_1 + l_1, \ldots, c_\eta + l_\eta) + \eta - 1)$ under unique variable name x.
   (c) Broadcast $(\mathsf{Add}, P_j, x, v_1, \ldots, v_\eta, c_1, \ldots, c_\eta)$ to all parties and the adversary.
3. Upon receiving (Open, sid, v) from party P_j:
   (a) Verify that there exists stored tuple (P_j, v, a, l), otherwise abort.
   (b) Verify that $|a| < 2^l$, if not then wait one round and broadcast $(\mathsf{Value}, P_j, v, \mathsf{Reject})$ and abort.
   (c) Wait one round. Broadcast $(\mathsf{Value}, P_j, v, a)$ to all parties and the adversary.

Figure 9.2: Functionality $\mathcal{F}_{\texttt{commit}'}$.

In the following subsection, the $\mathcal{F}_{\texttt{commit}'}$ functionality will be extended with a transfer command, which will transfer a stored commitment by party P_i to party P_j. Furthermore, we will extend the $\mathcal{F}_{\texttt{commit}'}$ functionality with a commitment multiplication feature, which can prove to the other parties that three given commitments, say of a, b and c, by a party P_i fulfill that $ab = c$.

9.4.1 Auxiliary protocols

In this section we let for notational convenience $[x]_i$ represent a LISS share done by party P_i. The LISS share can also be thought of as commitments, e.g., $[x]_i$ represent the commitment of a value x done by party P_i.

We need some auxiliary protocols to make the multiplication protocol secure against an active adversary.

- A *commitment transfer protocol* (CTP) allows party P_i to transfer a commitment to party P_j, i.e., to convert $[a]_i$ into $[a]_j$. It must be guaranteed that this protocol leaks no information to the adversary if P_i and P_j are honest throughout the protocol, but also that the new commitment contains the same value as the old, even if P_i and P_j are both corrupt.

- A *commitment sharing protocol* (CSP) allows party P_i to convert a committed value $[a]_i$ into a set of commitments to shares of a : $[a_1]_{\psi(1)}, \ldots,$ $[a_d]_{\psi(d)}$, where $(a_1, \ldots, a_d) = M\boldsymbol{\rho}$ for a random vector $\boldsymbol{\rho}$, with $\langle \boldsymbol{\rho}, \boldsymbol{\varepsilon} \rangle = a$, chosen by party P_i. This must hold even if P_i is corrupt, and must leak no information to the adversary if P_i is honest throughout the protocol.

- A *commitment multiplication protocol* (CMP) allows party P_i with committed values $[a]_i, [b]_i$ and $[c]_i$, to convince the other parties that $ab = c$. If P_i is corrupted, then the honest parties should accept the proof only if $ab = c$.

We start with the commitment transfer protocol. First note, that each of the involved parties have a set of parties that at some point accused them (see Chapter 8 for details). Let C_{P_i} denote the set of the party that transfers the commitment and let C_{P_j} denote the set of the party that receives the opening of the commitment.

Protocol CTP (Commitment Transfer Protocol)
The following protocol converts $[a]_i$ into $[a]_j$, where $a \in [-2^l..2^l]$.

1. P_i sends privately the distribution vector $\boldsymbol{\rho}$ determining a to P_j. If this information is not consistent, then P_j broadcasts (Accuse, P_i), and the protocol continues at step 6.
2. P_j commits to a (independently) using a new random distribution vector $\boldsymbol{\rho}'$, resulting in $[a]_j$. If the commitment is not accepted, the protocol continues at step 6.
3. Using linearity of commitments, P_j opens the difference $[a]_i - [a]_j$ to reveal 0, using the information from step 1 as if he created $[a]_i$ himself. That is, he broadcasts $\boldsymbol{\rho} - \boldsymbol{\rho}'$.
4. Each party computes his share of $[a]_i - [a]_j$ and compares it to what P_j broadcasted. If there is a disagreement, he broadcasts (Accuse, P_j) and is added to C_{P_j}.
5. If $C_{P_i} \cup C_{P_j}$ is in the adversary structure, the protocol ends. Otherwise do step 6.

6. If we arrive at this step, it is clear that at least one of P_i and P_j is corrupt, so P_i must then open $[a]_i$ in public, and we either disqualify P_i (if he fails) or continue with a default commitment to a assigned to P_j.

Functionality $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{CTP}, sid, P_j, v)$ from party P_i:
   (a) Verify that there exists stored tuple (P_i, v, a, l), otherwise abort.
   (b) Store tuple (P_j, v, a, l).
   (c) Broadcast $(\mathsf{CTP}, P_j, P_i, v)$ to all parties and the adversary and privately send $(\mathsf{CTP}, P_i, v, a)$ to P_j.

Figure 9.3: Functionality $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$.

The functionality in Fig. 9.3 is an extended version of $\mathcal{F}_{\mathtt{commit}'}$ given in Fig. 8.1. That is, it has the same functionality as $\mathcal{F}_{\mathtt{commit}'}$ (left out) with the additional command that captures the functionality of the CTP protocol.

Lemma 9.5. *The protocols* Commit, Open, *and* CTP *securely realize* $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$ *(Fig. 9.3) with an active adversary.*

Proof. The simulator acts as the simulator for $\mathcal{F}_{\mathtt{commit}'}$ with the following addition.

- There are four cases when using the CTP command in $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$.
  1. If the sender and receiver of the commitment are both corrupted, then just follow the protocol and send the $(\mathsf{CTP}, sid, P_{\mathcal{A}}, v)$ to the functionality on behalf of the sending corrupted party.
  2. If the sender is corrupted and the receiver is honest, then the simulator will receive all information of the commitment. Follow the protocol and send the $(\mathsf{CTP}, sid, P_i, v)$ to the functionality on behalf of the sending corrupted party.
  3. If the sender is honest and receiver is corrupted, then the simulator will receive the value a of the committed value. Modify the committed shares of the honest parties by $aM\boldsymbol{\kappa}$, and then follow the protocol, where $\boldsymbol{\kappa}$ is the sweeping vector for the corrupted parties.
  4. If the sender and receiver is honest, then just follow the protocol.

