Design of advanced primitives for secure multiparty computation : special shuffles and integer comparison

Design of Advanced Primitives for Secure Multiparty Computation: Special Shuffles and Integer Comparison

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Design of advanced primitives for secure multiparty computation:
special shuffles and integer comparison / door José Antonio Villegas Bautista. –
Eindhoven : Technische Universiteit Eindhoven, 2010. –
Proefschrift. – ISBN 978-90-386-2224-8
NUR 919
Subject headings : cryptology, secure computation, cryptographic protocols,
Fourier transforms, fast Fourier transforms, algorithms

Printed by Printservice TU/e.
Cover: Love game. Design by Paul Verspaget.

This research was financially supported by the Dutch Technology Foundation STW under project number 6680.

Design of Advanced Primitives for Secure Multiparty Computation: Special Shuffles and Integer Comparison

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 29 april 2010 om 16.00 uur

door

José Antonio Villegas Bautista

geboren te Salta, Argentinië

Dit proefscrift is goedgekeurd door de promotor:

prof.dr.ir. H.C.A. van Tilborg

Copromotor:
dr.ir. L.A.M. Schoenmakers

A mis queridos padres...

To my dear parents...

Contents

Chapter 1

Introduction

Secure multiparty computation is a well-known problem in modern cryptography. A set of mutually distrusting parties need to perform a joint computation, where each party contributes some private information as input, and these inputs should remain hidden as much as possible throughout the computation. This should be valid even in the presence of an *adversary* who may *corrupt* some of the parties, meaning that the adversary sees all their internal data and may make them behave arbitrarily. A large body of research initiated in the early 1980s has shown that secure multiparty computation is feasible for any computable function [Yao82, Yao86, GMW87, BGW88, CCD88]. Lots of improvements have been achieved since.

The design of secure multiparty protocols is very complex, since many aspects must be taken into account. One can have protocols withstanding adversaries with different capabilities. A *passive* adversary follows the protocol specification but records all information it has collected during the run of a protocol. In contrast, an *active* adversary behaves arbitrarily, possibly aborting the protocol prematurely. An adversary is *static* if the set of corrupted parties is decided at the onset and fixed throughout the protocol execution, or the adversary is *adaptive* if parties are corrupted on the fly as the computation proceeds.

Another distinction between multiparty protocols is the model for communication. In the *cryptographic* model, the adversary may see all the information exchanged by the parties. Security in this case can be only guaranteed under a computational assumption. The *information-theoretic* model assumes a private channel between every pair of parties. Security is possible in an unconditional manner, without assuming any bound on the computational power of the adversary. Another important aspect is the environment in which a protocol is executed. A protocol may be analyzed in a *stand-alone* setting, or in a more general setting, allowing for concurrent and possibly interleaved executions; see, e.g., [Can01, PS04, Can05, Küs06, CDPW07].

Efficiency is an important aspect of the design of secure multiparty protocols. There are three widely accepted performance measures for protocols, usually analyzed as a function of the size of the inputs. The *computational* complexity is the number of elementary computing steps needed to execute the protocol. The *communication* complexity gives the number of bits transmitted between the parties. The *round* complexity measures the number of messages exchanged by the parties running the protocol. Often, there are trade-offs between these complexity measures, which can be used to achieve a good balance in practical situations.

This thesis focuses on designing efficient protocols for two particular cryptographic

primitives. The first primitive concerns generalizations of *verifiable shuffles*, allowing for ways to restrict the permutations applied to a special subset of permutations. The second primitive is *integer comparison*. Both of these primitives are of general importance in the design of protocols for secure multiparty computation. For our purposes it suffices to consider the basic setting of a static, active, computationally-bounded adversary in a stand-alone setting. The basic setting allows for a relatively simple and concise presentation, capturing the essence of the novel techniques and approaches used in our solutions. Using by-now standard techniques and set-up assumptions our solutions should carry over to stronger security models.

In the following, we give a brief description of the two primitives studied in the thesis. We include some of the main applications, describe relations to other problems and give further motivation for studying these primitives.

1.1 Special Shuffles of Homomorphic Encryptions

A shuffle is a rearrangement of a list of encrypted messages which produces a fresh list of encrypted messages such that the multiset of plaintexts of both lists is identical. Put in other words, there exists a permutation linking the plaintexts of both lists of encryptions. The crucial requirement to apply shuffles in cryptographic protocols is that the applied permutation is kept secret.

Shuffles of homomorphic encryptions is a simple but very powerful primitive. This is accomplished by permuting and "re-blinding" the list of encryptions. Verifiability of a shuffle is achieved via a zero-knowledge proof of knowledge. One of the main application areas of shuffles is the construction of mix-networks. A mix-network [Cha81] consists of a cascade of shufflers which one after the other randomly shuffle the list of encryptions received from the previous shuffler. The result is that the input-output lists of plaintexts are permuted and if at least one of the shufflers is honest, the end-to-end permutation is random and unknown.

In this thesis we study a related primitive. Namely, we describe zero-knowledge proofs of shuffles where a cyclic rotation is applied instead of an arbitrary permutation. This kind of shuffles were first introduced by Reiter and Wang [RW04] together with applications in the context of mix-networks. They argued that a shuffler is deterred from revealing information if there is only a limited number of allowed permutations for a shuffle. In the case of rotations, revealing any input-output correspondence of the permutation applied completely reveals the permutation used in the shuffle.

We point out many other applications of rotations in cryptographic protocols. In fact, we note that rotations are a fundamental primitive for the design of secure protocols. For instance, rotations are used to conceal sensitive information in voting protocols, integer comparison solutions, and in some general approaches to secure function evaluation. Concretely, in scrambled circuit approaches, like that of Yao [Yao86] or Jakobsson and Juels [JJ00], the active row of the truth table selected during the evaluation of a scrambled gate must be hidden. This may be achieved by applying a rotation of the rows of the scrambled gate.

We use two interesting approaches to rotation. On the one hand, using properties of the Discrete Fourier Transform (DFT) we give a Σ-protocol for showing correctness of a rotation. The use of the DFT imposes some mild restrictions on the system parameters. We believe that this is the first time that the DFT is used as the core tool of a zero-

knowledge protocol. On the other hand, we use a completely different approach to prove a rotation. We present a zero-knowledge proof for which we show the witness-extended emulation property. This protocol works with virtually *any* homomorphic cryptosystem and does not put any constraints in the parameters. Our zero-knowledge protocols have roughly the same complexity as the most efficient ones for general shuffles (e.g., [Gro03]) while the only previously existing solution by Reiter and Wang [RW04] uses *four* invocations of a general shuffle.

Reiter and Wang [RW04] also introduced the more general notion of a k-fragile set of permutations where revealing any k input-output correspondences of a permutation identifies the permutation completely within the k-fragile set. That way, a shuffler who applied a k-fragile permutation may reveal up to $k-1$ input-output mappings of the permutation before the permutation itself gets completely exposed. A rotation is clearly 1-fragile.

In this thesis, we present the first zero-knowledge proofs of knowledge for particular cases of k-fragile permutations, with $k > 1$. We show how to prove that a shuffler applied an affine transformation (2-fragile) and a Möbius transformation (3-fragile). We complement this study by pointing out the equivalence with the well-known concept of k-transitive sets of permutations. In fact, this enables us to give an overview of the (non-)existence results for multiply fragile permutations.

1.2 Integer comparison

The basic instance of integer comparison was introduced by Yao [Yao82] as the *millionaires' problem*: two millionaires want to compare their net worths and know who is richer without revealing anything else. Secure integer comparison refers to any problem in which two integers must be compared. The essential requirement is that no information is leaked about the two integers and possibly the result of the comparison.

These problems represent a fundamental primitive in secure computation protocols and thereby they have received much attention in the literature. Applications that are based on this primitive include electronic auctions [NPS99], secure data mining [LP00] and secure linear optimization [Tof09c] among many other problems.

In this thesis, we present integer comparison protocols within the framework for secure multiparty computation based on threshold homomorphic cryptosystems by Cramer *et al.* [CDN01]. Our solutions assume that the inputs x and y are given as encrypted bits of their binary representation and the output is an encryption of the bit deciding whether $x > y$. The generality of our solutions enable the use of our protocols in numerous applications.

The first type of solutions involves the evaluation of an arithmetic circuit composed of elementary gates as in [CDN01]. Since the intermediate multiplications of the circuit are performed on encrypted bits, one can apply conditional gates from [ST04] and thus it can be based on threshold homomorphic ElGamal encryptions. Furthermore, we note that the circuit can be used to get unconditional security if encryptions are replaced by sharings as in [DFK$^+$06]. The main achievements of this circuit are both low computational complexity and low round complexity.

The second type of solutions uses a more intricate approach. We present protocols that only require a constant number of rounds assuming a fixed number of parties. The computational complexity compares favorably with other existing solutions. In fact, our

solutions outperform any other protocol in a two-party setting in all the complexity measures. The proof of security of these protocols is also interesting in its own right. Namely, we follow ideas from [ST06] in which a successful attacker of the protocol is converted into an attacker of the semantic security of the underlying cryptosystem. In this way, we show the integration of our solution in the general frameworks of [CDN01, ST04] mentioned above. We give a complete description for the two-party setting.

We discuss different variations of integer comparison which may be useful in certain applications. For instance, we analyze how to get public output (instead of encrypted output) and how this affects the performance. We also describe the connection of integer comparison with other related problems. For example, one can show an equivalence between greater-than comparison and computing the least-significant bit of an integer. We believe that shedding light on these connections may be of help in finding more efficient solutions.

1.3 Roadmap of this Thesis

Below we give an overview of the structure and results of the thesis.

Chapter 2: Preliminaries

This chapter describes basic cryptographic primitives and building blocks. At the same time, we present the notation that will be used throughout the rest of the thesis.

Chapter 3: Verifiable Rotations using the Discrete Fourier Transform

In this chapter, we present a protocol for rotation using the Discrete Fourier Transform (DFT). The obtained zero-knowledge protocol is a Σ-protocol and is as efficient as the most efficient protocol for general shuffles. The application of the DFT and its inverse using encrypted values represents a bottleneck in the computation. We note, however, that the computation can be reduced with the use of the Fast Fourier Transform (FFT). In particular, we have adapted Cooley-Tukey FFT and Bluestein's FFT to work with encrypted values.

Parts of this chapter are based on joint work with Sebastiaan de Hoogh, Berry Schoenmakers and Boris Škorić [dHSŠV09].

Chapter 4: General Verifiable Rotations

We present a completely new approach to get a zero-knowledge proof of rotation. This solution is general, applies to any homomorphic cryptosystem, and avoids any constraints that the DFT-based approach puts on parameters. We show that the protocol satisfies witness-extended emulation, using a detailed analysis that may be of independent interest.

The solution presented in this chapter improves upon the joint work with Sebastiaan de Hoogh, Berry Schoenmakers and Boris Škorić [dHSŠV09], following a different approach.

Chapter 5: Verifiable Multiply Fragile Shuffles

We consider zero-knowledge proofs for k-fragile permutations as defined by Reiter and Wang [RW04]. In fact, we present the first zero-knowledge proofs of knowledge to show that a shuffler used a k-fragile permutation, with $k = 2, 3$. Namely, we show how to shuffle according to an affine transformation (2-fragile) and a Möbius transformation (3-fragile). The chapter is complemented with an overview of (non-)existence results for multiply fragile sets of permutations. We do this by noting the link between fragility and the well-studied concept of set transitivity.

Parts of this chapter are included in the joint work with Sebastiaan de Hoogh and Berry Schoenmakers [dHSV10].

Chapter 6: Integer Comparison

Two types of protocols for integer comparison are presented. Our first solution is an arithmetic circuit yielding a protocol of $O(\log m)$ rounds where m is the bit-size of the inputs. The computational complexity is low, as the work is only about 50% more than the most efficient known solution of [ST04], which requires $O(m)$ rounds. Our second solution is based on a different approach that achieves constant rounds assuming a fixed number of parties, while minimizing the computational work. The resulting protocol improves substantially over the constant rounds protocol of [GSV07]. The chapter ends with an overview of different variants of integer comparison protocols and related problems.

Parts of this chapter are based on joint work with Juan A. Garay and Berry Schoenmakers [GSV07].

Chapter 2

Preliminaries

In this chapter we review several basic cryptographic concepts and primitives which play an important role throughout the thesis. The presentation is informal and mainly serves to introduce the notation and terminology that we need in later chapters. For concreteness and simplicity, we use a generic discrete log setting, involving a cyclic group of prime order, as the main setting for our cryptographic constructions. Many of the results are however more generally applicable, e.g., using an RSA-based setting, and where appropriate we will discuss such generalizations.

We will pay special attention to honest verifier zero-knowledge protocols, which play a central role in the thesis. The restriction to honest verifier zero-knowledge allows for relatively simple and efficient protocols, capturing the essence of our constructions. Using the Fiat-Shamir heuristic, these protocols can be converted –without loss of efficiency– into publicly-verifiable non-interactive proofs, for which the security can be proved in the random oracle model. We will use the notion of witness-extended emulation as the main security property, which captures both soundness and zero-knowledgeness. Σ-protocols satisfy the witness-extended emulation property. We include a direct proof of this known fact to illustrate the typical structure of the witness-extended emulators for our protocols in later chapters.

We review shuffles of homomorphic encryptions and applications, introducing some basic notation. Finally, we briefly present the frameworks put forth by Cramer *et al.* [CDN01], and Schoenmakers and Tuyls [ST04] which enable general secure multiparty computation from the evaluation of arithmetic circuits. The notion of security within the framework is discussed as well.

2.1 Basic Primitives

In this section we introduce some well-known cryptographic primitives for a standard, generic discrete log setting.

Discrete Log Setting

Let $G_q = \langle g \rangle$ be a (multiplicative) cyclic group of order q. Typical examples are prime order subgroups of the multiplicative group of a finite field, or prime order subgroups of the points of an ordinary elliptic curve over a finite field.

For our applications we assume that the *decision Diffie-Hellman* (DDH) problem to be infeasible. Namely, given random g^x, g^y and g^z in G_q it is infeasible to decide whether

$z = xy \bmod q$.

The DDH assumption implies that the *computational Diffie-Hellman* (CDH) problem is infeasible as well. Namely, it is infeasible to compute g^{xy} given g^x and g^y. In turn, the *discrete log* (DL) problem, which is to compute x from g^x, is infeasible too.

2.1.1 Pedersen Commitment

Consider a discrete log setting over the group $G_q = \langle g \rangle$. Let $h \in G_q \backslash \{1\}$ be chosen at random, such that $\log_g h$ is not known. In a Pedersen commitment, two parties, a committer C and a receiver R run a protocol in the following two phases. During the *commit* phase, the committer C commits to a private value $m \in \mathbb{Z}_q$ by computing $c = \mathsf{C}(m, r) = g^r h^m$ for random $r \in \mathbb{Z}_q$ and sending c to R. In the *reveal* phase, the committer C opens the commitment c by revealing the values m and r to R who then checks that $c = \mathsf{C}(m, r) = g^r h^m$.

Pedersen commitment has two properties, *hiding* and *binding*. Hiding property says that from c alone, R cannot get any information about m. Binding guarantees that C is not able to open c to a different value other than m after c has been fixed. More concretely, the scheme is *statistically* hiding since the distribution of $g^r h^m$ is statistically independent of the value of m. The scheme is *computationally* binding under the discrete log assumption on G_q since if two openings for a commitment c would exist, that is $c = g^r h^m = g^{r'} h^{m'}$ with $m \neq m'$, then $h = g^{(r-r')/(m'-m)}$.

The notation $c = \mathsf{C}(m, r)$ is used throughout this thesis to indicate the commitment function of a general commitment scheme where c is a commitment to $m \in \mathcal{M}$ using randomness $r \in \mathcal{R}$. We assume that C satisfies the properties of hiding and binding.

2.1.2 Threshold Homomorphic ElGamal Encryption

Consider a discrete log setting over the group $G_q = \langle g \rangle$. The secret key x is selected at random from $\mathbb{Z}_q$ and the public key is $h = g^x$. We present a variation of ElGamal cryptosystem [ElG85] that is additively homomorphic.

Upon message $m \in \mathbb{Z}_q$, a random $r \in \mathbb{Z}_q$ is chosen to produce the ciphertext $(a, b) = \mathsf{E}(m, r) = (g^r, g^m h^r)$. Decryption of the ciphertext (a, b) is done by retrieving m from $g^m = b/a^x$.

A drawback, though, is that the set of possible plaintexts $M \subset \mathbb{Z}_q$ to be decrypted should be small otherwise the discrete log problem must be solved. Often $|M| = 2$. The encryption scheme is semantically secure under the DDH assumption on G_q.

Most results presented in this thesis are general and work for any homomorphic cryptosystem. Hence, E denotes the encryption algorithm of a semantically secure homomorphic public key cryptosystem with message space $\mathcal{M}$ and randomness space $\mathcal{R}$. In the case of homomorphic ElGamal, $\mathcal{M} = \mathcal{R} = \mathbb{Z}_q$. Other cryptosystems may be instantiated as well. For instance, Goldwasser-Micali cryptosystem [GM84] where $\mathcal{M} = \{0, 1\}$, $\mathcal{R} = \mathbb{Z}_N^*$, and Paillier cryptosystem [Pai99] where $\mathcal{M} = \mathbb{Z}_N$ and $\mathcal{R} = \mathbb{Z}_N^*$ for an RSA modulus N.

We use different notations for encryptions, with different meanings. Given $m \in \mathcal{M}$ and $r \in \mathcal{R}$, the full notation $e = \mathsf{E}(m, r)$ is used for the encryption of the message m using randomness r. The bracketed notation $[\![m]\!]$ is a shorthand notation of $\mathsf{E}(m, r)$ where the randomization is left implicit. In algorithms, $[\![m]\!]$ is used to denote an encrypted variable m whose actual value may not be known. The notation $e = \mathsf{E}(m)$ means a *deterministic*

encryption of the message m. That is, the randomness used is a predefined and public value.

ElGamal cryptosystem is homomorphic. Given $(a_1, b_1) = \mathsf{E}(m_1, r_1)$ and $(a_2, b_2) = \mathsf{E}(m_2, r_2)$ two ElGamal encryptions as defined above. Then, $(a_1 a_2, b_1 b_2) = \mathsf{E}(m_1 + m_2, r_1 + r_2) = (g^{r_1+r_2}, g^{m_1+m_2} h^{r_1+r_2})$ is an encryption of $m_1 + m_2$. Furthermore, given constant $m' \in \mathbb{Z}_q$, it holds that $(a_1^{m'}, b_1^{m'}) = \mathsf{E}(m_1 m', r_1 m') = (g^{r_1 m'}, g^{m_1 m'} h^{r_1 m'})$ is an encryption of $m_1 m'$.

These homomorphic properties are described in a general cryptosystem assuming that the message space $(\mathcal{M}, +, \cdot)$ is a ring. Then, given encryptions $[\![m_1]\!]$ and $[\![m_2]\!]$, a multiplicative operation for the ciphertexts corresponds to an additive transformation of the plaintexts. That is, $[\![m_1]\!][\![m_2]\!] = [\![m_1 + m_2]\!]$. Also, for a constant m', $e = [\![m]\!]^{m'}$ denotes the deterministic transformation such that $e = [\![m \cdot m']\!]$.

A particularly useful property of homomorphic cryptosystems is the *re-randomization*. Given a ciphertext $[\![m]\!]$ it can be 'randomized' by multiplying it with a random encryption of 0. Clearly, if $[\![y]\!] = [\![x]\!]\mathsf{E}(0, s)$ for random $s \in \mathcal{R}$ then it holds that $x = y$.

Threshold Decryption

For $n > 1$ and $1 \leq t \leq n$, in a (t, n)-threshold cryptosystem public key encryption scheme the secret key is shared among a set of n parties $P_1, \ldots, P_n$. Each party holds a *share* of the secret key in such a way that if at least t of the parties collaborate, they are able to decrypt messages. Less than t parties, however, get no clue about the plaintext of a ciphertext.

Homomorphic ElGamal cryptosystem admits a threshold version. Firstly, parties involve in a *distributed key generation* protocol that allows the generation of a shared secret key along with the corresponding public key. An efficient distributed key generation protocol is presented in [Ped91, GJKR99]. The underlying idea is to share a the secret key x using a secret sharing scheme.

For instance, using Shamir's secret sharing [Sha79], the secret key may be shared as follows. Let f be a polynomial of degree $t - 1$ over $\mathbb{Z}_q$ such that $x_i = f(i)$ for $i = 1, \ldots, n$ where x_i is the share of the secret key of P_i. Party P_i outputs $h_i = g^{x_i}$. The secret key is defined as $x = f(0)$. The public key is obtained by Lagrange interpolation in the exponents: given h_i for $1 \leq i \leq t$ (or any other subset of t h_i's) one can obtain $h = g^{f(0)} = g^x$.

For *threshold decryption* of ciphertext $e = (a, b)$, party P_i produces $s_i = a^{x_i}$ for $i = 1, \ldots, n$. If at least t of these shares are correct the value of $a^x = \mathsf{R}(s_{i_1}, \ldots, s_{i_t})$ can be reconstructed via Lagrange interpolation in the exponents and thus the message can be recovered by solving $g^m = b/a^x$.

The notation $m \leftarrow \text{DECR}([\![m]\!])$ is used to denote a run of the threshold decryption protocol. The parties involved in this protocol as well as the keys involved are left implicit in the notation. However, all set up parameters should be understood from the context.

Other cryptosystems also admit a threshold variant, such as the case of Paillier and its generalization in [DJ01]. It should be noted that the distributed key generation of Paillier (or any other cryptosystem where the secret key is the factorization of an RSA modulus) is a computational intensive task [ACS02]. In contrast, threshold ElGamal requires a simple protocol [Ped91, GJKR99, ST04].

2.2 Honest-Verifier Zero-Knowledge Proofs of Knowledge

Zero-knowledge proofs are a general class of protocols between two parties, a prover P and a verifier V, modeled as interactive probabilistic machines. By means of a zero-knowledge proof, the prover convinces the verifier of the validity of a given statement without revealing any information beyond the truth of the statement.

Let the pair (P, V) denote the interaction between those parties, which we call an *interactive system*. We write $\mathtt{tr} \leftarrow \langle P(x), V(y) \rangle$ to denote the messages exchanged by P and V on inputs x and y respectively. At the end of the interaction V either accepts or rejects, denoted by $\langle P(x), V(y) \rangle = b$ or $\langle \mathtt{tr}, y \rangle = b$ with $b = 1$ or $b = 0$, respectively.

Definition 2.1 (NP-Relation) *An* NP-relation *is a binary relation* $R = \{(x; w)\} \subset \{0,1\}^* \times \{0,1\}^*$ *that can be evaluated efficiently and for which there exists a polynomial* p *such that* $|w| \leq p(|x|)$. *The* language *induced by* R *is defined as* $L_R = \{x : \exists w \text{ s.t. } (x; w) \in R\}$.

The pair $(x; w)$ can be thought of as an instance of a computational problem where w is the solution to that instance.

Consider a discrete log setting over a cyclic group $G_q = \langle g \rangle$ of prime order q. Then the discrete log relation is defined as

$$R_{DL} = \{(g, h; w) : h = g^w\},$$

where $w \in \mathbb{Z}_q$. The language induced is $L_{R_{DL}} = \{(g, h) : g, h \in G_q\}$.

2.2.1 Σ-Protocols

We now consider a class of 3-move interactive systems between P and V where P acts first and V provides randomly chosen challenges. Let (a, c, t) denote the messages exchanged between P and V. Based on the transcript (a, c, t) the verifier V either accepts or rejects. The structure of a Σ-protocol is given in Fig. 2.1.

Definition 2.2 (Σ-protocol) *A 3-move protocol between interactive machines* P *and* V *where* P *acts first is a* Σ-protocol *for relation* R *if the following holds.*

- ***Completeness.*** *For all* $(x; w) \in R$, $\langle P(x, w), V(x) \rangle = 1$.
- ***Special Soundness.*** *There exists an efficient extractor* E *such that on any pair of conversations* (a, c, t) *and* (a, c', t') *on common input* $x \in L_R$ *such that* $\langle (a, c, t), x \rangle = 1$, $\langle (a, c', t'), x \rangle = 1$, *and* $c \neq c'$, E *computes* $w \leftarrow E(x, a, c, t, c', t')$ *such that* $(x; w) \in R$.
- ***Special Honest-Verifier Zero-Knowledge.*** *There exists an efficient algorithm* $\mathcal{S}$ *that on input* (x, c) *with* $x \in L_R$, $\mathcal{S}$ *outputs a transcript* (a, c, t) *with the same probability distribution as the transcripts between honest* P *and* V *with common input* x *and challenge* c *where* P *uses any witness* w *such that* $(x; w) \in R$.

In a Σ-protocol the honest verifier is limited to provide random coin tosses independently of the inputs it has received. This kind of protocols is more generally referred to as *public coin*.

Special soundness indicates that if P is able to reply to two different challenges on a fixed first message then it P actually knows the witness. It can be proved that a cheating

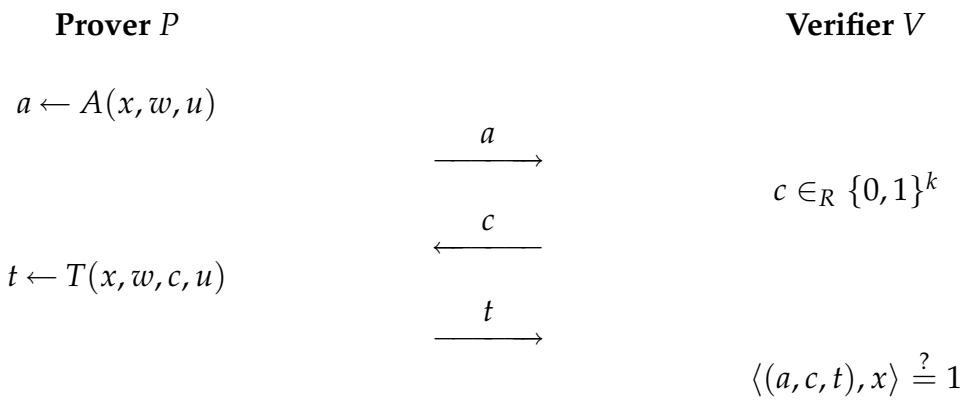


Figure 2.1: Structure of a Σ-protocol.

prover P^* that does not know a witness has probability $1/2^k$ of letting an honest verifier V accept. In other words, special soundness implies the standard notion of knowledge soundness [BG92] of proof systems. A proof of this fact can be found in [Dam08].