In case 1 the simulator will act on behalf of the honest parties, which obviously can be done indistinguishable from the real process. Hence, the only way $\mathcal{A}$ can distinguish the ideal from the real process based on this case is if it is possible

to change the committed value during the transfer, i.e., to transfer $[a]_i$ to $[a']_j$ for $a \neq a'$. However, since both the commitment $[a]$ and $[a']$ are accepted we know the honest parties have the values consistently shared among them. Furthermore, step 5 in the protocol ensures that the opening of the difference in step 3 $[a]_i - [a']_j$ is done as if the commitment was done by P_j. Hence, if accepted we are ensured by the commitment scheme that $a = a'$ as required.

In case 2 the simulator can obviously also act indistinguishable on behalf of the honest parties. Hence, the only way $\mathcal{A}$ can distinguish the two processes is if she can send a fallacious distribution vector in step 1 of the protocol. However, again, by the arguments above, the protocol ensures that this is not possible.

Case 3 is a bit more trickier. Since the committed value comes on behalf of an honest party from the functionality it is ensured that the value is in the right range, i.e. that the committed value y is bounded by the specified range $|y| \leq 2^l$. Hence, the privacy of LISS ensures that the modification of the honest shares cannot be detected with all but negligible probability.

Note that a corrupt party cannot transfer a commitment that is out of range to an honest party, since the honest party will complain in step 1 of the protocol.

In the last case (case 4) the protocol will be simulated on a commitment of zero. Furthermore, the distribution vector revealed in step 3 obviously has the right distribution, and hence, the simulation is indistinguishable from the real process.

Finally note, if a corrupted party $P_{\mathcal{A}}$ transfers a commitment $[a]_{\mathcal{A}}$ to an honest party P_i, i.e., to $[a]_i$. Then later $\mathcal{Z}$ asks P_i to transfer this honest party P_j. Then the simulation is done on the actual value a instead of 0, which is just like in the real process. Hence, indistinguishable. □

Protocol CSP (Commitment Sharing Protocol)

The protocol takes $[a]_i$ as input.

1. P_i commits to $[\rho_2]_i, \ldots, [\rho_e]_i$.
2. By the linearity of the commitment scheme compute $[a_1]_i, \ldots, [a_d]_i$ from $M\boldsymbol{\rho}$ using $\boldsymbol{\rho} = ([a]_i, [\rho_2]_i, \ldots, [\rho_e]_i)^T$ as distribution vector.
3. For $j = 1, \ldots, d$ P_i uses CTP to convert $[a_j]_i$ into $[a_j]_{\psi(j)}$.

The protocol only uses commands of the $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$ functionality. A security proof follows immediately in the $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$-hybrid model.

Protocol CMP (Commitment Multiplication Protocol)

The protocol takes commitments $[a]_i$, $[b]_i$, and $[c]_i$ as inputs. P_i claims that $ab = c$.

1. P_i chooses uniformly at random $\beta \in [-2^{l+2k}..2^{l+2k}]$, and commits to $[\beta]_i$ and $[\beta b]_i$.
2. The other parties jointly generate a random challenge $r \in [-2^k..2^k]$.
3. P_i opens the commitment $r[a]_i + [\beta]_i$ to reveal a value r_1. Then P_i opens commitment $r_1[b]_i - [\beta b]_i - r[c]_i$ to reveal 0.
4. If any of these openings fail, the proof is rejected, else it is accepted.

Functionality $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP},\mathsf{CMP}}$

The functionality proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{CMP}, sid, v_1, v_2, v_3)$ from party P_i:
   (a) Verify that there exists stored tuples (P_i, v_1, a, l_a), (P_i, v_2, b, l_b), and(P_i, v_3, c, l_c), otherwise abort.
   (b) Verify that $ab = c$, if not, then Broadcast $(\mathsf{CMP}, P_i, v_1, v_2, v_3, ab - c, \mathsf{Fail})$ and abort.
   (c) Broadcast $(\mathsf{CMP}, P_i, v_1, v_2, v_3)$ to all parties and the adversary.

Figure 9.4: Functionality $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP},\mathsf{CMP}}$.

The functionality in Fig. 9.4 is an extended version of $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$ given in Fig. 9.3. That is, it has the same functionality as $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$ (left out) with the additional command that captures the functionality of the CMP protocol.

Lemma 9.6. *The protocols* Commit, Open, CTP, *and* CMP *securely realize the functionality* $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP},\mathsf{CMP}}$ *(Fig. 9.4) with an active adversary.*

Proof. The simulator runs as the simulator for $\mathcal{F}_{\mathtt{commit}'}^{\mathsf{CTP}}$ with the following addition.

- If a corrupted party uses the CMP, just follow the protocol.
- If the simulator receives the tuple $(\mathsf{CMP}, P_i, v_1, v_2, v_3, d, \mathsf{Fail})$, then modify the shares of v_3 to secret share d instead of 0.
- If the simulator receives $(\mathsf{CMP}, P_i, v_1, v_2, v_3,)$ and the variables all refer to unknown values by simulator, then just follow the protocol. Otherwise, if v_3 is known and at least one of v_1 or v_2 are unknown, then the simulator should first modifies the shares, such that the protocol will be accepted.

First we argue, if a corrupted party uses CMP then he cannot get the protocol accepted, unless it is satisfied. However, if P_i is corrupt and $ab \neq c$ he can only reveal 0 in the last opening with probability less than $1/2^k$, which is negligible in the security parameter.

If an honest party fails to finish the protocol, then obviously the revealed distribution vector in the opening in step 3 is indistinguishable from the real process.

If it is an honest party using the protocol and it succeeds, then we need to argue that no information is leaked to the environment.

Note, that we neglect to argue the special cases where some values are known by the simulator. It follows by the privacy of LISS that the shares can be modified fit this matter.