Σ-protocols are proof systems with zero-knowledge property withstanding honest verifiers only. This means that an honest verifier gets no information whatsoever about the witness that the prover is using. Despite withstanding only honest verifiers, Σ-protocols are very useful and powerful building blocks. For technical reasons, Σ-protocol are required to be *special* honest verifier zero-knowledge which means that the simulator $\mathcal{S}$ produces conversations for a specified challenge.

A classic example of a Σ-protocol is Schnorr's protocol [Sch91] to show the knowledge of a witness for relation R_{DL}. Other examples include Okamoto's protocol [Oka93] to prove the knowledge of the opening of a Pedersen commitment.

Verifiable Non-Interactive Proofs

Σ-protocols can be made non-interactive using a cryptographic hash function. The Fiat-Shamir heuristic [FS87] makes this conversion by replacing the random coin tosses from the verifier with calls to a cryptographic hash function. The conversion does not affect the efficiency of the protocols. The security of the resulting protocols can be shown in the random oracle model [BR93]. This is a property that holds for any public coin proof systems in general.

In the random oracle model, this technique not only makes public-coin zero-knowledge protocol non-interactive, but it forces honest behavior. Another distinctive feature of the application of the Fiat-Shamir heuristic, is that any entity may play the role of the verifier and check whether a proof is valid or not. This property is known as *public verifiability*.

Composition Properties

Σ-protocols have some easily verified properties which we review in the following. For more insightful details we refer the reader to [CDS94, Cra97].

The class of Σ-protocols is closed under parallel composition. Namely, if two instances of a Σ-protocol with challenge length k are run in parallel, then the overall resulting protocol is a Σ-protocol with challenge length $2k$. This result can be used to prove that if

a Σ-protocol exists for relation R then there is a Σ-protocol for relation R for any challenge length k.

AND-Composition. Suppose that we have two Σ-protocols with challenge length k, one for relation R_0 and the other for relation R_1. Suppose that a prover P knows witnesses w_0 and w_1 such that $(x_0; w_0) \in R_0$ and $(x_1; w_1) \in R_1$. If both Σ-protocols are run in parallel using a *common* challenge for both instances, then the result is a Σ-protocol for relation $R_0 \wedge R_1$, where $R_0 \wedge R_1$ is defined as

$$R_0 \wedge R_1 = \{(x_0, x_1; w_0, w_1) : (x_0; w_0) \in R_0 \wedge (x_1; w_1) \in R_1\}.$$

This construction, referred to as the AND-composition, can be naturally generalized to prove the knowledge of various witnesses simultaneously.

OR-Composition. Now, let two relations R_0 and R_1 be given and a common input (x_0, x_1) such that $x_0 \in L_{R_0}$ and $x_1 \in L_{R_1}$. The prover P wants to prove the knowledge of a witness w such that $(x_0; w) \in R_0$ or $(x_1, w) \in R_1$ without indicating anything else. In particular, P does not want to disclose which case holds, if either $(x_0; w) \in R_0$ or $(x_1, w) \in R_1$ holds, or both. Namely, the prover P wants to prove it knows a witness for relation $R_0 \vee R_1$, where

$$R_0 \vee R_1 = \{(x_0, x_1; w) : (x_0; w) \in R_0 \vee (x_1; w) \in R_1\}.$$

If P knows the witness for x_b for a unique $b \in \{0, 1\}$ then P is able to give an accepting proof for that instance. However, P may not be able to do the same for x_{1-b} since it may simply not know a witness for it. Note that P may run the simulator for R_{1-b} on input x_{1-b}.

A Σ-protocol for $R_0 \vee R_1$ is constructed by giving some freedom to the prover in choosing the challenges in order to compute the final answer. Fig. 2.2 gives a description of the resulting protocol. It is easy to verify that it is a Σ-protocol for $R_0 \vee R_1$ and that no information is leaked about the witness that is used.

This construction is usually referred to as the OR-composition, or proof of partial knowledge, as first introduced in [CDS94]. Analogously to the AND-composition, one can proof the knowledge of at least one witness for various relations, and of course, combine AND and OR compositions in order to prove more elaborated statements.

2.2.2 Witness-Extended Emulation

Some of the honest verifier zero-knowledge protocols presented in this thesis are not Σ-protocols. Moreover, it is not easy and simple to prove that they satisfy knowledge soundness as defined in [BG92]. We are able to prove that they satisfy a weaker property. Namely, they have a witness-extended emulator, a concept introduced by Lindell [Lin03], slightly redefined by Groth [Gro04].

Roughly speaking, an honest verifier zero-knowledge proof (P, V) has witness-extended emulation w.r.t. an NP-relation R if given an adversarial prover P^* that produces accepting conversations with probability p, there exists an extractor that produces accepting transcripts and at the same time gives a witness for R with roughly probability p.

Before going into the formal definition of witness-extended emulation, we define some basic concepts.

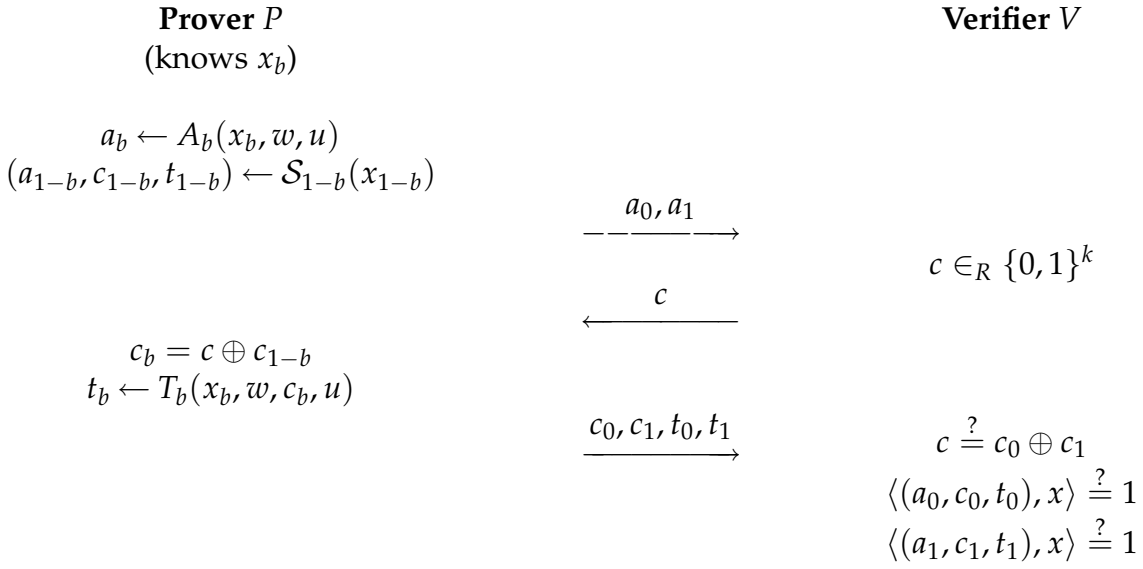


Figure 2.2: OR-composition of Σ-protocols for relations R_0 and R_1.

Definition 2.3 (Negligible Function) *A non-negative function $\delta : \mathbb{N} \to \mathbb{R}$ is said to be* negligible *if for all $c > 0$, there exists $k_0 > 0$ such that for any $k > k_0$ it holds that $\delta(k) < k^{-c}$.*

Definition 2.4 *Two functions $f, g : \mathbb{N} \to \mathbb{R}$ are* essentially the same *if $|f(k) - g(k)|$ is a negligible function.*

We write $f \cong g$ if f and g are essentially the same.

Definition 2.5 (Witness-extended Emulation) *An interactive system (P, V) has* witness-extended emulation *for relation R if for every deterministic polynomial-time machine P^* there exists an expected polynomial-time emulator $\mathcal{W}$ such that for all computationally bounded adversaries $\mathcal{A}$ we have,*

$$\mathbb{P}[(x,s) \leftarrow \mathcal{A}(1^k); \mathtt{tr} \leftarrow \langle P^*(x,s), V(x) \rangle : \mathcal{A}(\mathtt{tr}) = 1] \cong$$
$$\mathbb{P}[(x,s) \leftarrow \mathcal{A}(1^k); (\mathtt{tr}, w) \leftarrow \mathcal{W}^{P^*(x,s)}(x) : \mathcal{A}(\mathtt{tr}) = 1 \wedge (\langle \mathtt{tr}, x \rangle = 1 \Rightarrow (x; w) \in R)],$$

where k is a security parameter.

Any adversarial strategy is divided into two machines, P^* and $\mathcal{A}$. Intuitively, $\mathcal{A}$ programs P^* to perform an attack by handing over (x, s) where s can be thought as the randomness used in the attack. The emulator $\mathcal{W}$ has rewindable oracle access to P^* and it produces a pair $(\mathtt{tr}, w)$ in which the value $\mathtt{tr}$ must be indistinguishable from a transcript in which P^* interacts with V. Moreover, if $\mathtt{tr}$ happens to be an accepting conversation then w must be a witness of R. These two events must happen with essentially the same probability as P^* interacting with V would succeed in producing an accepting transcript.

The technical advantages of this definition are two-fold. In the first place, we have the original motivation of this definition given by Lindell [Lin03]. He observed that in the proof of security of a protocol, where proofs of knowledge are used as subroutines, the simulator needs both a transcript and a witness for those proofs. This may be obtained by first running the zero-knowledge simulator of each proof and later, in a separate stage, use rewinding techniques to extract a correct witness. A witness-extended

emulator obtains both in one go, simplifying the analysis considerably. Lindell [Lin03] formally proved the so-called *witness-extended emulation lemma* stating that any proof of knowledge as defined by Bellare and Goldreich [BG92] has a witness-extended emulator, and therefore it can be used in the proof of security of higher-level protocols.

In the second place, Groth [Gro04] showed that witness-extended emulation implies the concept of knowledge soundness of a *computationally convincing* proof of knowledge as defined by Damgård and Fujisaki [DF02]. Even though it is a weaker definition compared to conventional knowledge soundness [BG92], it is a widely accepted definition for protocols where a public key has been set up showing that, for example, trapdoor information of the keys gives no extra advantage to cheating provers.

Useful Results

The following lemma gives sufficient conditions to prove that a proof system has witness-extended emulation.

Lemma 2.6 *Let (P, V) be an interactive system, R an NP-relation and P^* a deterministic polynomial-time machine. An algorithm $\mathcal{W}$ with black-box access to P^* is a* witness-extended emulator *for R if for all computationally bounded adversaries $\mathcal{A}$ such that $(x, s) \leftarrow \mathcal{A}(1^k)$ the following holds: for $(\mathtt{tr}, w) \leftarrow \mathcal{W}^{P^*(x,s)}$,*

(i) Emulator $\mathcal{W}$ runs in expected polynomial-time;

(ii) The set of all transcripts $\mathtt{tr}$ produced by $\mathcal{W}$ have the same probability distribution as those produced by the real interaction $\langle P^*(x, s), V(x)\rangle$;

(iii) If $(x; w) \in R$ then $\langle \mathtt{tr}, x\rangle = 1$;

(iv) $\mathbb{P}[(x; w) \in R] \cong \mathbb{P}[\langle \mathtt{tr}, x\rangle = 1]$.

The lemma suggests that given a machine that uses black-box access to P^*, runs in expected polynomial-time, outputs transcripts that are indistinguishable from the real transcripts in the protocol, if whenever it gives a valid witness in R it gives an accepting conversation, and it provides a valid witness in R with essentially the same probability as accepting transcripts are produced, then we have a witness-extended emulator.

Before proving Lemma 2.6, we first sketch two useful properties.

Claim 2.7 *Let A and B be two events, then the following holds.*

(1) If $B \Rightarrow A$ and $\mathbb{P}[A] \cong \mathbb{P}[B]$ then $\mathbb{P}[A \Rightarrow B] \cong 1$.

(2) If $\mathbb{P}[B] \cong 1$ then $\mathbb{P}[A \wedge B] \cong \mathbb{P}[A]$.

Proof. To prove implication (1) we use basic properties of probabilities.

$$\begin{aligned}
\mathbb{P}[A \Rightarrow B] &= \mathbb{P}[\overline{A} \vee B] \\
&= 1 - \mathbb{P}[A \wedge \overline{B}] \\
&= 1 - \mathbb{P}[A] - \mathbb{P}[\overline{B}] + \mathbb{P}[A \vee \overline{B}] \\
&= \mathbb{P}[B] - \mathbb{P}[A] + \mathbb{P}[A \vee \overline{B}] \\
&\cong \mathbb{P}[A \vee \overline{B}] \\
&= \mathbb{P}[B \Rightarrow A] \\
&= 1.
\end{aligned}$$

For (2) suppose $\mathbb{P}[B] = 1 - \delta$ for negligible δ. Then,

$$\begin{aligned}\mathbb{P}[A \wedge B] &= \mathbb{P}[A] + \mathbb{P}[B] - \mathbb{P}[A \vee B] \\ &= \mathbb{P}[A] + 1 - \delta - \mathbb{P}[A \vee B] \\ &= \mathbb{P}[A] - \delta + \mathbb{P}[\overline{A} \wedge \overline{B}] \\ &\geq \mathbb{P}[A] - \delta.\end{aligned}$$

Since $\mathbb{P}[A \wedge B] \leq \mathbb{P}[A]$, we have that $\mathbb{P}[A] - \delta \leq \mathbb{P}[A \wedge B] \leq \mathbb{P}[A]$, showing that $\mathbb{P}[A] \cong \mathbb{P}[A \wedge B]$. ■

Proof of Lemma 2.6. Let $(\mathtt{tr}, w) \leftarrow \mathcal{W}(x)$. First, we show that

$$\mathbb{P}[\langle \mathtt{tr}, x \rangle = 1 \Rightarrow (x; w) \in R] \cong 1. \tag{2.1}$$

We define the event A as $\langle \mathtt{tr}, x \rangle = 1$ and the event B as $(x; w) \in R$. Then we have that hypothesis (iii) says $B \Rightarrow A$ and hypothesis (iv) states that $\mathbb{P}[A] \cong \mathbb{P}[B]$. Thus, by Claim 2.7 (1) we have that Eq. (2.1) holds.

Now, we define the events A and B as follows. Let A be the event $\mathcal{A}(\mathtt{tr}) = 1$ and B be $\langle \mathtt{tr}, x \rangle = 1 \Rightarrow (x; w) \in R$. By Claim 2.7 (2) and since $\mathbb{P}[B] \cong 1$ due to Eq. (2.1) it follows that

$$\mathbb{P}[\mathcal{A}(\mathtt{tr}) = 1 \wedge \langle \mathtt{tr}, x \rangle = 1 \Rightarrow (x; w) \in R] \cong \mathbb{P}[\mathcal{A}(\mathtt{tr}) = 1]. \tag{2.2}$$

We now show that the definition of witness-extended emulator holds.

$$\begin{aligned}& \mathbb{P}[(x, s) \leftarrow \mathcal{A}(1^k); (\mathtt{tr}, w) \leftarrow \mathcal{W}^{P^*(x,s)}(x) : \mathcal{A}(\mathtt{tr}) = 1 \wedge (\langle \mathtt{tr}, x \rangle = 1 \Rightarrow (x; w) \in R)] \\ \cong\ & \mathbb{P}[(x, s) \leftarrow \mathcal{A}(1^k); (\mathtt{tr}, w) \leftarrow \mathcal{W}^{P^*(x,s)}(x) : \mathcal{A}(\mathtt{tr}) = 1] \text{ (by Eq. (2.2))} \\ =\ & \mathbb{P}[(x, s) \leftarrow \mathcal{A}(1^k); \mathtt{tr} \leftarrow \langle P^*(x, s), V(x) \rangle : \mathcal{A}(\mathtt{tr}) = 1] \text{ (hypothesis (ii)).}\end{aligned}$$

Finally, using (i) we conclude that $\mathcal{W}$ runs in expected polynomial-time which means that $\mathcal{W}$ is a witness-extended emulator for relation R. ■

Σ-protocols have the witness-extended emulatability property. This can be proved using known results. Namely, Damgård [Dam08] proves that special soundness implies knowledge soundness which, in turn, implies witness-extended emulation using the witness-extended emulation lemma of Lindell [Lin03]. In the following, however, we present a direct proof which may be of independent interest. It follows the typical structure of a proof of witness-extended emulation in general.

Theorem 2.8 *Let (P, V) be a Σ-protocol for relation R. Then (P, V) has witness-extended emulation.*

Proof. W.l.o.g. assume that the challenge set is $\{0,1\}^k$ for security parameter k. Let P^* be a deterministic polynomial-time machine and $(x, s) \leftarrow \mathcal{A}(1^k)$. Algorithm 2.1 describes the emulator $\mathcal{W}$.

Note that once (x, s) is fixed, the fact that P^* produces an accepting transcript or not when it is challenged can be represented by a $\{0,1\}$-vector v of length 2^k. The vector v is such that $v_c = 1$ if and only if P^* produces an accepting transcript when challenge $c \in \{0,1\}^k$ is given. We define ϵ as the proportion of 1's in v. Clearly, $\epsilon = \mathbb{P}[\langle P^*(x, s), V(x) \rangle = 1]$ for an honest verifier V.

Algorithm 2.1 ($\mathcal{W}$) Witness-extended emulator for a Σ-protocol for relation R.

pick $c \in_R \{0,1\}^k$
run $(a, c, t) \leftarrow \langle P^*(x, s), \tilde{V}(c)\rangle$ $\quad \rightsquigarrow \tilde{V}(c)$ is an interactive machine that gives c to P^*.
if $\langle (a, c, t), x\rangle = 1$ **then**
  repeat
    pick $c' \in_R \{0,1\}^k$
    run $(a, c', t') \leftarrow \langle P^*(x, s), \tilde{V}(c')\rangle$
  until $\langle (a, c', t'), x\rangle = 1$
  if $c \neq c'$ **then**
    run $w \leftarrow E(x, a, c, t, c', t')$
    return $((a, c, t), w)$
  else
    return $((a, c, t), \perp)$
  end if
else
  return $((a, c, t), \perp)$
end if

We show that all conditions of Lemma 2.6 are met. To show (ii), one can easily see that the distribution of the transcripts $\mathtt{tr}$ that $\mathcal{W}$ outputs is identically distributed as that of the real execution of the Σ-protocol. As for (iii), observe that whenever a valid witness is produced, it comes with an accepting transcript $\mathtt{tr}$. Conditions (i) and (iv) will be shown respectively in the two following claims.

Claim 2.9 *Emulator $\mathcal{W}$ runs in expected polynomial-time.*

Proof. The running time of $\mathcal{W}$ is governed by the number of invocations of P^*. Let T be the random variable counting the number of invocations of P^*. We calculate $\mathbb{E}[T]$. The running time of $\mathcal{W}$ is clearly determined by the condition that $\langle (a, c, t), x\rangle = 1$. Therefore we have that

$$\begin{aligned}
\mathbb{E}[T] &= \mathbb{P}[\langle (a, c, t), x\rangle = 1]\mathbb{E}[T \mid \langle (a, c, t), x\rangle = 1] + \\
&\quad\ \mathbb{P}[\langle (a, c, t), x\rangle = 0]\mathbb{E}[T \mid \langle (a, c, t), x\rangle = 0] \\
&= \epsilon(1 + 1/\epsilon) + (1 - \epsilon) \\
&= 2.
\end{aligned}$$

Thus, $\mathcal{W}$ needs 2 expected runs of P^* which itself runs in *strict* polynomial time. This means that $\mathcal{W}$ runs in expected polynomial-time. ■

Claim 2.10 *Given $(\mathtt{tr}, w) \leftarrow \mathcal{W}^{P^*(x,s)}(x)$, then $\mathbb{P}[(x; w) \in R] \cong \mathbb{P}[\langle \mathtt{tr}, x\rangle = 1]$.*

Proof. We know that $\epsilon = \mathbb{P}[\langle P^*(x, s), V(x)\rangle = 1]$. Note that since the transcript $\mathtt{tr}$ given by $\mathcal{W}$ is generated in the same way as when an instance of the protocol (P, V) is run with an honest verifier, we have that $\epsilon = \mathbb{P}[\langle \mathtt{tr}, x\rangle = 1]$.

Consider the event that w is a valid witness, that is, $(x; w) \in R$. This only happens when the first transcript (a, c, t) is accepting (with probability ϵ) and the second transcript (a, c', t') hits a $c' \neq c$. We analyze the probability that $c' \neq c$ in terms of vector v.

Vector v has $2^k\epsilon$ 1's. The first accepting transcript (a, c, t) hits one of the 1's. We are interested to know if the second accepting transcript used $c' \neq c$, i.e., it hits a different 1. This yields the probability $(2^k\epsilon - 1)/(2^k\epsilon) = 1 - 1/2^k\epsilon$ that $\mathcal{W}$ gets to run extractor E. We therefore get $\mathbb{P}[(x; w) \in R] = \epsilon(1 - 1/2^k\epsilon) = \epsilon - 1/2^k \cong \epsilon$. ■

We meet all conditions of Lemma 2.6. Thus, $\mathcal{W}$ is a witness-extended emulator for a Σ-protocol. ■

2.2.3 Some Useful Relations

We now describe some useful NP-relations that will be used throughout this thesis. We assume that an homomorphic cryptosystem is set up in advance. If we consider a discrete log setting, the protocols for these relations are based on standard protocols like Schnorr's proof of knowledge of discrete logs [Sch91], and Okamoto's proof of knowledge of two exponents [Oka93] and suitable compositions thereof.

As a notational remark that will be used later the thesis, we use $\mathcal{S}_{DL}$ to denote the simulator, and $\mathcal{W}_{DL}$ denotes the witness-extended emulator of the corresponding Σ-protocol for relation R_{DL}.

Known Plaintext. A prover wants to show that a given ciphertext c encrypts a public plaintext x. That is, if $c = \mathsf{E}(x, s)$ for some random s, the prover wants to prove the knowledge of s without giving it away.

Since the cryptosystem is homomorphic, the problem can be rephrased to prove that $c\mathsf{E}(-x) = \mathsf{E}(0, s)$ where $\mathsf{E}(-x)$ is a deterministic encryption of $-x$. With d defined by $d = c\mathsf{E}(-x)$ the prover shows the knowledge of a witness for the following relation:

$$R_{\text{ZERO}} = \{(d; s) : d = \mathsf{E}(0, s)\}.$$

With R_{KNOWN} defined by

$$R_{\text{KNOWN}} = \{(c, x; s) : c = \mathsf{E}(x, s)\},$$

we conclude that $(d; s) \in R_{\text{ZERO}}$ if and only if $(d\mathsf{E}(x), x; s) \in R_{\text{KNOWN}}$.

Secret Plaintext. A prover wants to prove knowledge of *both* the plaintext and randomness of a given encryption. That is, given $c = \mathsf{E}(x, s)$ no information on both x and s is disclosed. This is captured by the following relation:

$$R_{\text{SECRET}} = \{(c; x, s) : c = \mathsf{E}(x, s)\}.$$

1-out-of-2 Plaintexts. A prover wants to show that $c = \mathsf{E}(x, s)$ with $x \in \{a, b\}$ without revealing neither x nor the value s. That is, a proof is given for the following relation:

$$R_{\binom{2}{1}\text{KNW}} = \{(c, a, b; x, s) : c = \mathsf{E}(x, s) \wedge x \in \{a, b\}\}.$$

This can be done via an OR-composition of two instances of R_{KNOWN}. That is, $(c, a, b; x, s) \in R_{\binom{2}{1}\text{KNW}}$ if and only if $(c, a; s) \in R_{\text{KNOWN}} \vee (c, b; s) \in R_{\text{KNOWN}}$.

The relation can be used to prove for instance that the plaintext of an encryption is a bit, or the plaintext is in the set $\{-1, 1\}$. We can generalize the relation to a prove that the plaintext is one of out of a finite number of possible plaintexts in a reasonably straightforward way [CDS94].

Non-zero Secret Plaintext. We have ciphertext $c = \mathsf{E}(x, s)$ and want to prove that the we know a plaintext and it is different from 0. We have the following relation:

$$R_{\text{NO-ZERO}} = \{(c; x, s) : c = \mathsf{E}(x, s) \wedge x \neq 0\}.$$

Correct Re-randomization. In an homomorphic cryptosystem it is possible to re-randomize a given encryption into another one such that both decrypt to the same plaintext. This is done by multiplying an encryption by a random encryption of 0. That is, given c, it follows that if $d = c\mathsf{E}(0, s)$ then $x = y$ where $c = [\![x]\!]$ and $d = [\![y]\!]$. However, as a consequence of the semantic security of the cryptosystem, given c and d it is impossible for an outsider to verify whether they have the same plaintext.

We want a protocol for the following relation:

$$R_{\text{BLIND}} = \{(c, d; s) : d = c\mathsf{E}(0, s)\}.$$

This relation can be put in terms of R_{ZERO}. Observe that $d = c\mathsf{E}(0, s)$ if and only if $dc^{-1} = \mathsf{E}(0, s)$ which means that $(dc^{-1}; s) \in R_{\text{ZERO}}$.

Multiplicative Relations. The prover shows knowledge of secret plaintexts in encryptions $c_1 = \mathsf{E}(x_1, s_1)$, $c_2 = \mathsf{E}(x_2, s_2)$ and $d = \mathsf{E}(y, t)$ satisfying $y = x_1 x_2$. Put in other terms, a prover wants to show that the plaintexts in the encryptions satisfy a multiplicative relation [CD98]. The relation is defined as follows:

$$R_{\text{MULT}} = \{(c_1, c_2, d; x_1, x_2, s_1, s_2, t) : c_1 = \mathsf{E}(x_1, s_1) \wedge c_2 = \mathsf{E}(x_2, s_2) \wedge d = \mathsf{E}(x_1 x_2, t)\}.$$

Private Multiplier. Using homomorphic properties of the cryptosystem one can multiply the plaintext of an encryption $c = [\![x]\!]$ with a publicly known constant y, by simply performing c^y. Sometimes, however, the constant y has to be kept secret. This is easily achieved by blinding c^y with a random encryption of 0. That is, define $d = c^y\mathsf{E}(0, s)$ for a random s.

For showing that encryption d is well-formed, we consider the following relation:

$$R_{\text{PRV-MLT}} = \{(c, d, e; y, s, t) : d = c^y\mathsf{E}(0, s) \wedge e = \mathsf{E}(y, t)\}.$$

Here, the auxiliary encryption $e = [\![y]\!]$ is published. If $(c, d, e; y, s, t) \in R_{\text{PRV-MLT}}$ and $c = [\![x]\!]$ then it holds that $d = [\![xy]\!]$.