Since it is an honest party using the protocol we can assume that a, b are bounded by some range 2^l. Note, that if the statistical distance between the distribution of β and the revealed value $r_1 = r[a]_i + [\beta]_i$ is negligible in the security parameter k, then obviously the indistinguishability follows. Define r_1 as *good* if r_1 is in the same range as β. Since β is chosen uniformly at random it follows that the probability that r_1 is not good is at most

$$\Pr(r_1 \text{ is not good}) = \frac{2^{l+k}}{2^{l+2k}} = \frac{1}{2^k},$$

since we can assume that $|a| \leq 2^l$. The statistical distance between β and r_1 is at most the probability that r_1 is not good. Hence, the indistinguishability follows. □

Now we are ready to make the protocols secure against an active adversary. First note, in the passive case inputs to the protocol are provided by secret sharings the values among the parties. In the active case, this is still the case, but the share components are now commitments owned by the corresponding party. Hence, if a party wants to give a as input he commits to a and runs the commitment sharing protocol (CSP) on the commitment. Hence, making a verifiable secret sharing.

Protocol Add$(a_1, \ldots, a_\nu, c_1, \ldots, c_\nu)$

The input of the protocol is shares $\boldsymbol{a}_1 = ([a_{1,1}]_{\psi(1)}, \ldots, [a_{1,d}]_{\psi(d)})^T, \ldots, \boldsymbol{a}_\nu = ([a_{\nu,1}]_{\psi(1)}, \ldots, [a_{\nu,d}]_{\psi(d)})^T$, of $a_1, \ldots, a_\nu$, respectively. The protocol should output a secret sharing of $c = \sum_{i=1}^{\nu} c_i a_i$.

1. For $j = 1, \ldots, d$ party $P_{\psi(j)}$ lets $[s_j]_{\psi(j)} = \sum_{i=1}^{\nu} c_i [a_{i,j}]_{\psi(j)}$.

Then $\boldsymbol{s} = ([s_1]_{\psi(1)}, \ldots, [s_d]_{\psi(d)})^T$ gives the desired secret sharing.

Note 9.5. Even though the Add protocol is done over commitments it is still non-interactive.

Protocol Multiplication(a, b)

The input of the protocol is commitments $[a_1]_{\psi(1)}, \ldots, [a_d]_{\psi(d)}$ and $[b_1]_{\psi(1)}, \ldots, [b_d]_{\psi(d)}$, where $a_1, \ldots, a_d$ and $b_1, \ldots, b_d$ are LISS shares that determine unique values a and b, respectively. The protocol outputs commitments $[c_1]_{\psi(1)}, \ldots, [c_d]_{\psi(d)}$, where $c_1, \ldots, c_d$ is a LISS share of the value $c = ab$.

1. For $\ell = 1, \ldots, n$, P_ℓ computes $a_i \cdot b_j = c_{(i,j)}$ for $i, j \in \psi^{-1}(\ell)$, commits to it, and performs CMP on inputs $[a_i]_\ell, [b_j]_\ell, [c_{(i,j)}]_\ell$.
2. *Re-sharing step:* P_ℓ performs CSP on $[c_{(i,j)}]_\ell$, resulting in the commitments $[c_{(i,j)_1}]_{\psi(1)}, \ldots, [c_{(i,j)_d}]_{\psi(d)}$.
3. *Recombination step:* For $\iota = 1, \ldots, d$, party $P_{\psi(\iota)}$ computes $c_\iota = \sum_{i,j} r_{(i,j)} c_{(i,j)_\iota}$, where $\boldsymbol{r} = (r_{(1,1)}, \ldots, r_{(d,d)})^T$ is the recombination vector of $\mathcal{M}$. All parties compute $[c_\iota]_{\psi(\iota)} = \sum_{i,j} r_{(i,j)} [c_{(i,j)_\iota}]_{\psi(\iota)} = [\sum_{i,j} r_{(i,j)} c_{(i,j)_\iota}]_{\psi(\iota)}$.

Remark 9.6. Note that the Commit-protocol ensures that the share components of the commitment are bounded as noted in Remark 8.1. Assume that this verification was omitted then obviously the adversary $\mathcal{A}$ could commit to a number that was greater than the bound defined by the bit-length l. This violates the privacy of the commitment which is not a problem since the privacy of $\mathcal{A}$ is not an issue. However, if the commitment is used as input in the Multiplication-protocol along with a commitment of an honest party. Then it is not obvious that the privacy of the Multiplication-protocol is not broken.

The following theorem follows directly by inspection of the protocol.

Theorem 9.3. *The* Add *and the* Multiplication *protocols securely realize* $\mathcal{F}_{\texttt{MPC}}$ *in the* $\mathcal{F}_{\texttt{commit}'}^{\mathsf{CTP},\mathsf{CMP}}$*-hybrid model with an active* $Q3$ *adversary.*

9.4.2 Communication Complexity

Again we consider the threshold-t access structure as an example. By communication complexity we consider all bits send during the success full protocol where all parties, including the corrupted parties, follow the protocol. Broadcasting m bits with an active $Q3$ adversary has communication complexity $O(mn^2)$, where n is the number of parties.

The communication complexity of the $\mathsf{Commit}(y_1, \ldots, y_\nu, l)$ protocol is,

$$O\bigg(n^3 \log(n)\Big(l + k + n + \log(\nu)\Big)\bigg)$$

bits, where we use that $\ell = k$. The Open protocol has communication complexity,

$$O(n^3 \log(n)(l + k + n))$$

bits. The CTP protocol has communication complexity,

$$O(n^3 \log(n)(l + k + n))$$

bits. The CSP protocol has communication complexity,

$$O(n^4 \log^2(n)(l + k + n))$$

bits. The CMP protocol has communication complexity,

$$O(n^3 \log(n)(l + k + n))$$

bits.

Each party has $\log(n)$ share components. Hence one would conclude that each party needs to re-share $O(\log^2(n))$ elements. Fortunately, by the proof of Lemma 9.1 only $O(\log(n))$ elements are necessary to be re-shared.

Note 9.6. For any given multiplicative ISP it is not necessarily given that only $O(\log(n))$ elements should be re-shared. However, then the construction in Lemma 9.1 can be applied to obtain the bound.

Hence, the Multiplication protocol has communication complexity,

$$O(n^5 \log^3(n)(l + k + n))$$

bits.

Chapter 10

Interval Proofs in the Information Theoretic Model

10.1 Introduction

Brickell et al. [16] solved the problem of convincing a party that a committed integer lies in a publically known interval. This problem has been useful in many scenarios, such as: electronic cash systems [23], group signatures [18], publically verifable secret sharings schemes [15, 72], among others. Boudot showed in [14] an elegant and non-interactive solution to the problem, where he bases the security on the strong RSA assumption [50] in the random oracle model.

In Chapter 5 we presented the tools to convince that a verifiable secret shared number lies in a public interval in a non-interactive way *without* the random oracle. The strategy was to verifiable secret share the integer x using an integer commitment scheme, and then provide a non-interactive distributed verifier proof that x is a square.

In this chapter we provide a protocol that proves that a committed integer is non-negative. This is done by using the information theoretic commitment scheme from Chapter 8 and the basic number theory result that for all $x \geq 0$ there exists $a, b, c, d \in \mathbb{Z}$ such that $x = a^2 + b^2 + c^2 + d^2$.

First consider the scenario by using Shamir shares over a finite field $\mathbb{F}$. That is, we think of a Shamir share as a commitment. Say, there are n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$. The dealer D chooses polynomials f_a, f_b, f_c and f_d of degree t, with $f_a(0) = a, f_b(0) = b, f_c(0) = c$ and $f_d(0) = d$. Then D chooses a degree $2t$ polynomial f_x such that $f_x(0) = x$. For $i = 1, \ldots, n$ D sends $(f_a(i), f_b(i), f_c(i), f_d(i), f_x(i))$ to party P_i. Finally, D broadcasts polynomial $h = f_x - f_a^2 - f_b^2 - f_c^2 - f_d^2$, which fulfills that $h(0) = 0$. Then each party P_i verifies that $h(i) = f_x(i) - f_a(i)^2 - f_b(i)^2 - f_c(i)^2 - f_d(i)^2$, if not, he complains.

Since every number in a finite field $\mathbb{F}$ can be written as a sum of four squares, the above protocol is rather useless. On the other hand, doing this over LISS is useful, since only non-negative integers can be written as the sum of four squares. Hence, by this primitive a dealer can prove that his secret shared value x belongs to a public known interval $[a..b]$ by proving that $x - a$ and $b - x$ are non-negative.

In this chapter we simply convert the above protocol to LISS. While the commitment from Chapter 8 ensures that the commitments are correct, care has to be taken when publishing the information equivalent to the polynomial h in LISS, since it might contain information about the values a, b, c, d and x.

Finally note, that given a commitment of x, then by the CSP from Chapter 9, this can be converted to a information theoretic verifiable secret sharing of x.

10.2 Preliminaries

In this chapter we let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ denote a strongly multiplicative ISP. In the following the distribution vectors $\boldsymbol{\rho}$ are of dimension e. Furthermore, the share vector $\boldsymbol{s} = M\boldsymbol{\rho}$ has d entries, and the ψ function maps each of the d rows to a party in $\mathcal{P} = \{P_1, \ldots, P_n\}$. Furthermore, given any vector $\boldsymbol{x}$, we let x_i denote the i-th entry in $\boldsymbol{x}$.

Then let $M^\diamond$ be defined such that

$$M^\diamond \boldsymbol{\rho}^{(x,y)} = M\boldsymbol{\rho}^{(x)} \diamond M\boldsymbol{\rho}^{(y)},$$

where $\boldsymbol{x} \diamond \boldsymbol{y}$ denotes the vector containing all entries of the form $x_i y_j$, where $\psi(i) = \psi(j)$ (as defined in Section 9.2). Furthermore, $\boldsymbol{\rho}^{(x,y)}$ denotes the vector containing all entries of the form $\rho_i^{(x)}\rho_j^{(y)}$. Note, that $\boldsymbol{\rho}^{(x,y)}$ has e^2 entries, but is uniquely defined from the $2e$ entries from $\boldsymbol{\rho}^{(x)}$ and $\boldsymbol{\rho}^{(y)}$. However, by the result from Proposition 9.1 the dimension of $\boldsymbol{\rho}^{(x,y)}$ can be reduced to $e \leq m = \sum_{i=1}^{n} d_i$, where d_i is the number of share components owned by party P_i, i.e., $d_i = |\psi^{-1}(P_i)|$. Note, if $\boldsymbol{s}^{(x,y)}$ is generated by $M^\diamond$, that is, if there exists $\boldsymbol{\rho}^{(x,y)}$ of dimension e^2 such that $\boldsymbol{s}^{(x,y)} = M^\diamond \boldsymbol{\rho}^{(x,y)}$, then by the construction in Proposition 9.1 the resulting matrix $M^{\diamond,\text{reduced}}$ with m columns, will also generate $\boldsymbol{s}^{(x,y)}$. To make the proofs more elegant, we do *not* use the reduced matrix, but use the un-reduced matrix $M^\diamond$ with e^2 columns.

In the following we let Commit and Open denote the protocols from Chapter 8. Note, that the Commit protocol secret shares one or more values over LISS, and if accepted, the values are *consistently shared* (Definition 8.1) among the honest parties. For each dealer D using the Commit and Open protocol there is associated a set C_D of all parties complaining. This set is contained throughout the protocol, and if at any point this set becomes qualified, then the dealer D is rejected.

For notational convenience we write $\mathsf{Commit}(x)$ when we commit to an integer x, where we omit randomness r and the bit-size l as arguments to the Commit protocol. We let $[x]$ denote a LISS sharing of x. That is, if we run $\mathsf{Commit}(x)$ it will result in $[x]$.