Note that if the plaintext of c is known then the relation R_{MULT} defined above can be used instead. In case of ElGamal encryptions, encryption e may be replaced by a Pedersen commitment to y. This results in a slightly optimized protocol for private multiplier, see [ST04].

Relation	Computation		Communication
	Prove	Verify	
R_{KNOWN} R_{ZERO} R_{BLIND}	2	3	2
R_{SECRET}	1	3	2
$R_{\binom{2}{1}\text{KNW}}$	4	5	4
$R_{\text{NO-ZERO}}$	3	5	3
R_{MULT}	5	4	4
$R_{\text{PRV-MLT}}$	5	6	4

Table 2.1: Complexity figures for Σ-protocols for basic relations based on homomorphic ElGamal. Computation is given by the number of modular exponentiations while communication expresses the number of group elements exchanged.

Performance Considerations

For the case in which homomorphic ElGamal is used as underlying cryptosystem, the complexity figures of the Σ-protocols for the above relations are given in Table 2.1. An estimation of the communication is given by the number of group elements that need to be exchanged. For the number of computations, we estimate the modular exponentiations needed. For the count of modular exponentiations, we assume that the product of two (resp. three) exponentiations cost 1.25 and (resp. 1.5) single exponentiations. This can be done using "Shamir's trick" described in [ElG85] which is a special case of Straus' algorithm [Str64]. For the total number of exponentiations we round to the closest integer.

Later in this thesis, we count the product of n exponentiations simply as n single exponentiations, even though there are many ways to speed up such multiexponentiations. We highlight that defining the "right" criteria to benchmark protocols depends in many factors, such as the platform where the protocols are executed, storage considerations, etc. It is difficult therefore to decide what is the most efficient protocol just based in the convention for the estimation of the computational complexity that we use in this thesis.

2.3 Verifiable Shuffles of Homomorphic Encryptions

A shuffle of a list of n ciphertexts is another list of n ciphertexts such that the multiset of plaintexts of both lists of ciphertexts is the same. Put in other words, there exists a permutation linking the plaintexts of the first list of encryptions with the plaintexts of the second list of encryptions. If the permutation does not need to be hidden, a shuffle is obtained easily: a permutation of the ciphertexts is performed in the clear. For cryptographic applications, though, it is required that such permutation is kept secret.

Shuffles of homomorphic encryptions have become popular since they are a powerful building block, applicable in many contexts. They are conceptually simple and easy to obtain. Given a list of n homomorphic encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$, we select a permutation π of the set $\{0,\ldots,n-1\}$ and compute the list of encryptions $\{[\![y_k]\!]\}_{k=0}^{n-1}$ by performing $[\![y_{\pi(k)}]\!] = [\![x_k]\!]\mathsf{E}(0,s_k)$ for random $s_k \in \mathcal{R}$, for all $0 \leq k < n$. Clearly, secrecy of π

follows from the indistinguishability of the encryptions (i.e., the semantic security of the cryptosystem).

Due to the semantic security as well, an outsider cannot verify that the shuffle was performed correctly. If, say, $[\![y_0]\!]$ is a random encryption of an arbitrary message then no one can decide on its own that $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and $\{[\![y_k]\!]\}_{k=0}^{n-1}$ are a shuffle of each other. Therefore, there must be a mechanism to verify a shuffle of the two sequences of homomorphic encryptions yet keeping the permutation secret.

We define the relation R_{SHUFFLE}, that captures a shuffle of homomorphic encryptions.

$$R_{\text{SHUFFLE}} = \{(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; \pi, \{s_k\}_{k=0}^{n-1}) : [\![y_{\pi(k)}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k)\}.$$

Giving a zero-knowledge proof of knowledge for this relation guarantees both goals of keeping the permutation secret and assuring that both lists of encryptions are indeed a shuffle of each other. If the proof is public coin one obtains a verifiable non-interactive proof of shuffle, referred to as *verifiable shuffle*.

In the literature there have been many attempts to get the complexity of these proofs of knowledge to practical levels. In fact, it is possible to get $O(n)$ computation and communication, where n is the number of encryptions being shuffled, see e.g. [Nef01, FS01, Gro03, Fur05, GL07].

2.3.1 Applications

The idea of shuffles of encryptions was first introduced by Chaum [Cha81] along with applications in anonymous email and voting. The most common use of verifiable shuffles is in the construction of verifiable mix-networks [SK95]. In a mix-network or a *cascade* of shuffles there are, say, m authorities each performing a verifiable shuffle. These authorities take a list of n homomorphic encryptions which they verifiably shuffle one after the other. Overall, if all shufflers performed the shuffle correctly, the output list of the cascade is a shuffle of the input list. Moreover, if at least one of the shufflers does not reveal the permutation it used during its turn, the permutation in the entire cascade is secret.

Mix-networks are a fundamental primitive for electronic voting protocols as a way of anonymizing encrypted votes [SK95, Abe99, FS01, Nef01, FMM+03]. They have been applied to solutions for secure integer comparison as a way to destroy leaking information [BK04, ABFL06, DGK07, GSV07].

2.3.2 Public Shuffle

We present basic solutions to a related problem of shuffles of homomorphic encryptions. Namely, we describe some relations to prove that a shuffler applies a *public* permutation. This particular problem is used in Chapter 5.

We should not get confused with a general shuffle where, as explained before, the permutation used does not have to be released. Here, we instead consider problem in which the permutation is public but the blinding randomizers used must be kept secret. Namely, we consider the relation R_{PERM} defined as follows:

$$R_{\text{PERM}} = \{(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}, \pi; \{s_k\}_{k=0}^{n-1}) : [\![y_{\pi(k)}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k)\}.$$

A zero-knowledge proof of knowledge for this relation is given using a composition of basic proofs. The fact that $(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}, \pi; \{s_k\}_{k=0}^{n-1}) \in R_{\text{PERM}}$ is equivalent to

saying that $(\llbracket x_{\pi(k)} \rrbracket, \llbracket y_k \rrbracket; s_k) \in R_{\text{BLIND}}$ for all $0 \leq k < n$. This can be accomplished by an n-way AND-composition thus giving an $O(n)$ complexity protocol.

Define now the relation $R_{\text{OR-PERM}}$ in which 1-out-of-2 known permutations is applied. Here, the permutation used is not revealed.

$$R_{\text{OR-PERM}} = \{(\{\llbracket x_k \rrbracket\}_{k=0}^{n-1}, \{\llbracket y_k \rrbracket\}_{k=0}^{n-1}, \pi_1, \pi_2; \pi, \{s_k\}_{k=0}^{n-1}) : \\ \llbracket y_{\pi(k)} \rrbracket = \llbracket x_k \rrbracket \mathsf{E}(0, s_k),\ \pi \in \{\pi_1, \pi_2\}\}.$$

A solution is obtained by an OR-composition of R_{PERM}.

This suggests a way to get to a solution for R_{SHUFFLE} via an $n!$-way OR-composition of R_{PERM}. Certainly, this yields an $O(n!)$ complexity protocol which is no practical at all. Note that in some applications, n may be around one million.

2.4 Secure Computation from Threshold Homomorphic Cryptosystems

Cramer, Damgård and Nielsen [CDN01] put forth an approach to achieve multiparty computation based on threshold homomorphic cryptosystems. Namely, a set of n parties set up a (t, n)-threshold homomorphic cryptosystem with plaintext space being the ring $\mathcal{M}$. A function f is represented as an *arithmetic circuit* over the ring $\mathcal{M}$. Each gate of the circuit is evaluated in such a way that no information about input/output wires is revealed.

More concretely, the approach is as follows. An m-input function f is represented as a circuit over the ring $\mathcal{M}$. Given the encryptions $\llbracket x_1 \rrbracket, \ldots, \llbracket x_m \rrbracket$, parties get involved in a protocol to produce an encryption $\llbracket f(x_1, \ldots, x_m) \rrbracket$ as a result. Every gate of the circuit is evaluated in a similar fashion: an encryption of the gate's output wire is produced starting from the encryptions of its input wires. This gate-by-gate evaluation is done such that no information about the actual values of manipulated wires is ever leaked.

Addition and multiplication by a public constants are gates that can be publicly evaluated in a deterministic way using homomorphic properties, without any interaction among the parties. Multiplication gates, however, require the execution of a protocol. A *multiplication protocol* takes two encryptions $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ as input and produces a random encryption $\llbracket z \rrbracket$ such that $z = xy$ without leaking any information about x, y or xy. We denote a run of the multiplication protocol with $\llbracket z \rrbracket \leftarrow \text{MULT}(\llbracket x \rrbracket, \llbracket y \rrbracket)$. Note that the actual plaintexts x and y need not be known to any party.

2.4.1 Efficiency of Arithmetic Circuits

Since multiplication gates represent the most expensive part of the execution of an arithmetic circuit, they usually give an indication of the round, broadcast and computational complexities of a protocol that evaluates the circuit. The usual measures for an arithmetic circuit are its *size*, defined as the total number of multiplication gates, and the *depth* given by the length of the critical path of multiplication gates. The size of a circuit is associated with the computational and broadcast complexities of the secure evaluation of a circuit. The depth gives an indication of the round complexity.

Addition and multiplication by public constants are assumed to be costless. No interaction is needed, parties evaluate them locally and, although some computation may be still required, multiplication protocols require higher order computations overall.

2.4.2 Concrete Instantiations

Multiplication gates are obtained differently depending on the underlying cryptosystem. In [CDN01] it is shown how to build multiplication gates for cryptosystems such as Paillier [Pai99] or its generalization [DJ01].

Schoenmakers and Tuyls [ST04] proved the impossibility of general multiplication gates under threshold homomorphic ElGamal. They showed, though, that a limited multiplication gate is realizable, the so-called *conditional gate*. This multiplication protocol only works when one of the multiplicands is in a two-valued domain, e.g. when one of the multiplicands is a bit.

2.4.3 Secure Gates

As mentioned earlier, any function represented as an arithmetic circuit composed of additions and multiplications can be evaluated securely within the framework by Cramer *et al.* [CDN01]. For certain functions, however, one may be able to provide protocols that evaluate them securely. Namely, there is a protocol that computes $[\![f(x_1, \ldots, x_m)]\!]$ from $[\![x_1]\!], \ldots, [\![x_m]\!]$, and it is not the direct evaluation of a circuit. If this protocol is proved to be secure, then it can be used as new gate within the framework of [CDN01].

Informally speaking, a protocol is secure if the view of corrupted parties can be simulated efficiently. More concretely, there is a simulator that on any sets of input and output of the protocol is able to reproduce the view created by the honest parties only having access to the information that corrupted parties are entitled to know.

As an example of the simulation of a protocol, we present a simplified version of the multiplication protocol given by Schoenmakers and Tuyls [ST04], the so-called conditional gate. We assume that 2 parties have set up a (2,2)-threshold homomorphic ElGamal. Protocol 2.1 gives the description of the two-party conditional gate. Correctness of the protocol follows from observing that the output is an encryption of $x_2y_2 = s_1s_2xs_1s_2y = xy$ since $s_1^2 = s_2^2 = 1$.

The security of this protocol in the semi-honest case follows intuitively from the observation that apart from randomized encryptions, parties only see the decrypted value x_2 which gives no info on the inputs or the output. In fact, x_2 hides statistically the original value of x which is multiplied by a random and unknown value in $\{-1, 1\}$.

In order to withstand malicious behavior each step of the protocol is accompanied with zero-knowledge proofs of knowledge. In fact, parties use a private multiplier relation $R_{\text{PRV-MLT}}$ and a proof that the private multiplier is either -1 or 1, using relation $R_{\binom{2}{1}\text{KNW}}$. Threshold decryption, denoted as DECR, is a protocol run by the two parties (see Section 2.1.2).

Algorithm 2.2 describes a simulation of Protocol 2.1 assuming that party P_1 is corrupted. The simulator runs witness-extended emulators and simulators for the proofs of zero-knowledge for the needed relations. Also, as a technical remark, it is assumed that for $x \leftarrow \text{DECR}([\![x]\!])$ there is a simulator $\mathcal{S}_{\text{DECR}}$. This simulator gets $[\![x]\!]$ (the ciphertext to decrypt) and x (the plaintext of the ciphertext) and gives a statistically indistinguishable view of that obtained during a run of $x \leftarrow \text{DECR}([\![x]\!])$.