Let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ be the ISP used for the Commit protocol, then if an honest dealer D runs $\mathsf{Commit}(x, y)$ then there will be distribution vectors $\boldsymbol{\rho}^{(x)}$ and $\boldsymbol{\rho}^{(y)}$, such that $\boldsymbol{s}^{(x)} = M\boldsymbol{\rho}^{(x)}$ and $\boldsymbol{s}^{(y)} = M\boldsymbol{\rho}^{(y)}$ will be the sharing vectors for $[x]$ and $[y]$, respectively. When using the Open protocol on, say, $[x]$, then D will publish $\boldsymbol{\rho}^{(x)}$.

Let $[x][y]$ denote the *lazy multiplication* of the share vectors $\boldsymbol{s}^{(x)}$ and $\boldsymbol{s}^{(y)}$. That is, that the parties compute $\boldsymbol{s}^{(x,y)} = \boldsymbol{s}^{(x)} \diamond \boldsymbol{s}^{(y)}$, which is done locally

without communication. D can run $\mathsf{Open}([x][y])$ simply by publishing $\boldsymbol{\rho}^{(x,y)}$ such that $M^{\diamond}\boldsymbol{\rho}^{(x,y)} = \boldsymbol{s}^{(x,y)}$.

10.3 The UC Functionality

In this section we define the UC functionailty $\mathcal{F}_{\geq 0}$, which basically is a commitment scheme that only accepts input that are non-negative.

Functionality $\mathcal{F}_{\geq 0}$

The functionality proceeds as follows, running with the dealer D, the parties $P_1, \ldots, P_n$ and an adversary S.

1. Upon receiving $(\mathsf{Input}, sid, x, l)$ from some party D:
   (a) Store (v, x, l) under a unique variable name v.
   (b) Broadcast $(\mathsf{Input}, D, v, l)$ to all parties and the adversary.
   (c) Wait one round, if $x \geq 0$ then broadcast (Accept, D, v), otherwise (Reject, D, v).
2. Upon receiving (Open, sid, v) from D:
   (a) Verify that there exists stored tuple (v, x, l), otherwise abort.
   (b) Verify that $|x| < 2^l$, if not, broadcast $(\mathsf{Value}, v, \mathsf{Reject})$ and abort.
   (c) Broadcast (Value, v, x) to all parties and the adversary.

Figure 10.1: Functionality $\mathcal{F}_{\geq 0}$.

Obviously, the $\mathcal{F}_{\geq 0}$ functionality can be extended to include the linear features given in the commitment scheme in Chapter 8.

10.4 Non-negative Number Proofs

In this section we exploit that every non-negative integer is the sum of four squares, that is, for all $x \in \mathbb{Z}, x \geq 0$, it holds that there exists $a, b, c, d \in \mathbb{Z}$ such that $x = a^2 + b^2 + c^2 + d^2$. The following protocol is obtained by that observation and uses the protocols Commit and Open. Let C_D be the set of parties that complain in the Commit and Open protocols. Note, that that the protocols share the same set C_D.

Protocol $\mathsf{NonNegative}(x)$

Let $\mathcal{M} = (M, \psi, \boldsymbol{\varepsilon})$ be the ISP used. Let C_D denote the set of parties that complain in any of the subprotocols Commit and Open.

1. D finds $a, b, c, d \in \mathbb{Z}$ such that $x = a^2 + b^2 + c^2 + d^2$.
2. D runs $\mathsf{Commit}^{(M^{\diamond})}(x), \mathsf{Commit}^{(M)}(a, b, c, d)$ to obtain $[x]$, $[a]$, $[b]$, $[c]$, $[d]$.

3. D runs Open on $[x]-[a][a]-[b][b]-[c][c]-[d][d]$ to reveal 0, otherwise the protocol is rejected.
4. If C_D is qualified, then reject the proof.

Note 10.1. One obvious problem with the protocol is that x is secret shared over $M^\diamond$ and not M. We will come back to this issue in Section 10.5

If D is honest, then obviously only a forbidden subset of parties will complain and the protocol will be accepted. On the other hand, if D is corrupt, then the Commit protocol ensures, that the values a, b, c, d and x are consistently shared among the parties. The security of the Open protocol is a bit more tricky, since we open a linear combination of some products. First of all, we need the ISP for $M^\diamond$ to be $Q3$ in order for the Open protocol to be reliable. Then we need to ensure, that the published distribution vector $\boldsymbol{\rho}$ does not reveal any information about the any of the values. Note, that if it was only a linear combination of the secrets, then the only information about the secret would be in the first entry, which in this case is 0. However, since we do the lazy multiplications this is *not* the case anymore.

The goal is to prove that the revealed distribution vector $\boldsymbol{\rho}$ of the value 0 cannot be distinguished from a "real" random sharing of 0, that is, an initial sharing of 0 done by an honest party. The following proposition shows how this can be achieved.

Proposition 10.1. *Assume that $x \in [0..2^l]$ then if the distribution vectors $\boldsymbol{\rho}_a$, $\boldsymbol{\rho}_b$, $\boldsymbol{\rho}_c$, $\boldsymbol{\rho}_d \in [-2^{l_0+k}..2^{l_0+k}]^e$ for the values a, b, c, d respectively, and $\boldsymbol{\rho}_x \in [-2^{2l_0+2k+\kappa}..2^{2l_0+2k+\kappa}]^{e'}$, then the statistically distance between the information revealed in step 3 and a real zero sharing is negligible in κ.*

Proof. The distribution vectors $\boldsymbol{\rho}_a, \boldsymbol{\rho}_b, \boldsymbol{\rho}_c$, and $\boldsymbol{\rho}_d$ are chosen from a space of dimension e, while the distribution vector $\boldsymbol{\rho}_x$ is chosen from a space of dimension e'. The reason of the different dimensions, is that the resulting sharing of the product is over a different threshold, and hence a different ISP. By using that the dimension $e' = e^2$, we get an easy notation, but e' can be optimized to be smaller than the number of resulting share components. This all follows from how the share components of $M\boldsymbol{\rho}_a \diamond M\boldsymbol{\rho}_a$ are computed. Note that for a party that owns row i and j the resulting share component $s_{i,j}$ can be computed as follows.

$$\begin{aligned} s_{i,j} &= s_i s_j = \sum_{\iota=1}^{e} m_{i,\iota}\rho_\iota \sum_{\iota=1}^{e} m_{j,\iota}\rho_\iota \\ &= m_{i,1}m_{j,1}\rho_1^2 + m_{i,1}m_{j,2}\rho_1\rho_2 + \cdots + m_{i,e}m_{j,e-1}\rho_e\rho_{e-1} + m_{i,e}m_{j,e}\rho_e^2, \end{aligned}$$

where s_i and s_j are the share components of $M\boldsymbol{\rho_a}$ and $M = [m_{i,j}]$ are the entries of M. By using the trivial construction, it is easy to see that the resulting distribution vector $\boldsymbol{\rho}_{a^2}$ has e^2 entries.

A real zero sharing is a sharing where the distribution vector $\boldsymbol{\rho}_0$ is chosen at random from $[-2^{2l_0+2k+\kappa}..2^{2l_0+2k+\kappa}]^{e'}$ with the restriction that first entry is 0.

The opened vector $\boldsymbol{\rho} = \boldsymbol{\rho}_x - \boldsymbol{\rho}_{a^2} - \boldsymbol{\rho}_{b^2} - \boldsymbol{\rho}_{c^2} - \boldsymbol{\rho}_{d^2}$, where vector $\boldsymbol{\rho}_{a^2}$ is defined by $\boldsymbol{\rho}_{a^2} = (\rho_1^2, \rho_1\rho_2, \ldots, \rho_e\rho_{e-1}, \rho_e^2)$. Likewise for $\boldsymbol{\rho}_{b^2}$, $\boldsymbol{\rho}_{c^2}$, and $\boldsymbol{\rho}_{d^2}$

We define the distribution vector $\boldsymbol{\rho}$ to be *good* if each entry is in the range $[-2^{2l_0+2k+\kappa}..2^{2l_0+2k+\kappa}]$. Note, that the first entry of $\boldsymbol{\rho}$ is zero as required. The probability that an entry is out of the range, is at most $4(2^{2l_0+2k}/2^{2l_0+2k+\kappa})$. Since the statistical distance is at most twice the probability that $\boldsymbol{\rho}$ is not good and using the union bound to estimate that probability that a single entry is out of range, we get the distance to be at most

$$8\frac{(e-1)2^{2l_0+2k}}{2^{2l_0+2k+\kappa}},$$

which is negligible in κ and concludes the proof. □

The following theorem follows immediately by the Commit and Open protocol and the above proposition.

Theorem 10.1. *If the ISP for $M^\diamond$ is Q3, then the* NonNegative *protocol securely realizes $\mathcal{F}_{\geq 0}$ in the information theoretic model.*

10.5 Further Considerations

As pointed out in Note 10.1 the resulting commitment of x will be secret shared over ISP $M^\diamond$ and not M which would be more useful.

Say, x was committed by using M instead. Then this commitment could be used as input in the CSP protocol, hence, it would provide input to the MPC protocols defined in Chapter 9. In this way, we get a short and easy proof, that an input to the MPC computations is non-negative, which can be used to prove that an input is contained in some given public interval. This all comes in handy, but leads us back to the problem pointed out in Note 10.1, that x is secret shared over ISP $M^\diamond$.

One way to avoid the problem is to initially secret share x over RISS (see Chapter 5), then locally convert x to a LISS share over M and $M^\diamond$. This can be done by Theorem 5.4, if the initial RISS share is done over an access structure $\Gamma^{\mathcal{R}}$ for which $\Gamma, \Gamma^\diamond \subseteq \Gamma^{\mathcal{R}}$, where Γ is the access structure for M and $\Gamma^\diamond$ is the access structure for $M^\diamond$.

Part IV

Beyond this Thesis

Chapter 11

Future Work

11.1 Conversion

Let $[a]$ represent a LISS sharing of the value a, and let $[a]_p$ denote a linear secret sharing (LSS) of $a \in \mathbb{Z}_p$. Given $[a]$, then one great advantage of LISS is, that $[a]$ can be locally converted without interaction to $[a \bmod p]_p$ for any p (not necessarily a prime). This has shown great advantages, since some tasks are more efficient in LISS, while others are more efficient over a prime field.

Take the example pointed out in Chapter 6. There the dealers secret share their inputs over LISS, use the protocol from Chapter 5 to prove the input is non-negative. Then the parties involved in the computations locally convert the LISS shares to LSS shares over any prime field of choice to proceed with the MPC computations, which is more efficient over a LSS scheme than over the LISS scheme.

However, consider the following scenario. Say, given a LISS share $[a]$, the parties can locally convert $[a \bmod 2]_2$ and reveal the shares of $[a \bmod 2]_2$ to obtain the least significant bit of a.

This was done by revealing only one value. We stress here, that it is *not* known how to do the same task as efficient over a prime field $\mathbb{Z}_p$ with $p > 2$.

Now consider the following, assume that the conversion $[a]_p$ to $[a]$ could be done efficiently. First of all, let us review a result by Nishide and Ohta [75]. They observe, that $[a < p/2]_p$ can be computed by extracting the least significant bit of $[2a \bmod p]_p = [2a]_p$. Furthermore, they observe that by computing $(a < p/2), (b < p/2)$ and $(a - b < p/2)$ they can achieve $(a < b)$, which is represented in Table 11.1.

Hence, if we could efficiently convert from a LSS scheme to the LISS scheme, then we could efficiently do comparison on unbounded values secret shared over a prime field.

Unfortunately, we do *not* know how to efficiently convert from a LSS scheme to the LISS scheme. However, the above observation indicates, that this conversion probably has a cost that is at least in the same magnitude of a general comparison over a LSS scheme.