An explanation of Algorithm 2.2 follows. First, the simulator waits for the encryptions that Party P_1 is supposed to give. Then, witness-extended emulators for the corresponding proofs are run in order to both get a transcript and extract the witnesses. The transcripts are printed by the simulator. Extracted witnesses are used to check that P_1 is

Protocol 2.1 Multiplication gate

Input: $[\![x]\!]$, $[\![y]\!]$, $x \in \{-1, 1\}$
Output: $[\![xy]\!]$

Party P_1		**Party** P_2
pick $s_1 \in_R \{-1, 1\}$		
pick $t_1, t_2, t_3 \in_R \mathcal{R}$		
$[\![x_1]\!] = [\![x]\!]^{s_1}\mathsf{E}(0, t_1)$		
$[\![y_1]\!] = [\![y]\!]^{s_1}\mathsf{E}(0, t_2)$		
$[\![s_1]\!] = \mathsf{E}(s_1, t_3)$		
zk-proof [		
$([\![x]\!], [\![x_1]\!], [\![s_1]\!]; s_1, t_1, t_3) \in R_{\text{PRV-MLT}}$		
$([\![x]\!], [\![y_1]\!], [\![s_1]\!]; s_1, t_2, t_3) \in R_{\text{PRV-MLT}}$		
$([\![s_1]\!], -1, 1; s_1, t_3) \in R_{\binom{2}{1}\text{KNW}}$]		
	$\xrightarrow{\quad [\![x_1]\!], [\![y_1]\!], [\![s_1]\!] \quad}$	
		pick $s_2 \in_R \{-1, 1\}$
		pick $u_1, u_2, u_3 \in_R \mathcal{R}$
		$[\![x_2]\!] = [\![x_1]\!]^{s_2}\mathsf{E}(0, u_1)$
		$[\![y_2]\!] = [\![y_1]\!]^{s_2}\mathsf{E}(0, u_2)$
		$[\![s_2]\!] = \mathsf{E}(s_2, u_3)$
		zk-proof [
		$([\![x_1]\!], [\![x_2]\!], [\![s_2]\!]; s_2, u_1, u_3) \in R_{\text{PRV-MLT}}$
		$([\![y_1]\!], [\![y_2]\!], [\![s_2]\!]; s_2, u_2, u_3) \in R_{\text{PRV-MLT}}$
		$([\![s_2]\!], -1, 1; s_2, u_3) \in R_{\binom{2}{1}\text{KNW}}$]
	$\xleftarrow{\quad [\![x_2]\!], [\![y_2]\!], [\![s_2]\!] \quad}$	
run $x_2 \leftarrow \text{DECR}([\![x_2]\!])$		
return $[\![y_2]\!]^{x_2}$		

executing the protocol correctly. If this is not the case, the simulator aborts. In the second part, the simulator emulates the view of Party P_2. The first detail to notice is that the protocol for threshold decryption on $[\![x_2]\!]$ has to be simulated. Hence, the simulator must provide $[\![x_2]\!]$ and the plaintext x_2 to $\mathcal{S}_{\text{DECR}}$. Since this cannot be done by the simulator on the "actual" $[\![x_2]\!]$, it will generate a random $x_2' \in \{-1, 1\}$ instead. The rest of simulation is completed thanks to the simulators for private multiplier.

In the following, we note that all is done in a consistent way meaning that the values of the manipulated plaintexts follow the same probability distribution as in the real protocol. For instance, the simulator $\mathcal{S}_{\text{PRV-MLT}}$ is run with input $([\![x_1]\!], [\![x_2]\!], [\![s_2]\!])$ which means that the plaintexts should satisfy that $x_2 = s_2 x_1$. This is indeed the case, because s_2 is defined as $x_2'(s_1 x) = x_2' x_1$ where, by definition, $x_2' = x_2$ and $x_1 = s_1 x \in \{-1, 1\}$. Finally, we have that $s_2 = x_2 x_1$ which is equivalent to $x_2 = s_2 x_1$, exactly as in the definition of s_2 in the simulation.

The same check can be done to the other simulation for $\mathcal{S}_{\text{PRV-MLT}}$ with inputs $[\![y_1]\!]$, $[\![y_2]\!]$, and $[\![s_2]\!]$. We have see that $y_2 = y_1 s_2 = s_1 x x_2 x_1 = s_1 x x_2 s_1 y = x_2 xy$ as defined in the protocol with the encrypted values.

In the last step, the simulator of the threshold decryption is invoked which we as-

Algorithm 2.2 Simulator for Protocol 2.1

Input: $[\![x]\!], [\![y]\!], [\![xy]\!]$
Party P_1 gives $[\![x_1]\!], [\![y_1]\!], [\![s_1]\!]$
run $(\mathtt{tr}_1, (s_1, t_1, t_3)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![x]\!], [\![x_1]\!], [\![s_1]\!])$
run $(\mathtt{tr}_2, (s_1, t_2, t_3)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![y]\!], [\![y_1]\!], [\![s_1]\!])$
run $(\mathtt{tr}_3, (s_1, t_3)) \leftarrow \mathcal{W}_{\binom{2}{1}\text{KNW}}([\![s_1]\!], -1, 1)$
print $\mathtt{tr}_1, \mathtt{tr}_2, \mathtt{tr}_3$
if $s_1 \notin \{-1, 1\}$ or any of transcript is non-accepting **then**
  abort
end if

pick $x_2' \in_R \{-1, 1\}$
$[\![s_2]\!] = [\![x]\!]^{x_2' s_1}$
$[\![x_2]\!] = [\![x_2']\!]$
$[\![y_2]\!] = [\![xy]\!]^{x_2'}$
print $[\![x_2]\!], [\![y_2]\!], [\![s_2]\!]$
print $\mathcal{S}_{\text{PRV-MLT}}([\![x_1]\!], [\![x_2]\!], [\![s_2]\!])$
print $\mathcal{S}_{\text{PRV-MLT}}([\![y_1]\!], [\![y_2]\!], [\![s_2]\!])$
print $\mathcal{S}_{\binom{2}{1}\text{KNW}}([\![s_2]\!], -1, 1)$
print $\mathcal{S}_{\text{DECR}}([\![x_2]\!], x_2)$

sumed it gives a indistinguishable view given $[\![x_2]\!]$ and x_2. All the encryptions produced by the simulation have plaintexts that are consistent with the distribution of the protocol. The simulators produce statistically indistinguishable views, and so does the simulator for Protocol 2.1.

Chapter 3

Verifiable Rotations using the Discrete Fourier Transform

In this chapter, we present the first protocols for special verifiable shuffles. Namely, we give a zero-knowledge proof of knowledge to show that a shuffle is performed on a cyclic rotation instead of a general permutation. We present the problem of verifiable rotation including an overview of some applications. Later on, we focus our attention on the Discrete Fourier Transform (DFT) and show how it can be used to express conveniently a rotation of homomorphic encryptions. In the end, we give an overview on the Fast Fourier Transform and show how it can be used to optimize the application of the rotation protocol based on the DFT.

3.1 Verifiable Rotation

A rotation of a list of n encryptions is a new list of n encryptions in which the multisets of plaintexts of both lists of encryptions are a cyclic rotation of each other. This can be seen as a special shuffle of encryptions where instead of a general permutation $\pi \in \mathbb{S}_n$ linking the plaintexts of the two lists of encryptions, the permutation π is of the form $\pi(k) = k + r \bmod n$ for some $0 \leq r < n$. Similar to shuffles, we require that the rotation offset is kept secret.

When we consider a homomorphic cryptosystem, a rotation of a list of n encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ can be achieved by rotating the encryptions and re-randomizing them like in most shuffling schemes based on homomorphic encryptions. The list $\{[\![y_k]\!]\}_{k=0}^{n-1}$ defined then as $[\![y_{k+r}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k)$, for random $s_k \in \mathcal{R}$ for all $0 \leq k < n$ where r, with $0 \leq r < n$, is the rotation offset applied.

Secrecy of the rotation offset r between the two lists of encryptions comes from the semantic security of the cryptosystem. Verifying that the rotation was correctly performed without disclosing the rotation offset is guaranteed via a zero-knowledge proof for relation R_{ROT} defined as follows:

$$R_{\text{ROT}} = \{(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; r, \{s_k\}_{k=0}^{n-1}) : [\![y_{k+r}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k)\}.$$

The prover giving a proof for this relation is referred to as a *rotator*.

3.1.1 Cascade of Rotators

Similarly to a cascade of shuffles (a.k.a. mix-network), a cascade of verifiable rotations yields a verifiable rotation of the input/output of the cascade which, unless all rotators collude, the overall rotation offset is not known to any of the parties. In a cascade of verifiable rotations, a group of rotators sequentially take turns, one after the other, in rotating and re-randomizing the list of homomorphic encryptions obtained from the previous rotator. Each rotator proves in zero-knowledge that it knows witnesses for relation R_{ROT}.

It is easy to verify that if all these proofs are accepting then the initial and final lists of encryptions in the cascade are a rotation of each other. Moreover, if at least one of the rotators in the cascade chooses a random rotation offset and keeps it secret then the overall rotation offset is random and remains unknown.

3.1.2 Applications

Verifiable rotation has actually been introduced by Reiter and Wang [RW04] in the context of mix-networks. Under the name of "loop permutation", they define the more general concept of 'fragile mixing' as a form of shuffling that deters leakage of information. Namely, when a single input-output correspondence of the permutation used in the fragile mix is revealed, then the permutation is completely revealed. A fragile mix may therefore be restricted to the use of rotations. The protocol for rotations by Reiter and Wang uses *four* invocations of a verifiable shuffle protocol (and some further work) to perform a verifiable rotation. In contrast, the solutions in this thesis reduce this to the work of about one verifiable shuffle, by following completely different, more direct approaches to the problem.

Apart from fragile mixing, however, there are many more applications of rotations of homomorphic encryptions. An important application arises in the context of secure integer comparisons, as first noted in [RT09]. A common step in many integer comparison protocols [BK04, ABFL06, GSV07, DGK07, RT09], requires parties to find out whether a special value occurs in a given list of encryptions. For example, whether there is a 0 among otherwise random values. The position of the special value should remain hidden. To this end, the list will be randomly permuted before decrypting the encryptions. However, rather than using a fully random permutation, as commonly proposed, a random rotation suffices to hide the position of the special value.

Similarly, it is easily seen that for protocols such as Mix & Match [JJ00], which involve mixing of truth tables of Boolean gates, it suffices to apply a random rotation to the rows of a truth table, rather than a fully random permutation. The reason is that in the matching stage exactly one row will match, and a random rotation fully hides the corresponding row of the original truth table. The same observation applies to other forms of 'garbled circuit evaluation', which can be seen as variations of Yao's original method [Yao86]. Likewise, in protocols for secure linear programming [LA06] the position of the pivot in each iteration of the simplex algorithm must be hidden to avoid leakage of information. Again, we note that the use of a rotation instead of a general permutation suffices.

Finally, we note that further applications exist in the context of electronic voting, where randomly rotated lists of encryptions are used in the construction of encrypted ballot forms (see, e.g., Prêt-à-Voter voting systems by Ryan and Schneider [RS06, Rya06]

and references therein): voters get a receipt in which one out of n positions is marked; due to a random rotation, the marked position does not reveal the identity of the candidate chosen. Wen and Buckland [WB09] present a secure protocol for determining the winner of an election in a preferential electoral system which uses rotations as a method to conceal sensitive information when scrutinizing an election.

3.2 DFT-based Solution

The key mathematical tool the solution presented in this chapter is the Discrete Fourier Transform (DFT). Using the DFT one can express conveniently that two lists of encryptions are rotated versions of each other. This allows for an efficient Σ-protocol to make a rotation verifiable.

3.2.1 Introduction

Roughly speaking, the Fourier Transform converts a function into another. Both functions are usually defined over different domains. Some manipulation of data in the transformed domain may represent a certain kind of manipulation of data in the original domain, and vice versa. The Fourier transform is invertible and consequently after some processing has been applied in one domain, the transform (or its inverse) is applied to execute some dual processing in the other domain.

In signal processing, functions defined on the complex plane are usually considered. The domain in which the original data live is usually referred to as the *time domain*, while the data after the transform is applied is said to be in the *frequency domain*. For instance, a chord of music which is a combination of notes per unit of time, can be described as a number of oscillatory functions per frequency. The DFT realizes this conversion between domains in a very simple way.

The DFT can be adapted to any finite field. Although it lacks any physical interpretation, it is still attractive due to its interesting properties. In this thesis, we focus on the Discrete Fourier Transform over a finite field and exploit its properties to conveniently derive some rules for manipulating lists of finite field elements. Concretely, one can express, in an easy manner, a rotation over a list of finite field elements, on which the DFT is well-defined.

3.2.2 Discrete Fourier Transform

Consider the field of integers reduced modulo a prime q. Let n be a divisor of $q-1$. There exists an element $\alpha \in \mathbb{Z}_q$ of order n, meaning that $\alpha^n = 1 \bmod q$ and $\alpha^j \neq 1 \bmod q$ for all $1 \leq j < n$. We say that α is a *primitive n-th root of unity* and assume it to be fixed in the sequel.

Definition 3.1 (Discrete Fourier Transform) *Given a list $\{x_k\}_{k=0}^{n-1}$ of n elements of $\mathbb{Z}_q$ its* Discrete Fourier Transform *with respect to α is the list $\{x'_k\}_{k=0}^{n-1}$ defined by*

$$x'_k = \sum_{j=0}^{n-1} x_j \alpha^{kj} \bmod q. \tag{3.1}$$

We write $\{x'_k\}_{k=0}^{n-1} = \text{DFT}(\{x_k\}_{k=0}^{n-1})$ to denote the application of the DFT.

This is an invertible transformation. The inverse DFT is presented in the next proposition together with a proof that it indeed is the inverse.

Proposition 3.2 *Let $\{x'_k\}_{k=0}^{n-1} = \text{DFT}(\{x_k\}_{k=0}^{n-1})$, we define the list $\{\tilde{x}_k\}_{k=0}^{n-1}$ as follows:*

$$\tilde{x}_k = n^{-1} \sum_{i=0}^{n-1} x'_i \alpha^{-ik} \bmod q. \tag{3.2}$$

Then it holds that $x_k = \tilde{x}_k$ for all $0 \leq k < n$.

Proof.

$$\begin{aligned}
\tilde{x}_k &= n^{-1} \sum_{i=0}^{n-1} x'_i \alpha^{-ik} \bmod q \\
&= n^{-1} \sum_{i=0}^{n-1} \Big(\sum_{j=0}^{n-1} x_j \alpha^{ij}\Big) \alpha^{-ik} \bmod q \\
&= \sum_{j=0}^{n-1} x_j \big(n^{-1} \sum_{i=0}^{n-1} \alpha^{i(j-k)}\big) \bmod q \\
&= \sum_{j=0}^{n-1} x_j \delta_{k,j} \bmod q \\
&= x_k.
\end{aligned}$$

Note that $\delta_{k,j} = n^{-1} \sum_{j=0}^{n-1} \alpha^{i(j-k)} \bmod q$ is the Kronecker delta, defined by $\delta_{k,j} = 0$ if $k \neq j$ and $\delta_{k,j} = 1$ if $k = j$. ■

We denote the application of the inverse DFT by $\{x_k\}_{k=0}^{n-1} = \text{DFT}^{-1}(\{x'_k\}_{k=0}^{n-1})$.

Viewing this from a different perspective, the DFT is a linear transformation whose associated matrix is $A_{ij} = \alpha^{ij} \bmod q$. Since α is a primitive n-root of unity it turns out that A is a Vandermonde matrix and therefore we can find the respective inverse matrix, showing that the DFT is an invertible transformation.

From this point on, we work with elements in the finite field $\mathbb{Z}_q$ in which an α is a primitive root of n-th root of unity, so $n \mid q - 1$.

3.2.3 Properties of the DFT

There are various useful properties of the DFT. As mentioned earlier some processing on the transformed domain produces some effect in the original domain. In the following we state and prove some of these properties. The DFT is attractive because most of these properties have as common feature that performing an operation in the transformed domain realizes a desired processing in the original domain in a more convenient way than performing the processing directly over the original data domain.

In the following, we review some of the basic properties of the DFT.

Proposition 3.3 (Periodicity) *For every integer ℓ it holds that $x'_{k+\ell n} = x'_{k'}$ for all $0 \leq k < n$.*

This property suggest that we can evaluate the transform for all the integers k instead of only for $0 \leq k < n$. Its validity follows from the definition of the DFT and the periodicity property of the root of unity α.

$$x'_{k+\ell n} = \sum_{j=0}^{n-1} x_j \alpha^{(k+\ell n)j} = \sum_{j=0}^{n-1} x_j \alpha^{kj} \underbrace{\alpha^{\ell n j}}_{=1} = x'_k.$$

In other words, the DFT w.r.t. an n-th root of unity has period n. An interpretation of this property is that the indices can be reduced modulo n.

Proposition 3.4 *Let $\{x_k\}_{k=0}^{n-1}$ and $\{y_k\}_{k=0}^{n-1}$ be two lists of n elements in $\mathbb{Z}_q$. Let r be an integer $0 \leq r < n$. Then $y_{k+r} = x_k$ if and only if $y'_k = \alpha^{rk} x'_k \bmod q$.*

Proof. We verify this property using the definitions.

$$\begin{aligned} y'_k &= \sum_{j=0}^{n-1} y_j \alpha^{kj} \\ &= \sum_{j=0}^{n-1} x_{j-r} \alpha^{kj} \\ &= \sum_{j=0}^{n-1} x_j \alpha^{k(j+r)} \\ &= \alpha^{rk} x'_k = \beta^k x'_k, \end{aligned}$$

where $\beta = \alpha^r$. ■

This says that two lists of values are a rotation of the other in the time domain if and only if there is an element-wise rescaling in the frequency domain. The element-wise rescaling factors are well defined and uniquely determined by the rotation offset r applied.

Hence, first apply DFT to a list $\{x_k\}_{k=0}^{n-1}$ yielding $\{x'_k\}_{k=0}^{n-1}$, then compute $\{y'_k\}_{k=0}^{n-1}$ by setting

$$y'_k = \beta^k x'_k = \alpha^{rk} x'_k, \tag{3.3}$$

for $0 \leq k < n$, and finally apply inverse DFT to obtain the list $\{y_k\}_{k=0}^{n-1}$. Then, it follows that $y_{k+r} = x_k$ and thus, the two lists are a rotation of each other.

We use this observation to efficiently achieve rotations of lists of homomorphic encryptions. It allows us to perform both a rotation and prove in zero-knowledge that this is the case, as we see later in this chapter.

We continue describing some properties of the DFT. The following one applies to finite convolutions, used later on in this chapter.

Definition 3.5 (Circular Convolution) *The* convolution *of two lists $\{x_k\}_{k=0}^{m-1}$ and $\{y_k\}_{k=0}^{m-1}$ of m elements in $\mathbb{Z}_q$ is a new list $\{z_k\}_{k=0}^{m-1}$ such that*

$$z_k = \sum_{j=0}^{m-1} x_j y_{k-j},$$

where the indices wrap-around cyclically.

We denote the circular convolution with $\{z_k\}_{k=0}^{m-1} = \text{CONV}(\{x_k\}_{k=0}^{m-1}, \{y_k\}_{k=0}^{m-1})$.

The DFT allows the computation of convolutions by going to the transformed domain and multiplying the the lists element-wise as stated in the following proposition.

Proposition 3.6 *Given two lists* $\{x_k\}_{k=0}^{n-1}$ *and* $\{y_k\}_{k=0}^{n-1}$. *Let* $\{x'_k\}_{k=0}^{n-1} = \text{DFT}(\{x_k\}_{k=0}^{n-1})$ *and* $\{y'_k\}_{k=0}^{n-1} = \text{DFT}(\{y'_k\}_{k=0}^{n-1})$. *Then,*

$$\text{CONV}(\{x_k\}_{k=0}^{n-1}, \{y_k\}_{k=0}^{n-1}) = \text{DFT}^{-1}(\{x'_k y'_k\}_{k=0}^{n-1}).$$

Proof. The proof follows from the definitions. Let $\{z_k\}_{k=0}^{n-1} = \text{DFT}^{-1}(\{x'_k y'_k\}_{k=0}^{n-1})$. Then,

$$\begin{aligned}
z_k &= n^{-1} \sum_{i=0}^{n-1} x'_i y'_i \alpha^{-ik} \\
&= n^{-1} \sum_{i=0}^{n-1} \Big(\sum_{j_1=0}^{n-1} x_{j_1} \alpha^{ij_1}\Big)\Big(\sum_{j_2=0}^{n-1} y_{j_2} \alpha^{ij_2}\Big)\alpha^{-ik} \\
&= \sum_{j_1=0}^{n-1} x_{j_1} \sum_{j_2=0}^{n-1} y_{j_2} \big(n^{-1} \sum_{i=0}^{n-1} \alpha^{i(j_1+j_2-k)}\big) \\
&= \sum_{j_1=0}^{n-1} x_{j_1} \sum_{j_2=0}^{n-1} y_{j_2} \delta_{j_2, k-j_1} \\
&= \sum_{j_1=0}^{n-1} x_{j_1} y_{k-j_1}.
\end{aligned}$$

The value $\delta_{i,j}$ denotes the Kronecker delta. ■

3.2.4 Rotation of Homomorphic Encryptions using DFT

We show in the following how to perform DFT of encrypted values and how to use the properties of DFT to rotate homomorphic encryptions. Since we will use the DFT of encrypted values, we demand a homomorphic cryptosystem that has $\mathbb{Z}_q$, with q prime as the message space. Moreover, we consider lists of n elements with $n \mid q-1$ such that a primitive n-th root of unity exists.

Algorithm 3.1 describes how to rotate a list of encryptions n by an offset r using a DFT w.r.t. an n-th root of unity α. The DFT and inverse DFT can be performed by just using homomorphic properties, since both transformations are a linear combination of the encrypted values on known coefficients. This is done in Steps 1 and 3 which can be performed by anyone publicly.

Note that the actual plaintexts need not be known, but it follows from Proposition 3.4 that there is a rotation of the plaintexts of the input and output encrypted lists. This is done in Step 2, by using the property that rescaling in the transformed domain translates into a rotation in the original domain. The rescaling can also be done using homomorphic properties only. However, since our goal is to hide the rotation offset used, a random encryption of 0 is used as a blinding factor.

Therefore, the lists of plaintexts of the input and output ciphertexts are a rotation of each other by an offset of r positions. In Algorithm 3.1, moreover, this offset is kept secret.

Algorithm 3.1 Rotating Homomorphic Encryptions using DFT

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$, rotation offset r
Output: $\{[\![y_k]\!]\}_{k=0}^{n-1}$ such that $y_{k+r} = x_k$.

Step 1 (Apply DFT). Using homomorphic properties compute:

for $k = 0$ to $n-1$ **do**
  $[\![x'_k]\!] = \prod_{j=0}^{n-1} [\![x_j]\!]^{\alpha^{kj}}$
end for

Step 2 (Rotation).

for $k = 0$ to $n-1$ **do**
  pick $s_k \in_R \mathcal{R}$
  $[\![y'_k]\!] = [\![x'_k]\!]^{\alpha^{rk}} \mathsf{E}(0, s_k)$
end for

Step 3 (Apply inverse DFT). Using homomorphic properties compute

for $k = 0$ to $n-1$ **do**
  $[\![y_k]\!] = \left(\prod_{i=0}^{n-1} [\![y'_i]\!]^{\alpha^{-ik}}\right)^{n^{-1}}$
end for
return $\{[\![y_k]\!]\}_{k=0}^{n-1}$

3.2.5 Proof of Rotation using DFT

The method to rotate lists of encryptions given in Algorithm 3.1 is fine if we consider an honest rotator who wants to hide the rotation offset used. To be covered against a possibly malicious rotator we must provide a zero-knowledge proof of knowledge that Phase 2 of the algorithm is performed properly. In other words, we want a zero-knowledge proof of knowledge for the following relation:

$$R_{\text{DFT}} = \{(\{[\![x'_k]\!]\}_{k=0}^{n-1}, \{[\![y'_k]\!]\}_{k=0}^{n-1}; r, \{s_k\}_{k=0}^{n-1}) : [\![y'_k]\!] = [\![x'_k]\!]^{\alpha^{rk}} \mathsf{E}(0, s_k), \text{ for } 0 \le k < n\}.$$

Defining $\beta = \alpha^r$ we get that $[\![y'_k]\!] = [\![x'_k]\!]^{\beta^k} \mathsf{E}(0, s_k)$ and thus, we redefine R_{DFT} as follows.

$$R_{\text{DFT}} = \{(\{[\![x'_k]\!]\}_{k=0}^{n-1}, \{[\![y'_k]\!]\}_{k=0}^{n-1}; \beta, \{s_k\}_{k=0}^{n-1}) : [\![y'_k]\!] = [\![x'_k]\!]^{\beta^k} \mathsf{E}(0, s_k), \text{ for } 0 \le k < n, \beta^n = 1\}.$$

Observe that $\beta^n = 1$ if and only if there exists r, $0 \le r < n$ such that $\alpha^r = \beta$ because α is an element of order n.

We sketch in the following how to get a Σ-protocol to prove in zero-knowledge the relation R_{DFT}. We decompose the relation into simpler relations for which we know that there is an efficient Σ-protocol. Note that proving that $[\![y'_k]\!] = [\![x'_k]\!]^{\beta^k} \mathsf{E}(0, s_k)$ means that the scalar β^k is multiplied into the plaintext of $[\![x'_k]\!]$ like in the private multiplier relation $R_{\text{PRV-MLT}}$ (see Section 2.2.3). Auxiliary encryptions of β^k may be required.

In fact, we decompose the zero-knowledge proof of relation R_{DFT} into two parts. First, the rotator generates auxiliary encryptions to all powers of some β. With these, the rotator proves that they are all powers of some β and also proves that $\beta^n = 1$. Secondly, those

auxiliary encryptions are employed to prove that $[\![y'_k]\!] = [\![x'_k]\!]^{\beta^k}\mathsf{E}(0, s_k)$ using the private multiplier relation $R_{\text{PRV-MLT}}$.

For the first part, the rotator gives the encryption $c_1 = \mathsf{E}(\beta, t_1)$ and defines $c_{k+1} = c_k^{\beta}\mathsf{E}(0, t_{k+1})$ for all $1 \leq k < n$. By construction it follows that $c_k = [\![\beta^k]\!]$. This way of generating random encryptions of the powers of β can be proved to be correct using a private multiplier relation since for all $1 \leq k < n$ it holds that $(c_k, c_{k+1}, c_1; \beta, t_{k+1}, t_1) \in R_{\text{PRV-MLT}}$. Moreover, using a relation of known plaintext, R_{KNOWN}, it can be shown that $c_n = \mathsf{E}(1, t_n^*)$ where $t_n^* = \sum_{j=0}^{n} \beta^{n-j} t_j$. This shows that $\beta^n = 1$.

In the second part, a proof that $[\![y'_k]\!] = [\![x'_k]\!]^{\beta^k}\mathsf{E}(0, s_k)$ is given using a private multiplier. With the encryptions $c_k = \mathsf{E}(\beta^k, t_k^*)$, the rotator gives a proof that the powers of β are used correctly, since $([\![x'_k]\!], [\![y'_k]\!], c_k; \beta^k, s_k, t_k^*) \in R_{\text{PRV-MLT}}$. It can be easily checked that $t_k^* = \sum_{j=0}^{k} \beta^{k-j} t_j$.

Overall, using the appropriate compositions of Σ-protocols, we can construct a Σ-protocol for the relation R_{DFT}.