$(a < p/2)$	$(b < p/2)$	$(a - b < p/2)$	$(a < b)$
1	0	$*$	1
0	1	$*$	0
0	0	0	1
0	0	1	0
1	1	0	1
1	1	1	0

Table 11.1: Truth table for $(a < b)$.

11.2 Further Research

When starting a new branch of research, like the LISS scheme, there are obviously many paths to search for results. As noted in Section 11.1, one nice feature of the LISS scheme we did *not* exploit, is that the least significant bit can be efficiently extracted from a LISS share. As also noted, to extract the least significant bit from a share over a finite field has some great benefits, but is unfortunately not easy to do. Therefore it may be possible that this feature could be exploited over the LISS scheme in some context where it would be beneficial.

In this thesis we have exploited that each non-negative integer can be written as the sum of four squares. This is a property that is useless over a prime field $\mathbb{Z}_p$, since all numbers $x \in \mathbb{Z}_p$ are the sum of four squares.

Another feature of the integers we used in this thesis is that an integer $x \in \mathbb{Z}$ can be interpreted as a number $x \bmod N$, for any N. This was exploited in the distributed exponentiation protocol. That is, we exploit that secret sharing over the integers in some sense contains more information, than secret sharing over a finite field (or similar).

In order to draw benefit from the LISS scheme, one has to look for properties of the integers that a prime field does not have. The example in Section 11.1 shows, combining LISS with LSS is a possibility of finding new useful protocols.

Finally note, that many protocols "translate" directly from LSS schemes to the LISS scheme. However, since the LISS scheme most often has bigger shares compared to a LSS scheme, the "translation" does not benefit anything.

Bibliography

[1] M. Abe, R. Cramer, and S. Fehr. Non-interactive distributed-verifier proofs and proving relations among commitments. In Y. Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 206–223. Springer, 2002.

[2] J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Yung [91], pages 417–432.

[3] M. Backes, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional reactive simulatability. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2006.

[4] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of dolev-yao-style xor. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.

[5] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, pages 201–209, 1989.

[6] F. Beauregard. *Linear Algebra. Third Edition.* Addison-Wesley Publishing Company, Inc., 1995.

[7] D. Beaver. Minimal-latency secure function evaluation. In *EUROCRYPT*, pages 335–350, 2000.

[8] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[10] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer, 1988.

[11] P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A practical implementation of secure auctions based on multiparty integer computation. In G. D. Crescenzo and A. D. Rubin, editors, *Financial Cryptography*, volume 4107 of *Lecture Notes in Computer Science*, pages 142–147. Springer, 2006.

[12] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Cachin and Camenisch [17], pages 223–238.

[13] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[14] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[15] F. Boudot and J. Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In V. Varadharajan and Y. Mu, editors, *ICICS*, volume 1726 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 1999.

[16] E. F. Brickell, D. Chaum, I. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret. In C. Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 156–166. Springer, 1987.

[17] C. Cachin and J. Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[18] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In Wiener [88], pages 413–430.

[19] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[20] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multiparty computation. In *STOC*, pages 639–648, 1996.

[21] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[22] R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. In Cachin and Camenisch [17], pages 207–222.

[23] A. H. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. In *EUROCRYPT*, pages 561–575, 1998.

[24] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19. ACM, 1988.

[25] B. Chor, M. Geréb-Graus, and E. Kushilevitz. Private computations over the integers. *SIAM J. Comput.*, 24(2):376–386, 1995.

[26] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395. IEEE, 1985.

[27] B. Chor and E. Kushilevitz. Secret sharing over infinite domains. *J. Cryptology*, 6(2):87–95, 1993.

[28] R. Cramer and I. Damgård. Secret-key zero-knowlegde and non-interactive verifiable exponentiation. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2004.

[29] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Kilian [67], pages 342–362.

[30] R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[31] R. Cramer, V. Daza, I. Gracia, J. J. Urroz, G. Leander, J. Martí-Farré, and C. Padró. On codes, matroids, and secure multiparty computation from linear secret-sharing schemes. *IEEE Transactions on Information Theory*, 54(6):2644–2657, 2008.

[32] R. Cramer and S. Fehr. Optimal black-box secret sharing over arbitrary abelian groups. In Yung [91], pages 272–287.

[33] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer, 2003.

[34] R. Cramer, S. Fehr, and M. Stam. Black-box secret sharing from primitive sets in algebraic number fields. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 2005.

[35] I. Damgård and K. Dupont. Efficient threshold rsa signatures with general moduli and no extra assumptions. In S. Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 346–361. Springer, 2005.

[36] I. Damgård, N. Fazio, and A. Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In Halevi and Rabin [59], pages 41–59.

[37] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Halevi and Rabin [59], pages 285–304.

[38] I. Damgård, D. Hofheinz, E. Kiltz, and R. Thorbek. Public-key encryption with non-interactive opening. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2008.

[39] I. Damgård and M. Koprowski. Practical threshold rsa signatures without a trusted dealer. In B. Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2001.

[40] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[41] I. Damgård and R. Thorbek. Linear integer secret sharing and distributed exponentiation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2006.

[42] I. Damgård and R. Thorbek. Non-interactive proofs for integer multiplication. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.

[43] I. Damgård and R. Thorbek. Efficient conversion of secret-shared values between different fields. Cryptology ePrint Archive, Report 2008/211, 2008.

[44] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In Halevi and Rabin [59], pages 360–379.

[45] Y. Desmedt and Y. Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discrete Math.*, 7(4):667–679, 1994.

[46] S. Fehr. Efficient construction of dual msp. In *Manuscript*, 1999.

[47] M. Fitzi, M. Hirt, and U. M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Krawczyk [68], pages 121–136.

[48] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *FOCS*, pages 384–393, 1997.

[49] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.

[50] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Jr. [65], pages 16–30.

[51] E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *EUROCRYPT*, pages 32–46, 1998.

[52] A. Gál. Combinatorial methods in boolean function complexity. In *PhD thesis*. University of Chicago, 1995.

[53] D. Galindo. Breaking and repairing damgard et al. public key encryption scheme with non-interactive opening. *Topics in Cryptology - CT-RSA 2009: The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009*, In LNCS 5473:389–398, 2009.

[54] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[55] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[56] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of rsa functions. *J. Cryptology*, 13(2):273–300, 2000.

[57] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[58] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity for all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[59] S. Halevi and T. Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006.

[60] L. L. Helms. *Introduction to Probability Theory*. Freeman, 1997.

[61] M. Hirt and U. M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC*, pages 25–34, 1997.

[62] S. Hoory, A. Magen, and T. Pitassi. Monotone circuits for the majority function. In J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick, editors, *APPROX-RANDOM*, volume 4110 of *Lecture Notes in Computer Science*, pages 410–425. Springer, 2006.

[63] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[64] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structures. In *Proc. IEEE Global Telecommunication Conf.*, pages 99–102, 1987.

[65] B. S. K. Jr., editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997.

[66] M. Karchmer and A. Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

[67] J. Kilian, editor. *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*. Springer, 2005.

[68] H. Krawczyk, editor. *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[69] S. Lang. *Algebra. Third Edition.* Addison-Wesley Publishing Company, Inc., 1993.

[70] Y. Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403. IEEE Computer Society, 2003.

[71] MacLane and Birkoff. *Algebra. Second Edition.* Macmillan Publishing Co., Inc., 1979.

[72] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 1998.

[73] J. B. Nielsen. On protocol security in the cryptographic model. In *PhD thesis*. Aarhus University, 2003.

[74] V. Nikov, S. Nikova, and B. Preneel. On the size of monotone span programs. In C. Blundo and S. Cimato, editors, *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 249–262. Springer, 2004.

[75] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In T. Okamoto and X. Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2007.

[76] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *PODC*, pages 51–59, 1991.

[77] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[78] T. Rabin. A simplified approach to threshold and proactive rsa. In Krawczyk [68], pages 89–104.

[79] J. Radhakrishnan. Better lower bounds for monotone threshold formulas. *J. Comput. Syst. Sci.*, 54(2):221–226, 1997.

[80] A. D. Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *STOC*, pages 522–533, 1994.

[81] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[82] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[83] V. Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.

[84] R. Thorbek. Proactive linear integer secret sharing. Cryptology ePrint Archive, Report 2009/183, 2009.

[85] T. Toft. Primitives and applications for multi-party computation. In *PhD thesis*. Aarhus University, 2007.

[86] L. G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984.

[87] B. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.

[88] M. J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

[89] A. C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, pages 160–164, 1982.

[90] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE, 1986.

[91] M. Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

Appendix A

One-Time Signature

A signature scheme is a tuple $\mathsf{OTS} = (\mathsf{SGen}, \mathsf{SSign}, \mathsf{SVer})$ of PPT algorithms such that:

- The randomized key generation algorithm SGen takes as input a security parameter 1^k and outputs a verification key vk and a signing key $sigk$. We write $(sigk, vk) \leftarrow_R \mathsf{SGen}(1^k)$.
- The randomized signing algorithm SSign takes as input a signing key $sigk$ and a message $m \in \{0,1\}^*$ and outputs a signature σ. We write $\sigma \leftarrow_R \mathsf{SSign}_{sigk}(m)$.
- The deterministic verification algorithm SVer takes as input a verification key vk and a signature σ. It returns `accept` or `reject`. We write $\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{SVer}_{vk}(m, \sigma)$.

For an adversary F, consider the following game:

1. $\mathsf{SGen}(1^k)$ outputs $(sigk, vk)$. Adversary F is given 1^k and vk.
2. The adversary may make one query m to a signing oracle $\mathsf{SSign}_{sigk}(\cdot)$ which results in the signature σ.
3. Finally, F outputs a message m^* and a signature σ^*.

Denote F's advantage by

$$\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(k) := \Pr[\mathsf{SVer}_{vk}(m^*, \sigma^*) = \texttt{accept} \wedge (m, \sigma) \neq (m^*, \sigma^*)].$$

OTS is called strongly unforgeable against one-time attacks (SUF-OT secure) if for all adversaries F $\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(\cdot)$ is negligible.

Index

M_A, 21
R-module, 17
Δ, 15
Δ^+, 16
Γ, 15
Γ^-, 16
$\boldsymbol{x}_A$, 21
$\kappa_{\max}$, 21
σ-algebra, 17
1, 61

access structure, 15
    Q2, 16
    Q3, 16
adversary structure, 15
algebra, 17
average share size, 21

binding, 86
bit complexity, 72
Boolean formula, 23

canonic ISP, 61
canonic span program, 61
commitment, 85
commitment scheme, 48
commits, 85
committed, 85
committer, 85
comparison protocol, 72
conversion problem, 77
corresponding access structure, 15
corresponding adversary structure, 15

distribution vector, 22

expansion rate, 21

forbidden, 20

hiding, 86
homorphic commitment scheme, 48

inner product, 21
Integer Span Program, 21
    size, 21
interval proof problem, 72
ISP, 21
ISP for Γ, 21

labeled, 21
labeled matrix, 21
lazy multiplication, 114
linear integer secret sharing, 20
Linear Integer Secret Sharing, 19
linear system
    consistent, 64
    Gauss-Jordan method, 64
LISS, 20
locally convertible, 61

maximal forbidden set, 16
minimal qualified set, 16
monotone access structure, 15
monotone adversary structure, 15
monotone formula, 23
MPC, 71, 77, 93
multi-party computation, 71, 77, 93

owned, 21, 25
owns, 23

PRF, 63
probability space, 17
pseudorandom function, 63

Q2, 16
Q3, 16
qualified, 20

random variable, 18
reconstruction vector, 21

Replicated Integer Secret-Sharing, 60
RISS, 60

share, 22
share completion property:, 35
share component, 22
share size, 21
share vector, 22
share element, 79
size of $\mathcal{M}$, 21
span program, 21
statistical distance, 18
sweeping vector, 21

target vector, 21
threshold access structure, 16

Union bound, 18

verifiable secret sharing, 54
VSS, 54