3.2.6 Proof of Rotation of ElGamal Encryptions

In this section we present a Σ-protocol for the relation R_{DFT} using homomorphic ElGamal, including some optimizations. More concretely, given $[\![x'_k]\!] = (a_k, b_k)$ and $[\![y'_k]\!] = (d_k, e_k)$, for $0 \leq k < n$, the rotator has to prove that it knows $\beta \in \mathbb{Z}_q$, such that $\beta^n = 1$, and randomizers $s_k \in \mathbb{Z}_q$, such that

$$d_k = a_k^{\beta^k} g^{s_k}, e_k = b_k^{\beta^k} h^{s_k},$$

for all $0 \leq k < n$.

We basically follow the decomposition given in the previous section of two private multipliers. One particular optimization is applied in the private multipliers of homomorphic ElGamal. Namely, we use Pedersen commitments for the private multiplier instead of an encryption, see Section 2.2.3.

Accordingly, the prover produces auxiliary Pedersen commitments to all n powers of β by defining $c_1 = \mathsf{C}(\beta) = g^{\beta}\widetilde{h}^{t_1}$, and then $c_{k+1} = c_k^{\beta}\widetilde{h}^{t_{k+1}} = \mathsf{C}(\beta^{k+1})$ for all $1 \leq k < n$. Clearly, two successive commitments satisfy a multiplicative relation, since $(c_1, c_k, c_{k+1}; \beta, \beta^k, t_1, t_k^*, t_{k+1}^*) \in R_{\text{MULT}}$. To prove that $\beta^n = 1$ holds, observe that $c_n = \mathsf{C}(1, t_n^*)$ as well, then $(c_n, 1; t_n^*) \in R_{\text{KNOWN}}$. As before, $t_k^* = \sum_{j=0}^{k} \beta^{k-j} t_j$. Finally, a relation of private multiplier is satisfied between $[\![x'_k]\!]$, $[\![y'_k]\!]$ and c_k, since $([\![x'_k]\!], [\![y'_k]\!], c_k; \beta^k, s_k, t_k^*) \in R_{\text{PRV-MLT}}$.[1]

Protocol 3.1 is the Σ-protocol resulting from the composition of all relations described above.

3.2.7 Cascade of Rotators

As depicted in Algorithm 3.1, a verifiable rotation using DFT takes input list $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and converts it to the transformed domain (Step 1). The rotation is executed in the trans-

[1] We note an abuse of notation with respect to the definitions of relations R_{MULT}, R_{KNOWN} and $R_{\text{PRV-MLT}}$ given in Section 2.2.3. Those relations were introduced over encryptions and not over commitments as done in this section. However, we note that realizing such proofs over commitments in a discrete log setting is quite straightforward.

Protocol 3.1 Honest verifier zero-knowledge proof of knowledge for relation R_{DFT} for homomorphic ElGamal cryptosystem

Common Input : $\{(a_k, b_k)\}_{k=0}^{n-1}, \{(d_k, e_k)\}_{k=0}^{n-1}$
Prover's Private Input : $\beta, \{s_k\}_{k=0}^{n-1}$

Prover		**Verifier**
$\widetilde{m}, b \in_R \mathbb{Z}_q$		
$\widetilde{C} = \widetilde{h}^{\widetilde{m}}$		
$m_k, t_k, u_k, v_k, \widetilde{m}_k \in_R \mathbb{Z}_q$		
$c_{k+1} = c_k^{\beta} \widetilde{h}^{t_k}$		
$C_{k+1} = c_k^{b} \widetilde{h}^{m_k}$		
$\widetilde{C}_k = g^{u_k} \widetilde{h}^{\widetilde{m}_k}$		
$D_k = a_k^{u_k} g^{v_k}$		
$E_k = b_k^{u_k} h^{v_k}$		
	$\xrightarrow{\widetilde{C}, \{c_{k+1}, C_{k+1}, D_k, E_k, \widetilde{C}_k\}_{k=0}^{n-1}}$	
	$\xleftarrow{\lambda}$	$\lambda \in_R \{0,1\}^k$
$t_{-1}^* = 0$		
$t_k^* = \sum_{j=0}^{k} \beta^{k-j} t_j$		
$\sigma = b + \lambda\beta,$		
$\eta = \widetilde{m} + \lambda t_{n-1}^*$		
$\psi_k = m_k + \lambda t_k$		
$\mu_k = u_k + \lambda\beta^k$		
$\nu_k = v_k + \lambda s_k$		
$\rho_k = \widetilde{m}_k + \lambda t_{k-1}^*$		
	$\xrightarrow{\sigma, \eta, \{\psi_k, \mu_k, \nu_k, \rho_k\}_{k=0}^{n-1}}$	$\widetilde{h}^{\eta} \stackrel{?}{=} \widetilde{C}(c_n/g)^{\lambda}$
		$c_k^{\sigma} \widetilde{h}^{\psi_k} \stackrel{?}{=} C_{k+1} c_{k+1}^{\lambda}$
		$g^{\mu_k} \widetilde{h}^{\rho_k} \stackrel{?}{=} \widetilde{C}_k c_k^{\lambda}$
		$a_k^{\mu_k} g^{\nu_k} \stackrel{?}{=} D_k d_k^{\lambda}$
		$b_k^{\mu_k} h^{\nu_k} \stackrel{?}{=} E_k e_k^{\lambda}$

formed domain (Step 2). Finally the encryptions are brought back to the original domain using inverse DFT (Step 3). Note that Step 1 must also be executed for the verifiers of the proof given by a rotator. Step 3, on the other hand, has to be performed by the intended receivers of the rotated list. Even though Steps 1 and 3 consist of public homomorphic properties, they represent a computational bottleneck $O(n^2)$ computation is needed to complete those steps. Verifiability of the whole rotation process relies on the proof of knowledge for relation R_{DFT} after Step 2 is performed.

In the context of a cascade of rotators, it naively means that every time a rotation needs to be performed, these steps must be executed in sequence. That is, Steps 1, 2 and 3 for one rotation, and again Steps 1, 2 and 3 for the next rotation, and so on. However, note that once the input list of encryptions to the cascade is converted to the transformed domain (Step 1), all rotations can be performed in sequence *without* need of switching domains. After all rotators do their job (in the transformed domain), the transformed is

Protocol 3.2 Cascade of Rotators using DFT

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$
Output: $\{[\![y_k]\!]\}_{k=0}^{n-1}$ such that $y_{k+r} = x_k$.

Step 1 (Apply DFT). Convert input encrypted list using the DFT.

for $k = 0$ to $n-1$ **do**
  $[\![x'^{(0)}_k]\!] = \prod_{j=0}^{n-1} [\![x_j]\!]^{\alpha^{kj}}$
end for

Step 2 (Cascade of Rotations). Rotators take turns to rotate.

rename $\{[\![x^{(0)}_k]\!]\}_{k=0}^{n-1} = \{[\![x_k]\!]\}_{k=0}^{n-1}$
for $i = 1$ to m **do**
  i-th rotator does the following:
  pick $r_i \in_R \{0, \ldots, n-1\}$
  for $k = 0$ to $m-1$ **do**
    pick $s^{(i)}_k \in_R \mathcal{R}$
    $[\![x'^{(i)}_k]\!] = [\![x'^{(i-1)}_k]\!]^{\alpha^{r_i k}} \mathsf{E}(0, s^{(i)}_k)$
  end for
end for

Step 3 (Apply inverse DFT). Convert final list of encryptions using the inverse DFT.

for $k = 0$ to $n-1$ **do**
  $[\![y_k]\!] = \left(\prod_{i=0}^{n-1} [\![x'^{(m)}_i]\!]^{\alpha^{-ik}}\right)^{n^{-1}}$
end for
return $\{[\![y_k]\!]\}_{k=0}^{n-1}$

inverted only once at the end of the cascade (Step 3) to get the encryptions back to the original domain.

A rotation of a list of encryptions through a cascade of rotators is described in Protocol 3.2. It is assumed that a set of m rotators take turns to rotate a list of homomorphic encryptions. Steps 1 and 3 are identical to Steps 1 and 3 of Algorithm 3.1 for a single rotation. In Step 2 of Protocol 3.2, rotations are performed one after another in the cascade keeping the data in the transformed domain.

It is easy to see that the cascade works. Consider a list of encryptions $\{[\![x'_k]\!]\}_{k=0}^{n-1} = \text{DFT}(\{[\![x_k]\!]\}_{k=0}^{n-1})$ applying Step 1 of Algorithm 3.1. A rotator gives list $\{[\![y'_k]\!]\}_{k=0}^{n-1}$ by performing $[\![y'_k]\!] = [\![x'_k]\!]^{\alpha^{r_1 k}} \mathsf{E}(0, s_k)$ for randomizers s_k and rotation offset r_1. Next rotator in the cascade takes as input the list $\{[\![y'_k]\!]\}_{k=0}^{n-1}$ and produces the list $\{[\![z'_k]\!]\}_{k=0}^{n-1}$ defined as $[\![z'_k]\!] = [\![y'_k]\!]^{\alpha^{r_2 k}} \mathsf{E}(0, t_k)$ for randomizers t_k and rotation offset r_2.

Let $\{[\![z_k]\!]\}_{k=0}^{n-1} = \text{DFT}^{-1}(\{[\![z'_k]\!]\}_{k=0}^{n-1})$. Then $z_{k+r_1+r_2} = x_k$ for all $0 \leq k < n$. By construc-

tion, for all $0 \leq k < n$,

$$\begin{aligned}
[\![z'_k]\!] &= [\![y'_k]\!]^{\alpha^{r_2 k}} \mathsf{E}(0, t_k) \\
&= ([\![x'_k]\!]^{\alpha^{r_1 k}} \mathsf{E}(0, s_k))^{\alpha^{r_2 k}} \mathsf{E}(0, t_k) \\
&= [\![x'_k]\!]^{\alpha^{(r_1+r_2)k}} \mathsf{E}(0, \alpha^{r_2 k} s_k + t_k),
\end{aligned}$$

which by Algorithm 3.1 means that $z_{k+r_1+r_2} = x_k$.

3.2.8 Performance Analysis

It can be checked that Protocol 3.1 requires $O(n)$ exponentiations, with small constants. However, in view of Algorithm 3.1, this proof of knowledge requires precomputations and postcomputations (i.e., Steps 1 and 3). Even though these steps can be done publicly, we note that they require $O(n^2)$ exponentiations in the case of homomorphic ElGamal. This is because computing the DFT and the inverse DFT of encrypted values requires the computation of n times n-way exponentiations which involve $O(n^2)$ exponentiations.

In Section 3.3 we show that under certain conditions one can take advantage of particular algorithms to compute the DFT efficiently, known as Fast Fourier Transforms (FFT for short). These algorithms allow a reduction of the computation of the DFT and the inverse DFT of encrypted values to $O(n \log n)$ exponentiations. This means an overall $O(n \log n)$ computation to carry out a rotation using Algorithm 3.1.

We observe that in a cascade of rotators we can amortize the computation per rotator. Assuming that an efficient FFT algorithm is possible, one can average the computational work of the whole cascade. As the FFT needs to be applied in the beginning and possibly at the end of the cascade, then when the length of the cascade is $\Omega(\log n)$ the overall average work per rotator is reduced to $O(n)$.

3.3 Fast Fourier Transform

An issue in the efficiency of the approach for rotation based on DFT is the computation of the DFT and the inverse DFT of encrypted values. Although they can be applied just using homomorphic operations on the ciphertexts, these steps may be a computational bottleneck as they involve the computation of n times n-way exponentiations for the case of ElGamal encryptions. This costs $O(n^2)$ computations.

This is an issue inherent to the DFT in general. There are several approaches to reduce the computational cost depending on the parameters of the DFT. These techniques, referred to as the Fast Fourier Transform (FFT), are algorithms that take advantage of the symmetry and periodicity of roots of unity. Although all these algorithms were developed for Fourier transforms with data on the complex field, they all only rely on basic properties of the root of unity.

FFT algorithms seem to be transferable to transforms over finite fields but this is not always the case. Basically, FFT algorithms over the field of the complex numbers rely on the fact that n-th roots of unity for every integer n exist. In the field $\mathbb{Z}_q$, however, n-th roots of unity only exist for the case in that $n \mid q-1$. In particular, if α is a n-th root of unity, α^d is an n/d-th root of unity modulo q, for every integer d such that $d \mid n$.

Most FFT algorithms decompose the computation of the DFT by making use of the factorization of n. They use the roots of unity for different divisors of n. Asymptotically

speaking, they reduce the computation of the DFT from $O(n^2)$ operations to $O(n \log n)$ operations.

3.3.1 Cooley-Tukey FFT Algorithm

The most popular and easy to understand FFT algorithm is the radix-2 Cooley-Tukey algorithm [CT65] applicable for the case that n is even. Below, we show how this algorithm reduces the computation of the DFT of encrypted values from n^2 to $2(n/2)^2$ exponentiations, saving a factor 2 in the number of operations. If n is an integral power of 2, the algorithm can be applied recursively reducing the overall computation to proportional to $n \log n$.

Case n is Even

Algorithm 3.2 illustrates the application of the radix-2 Cooley-Tukey FFT algorithm. In the following we explain how it works. Suppose that α is an n-th root of unity modulo q with n even. Then the DFT of encrypted values can be splitted in the following way:

$$\begin{aligned}
[\![x'_k]\!] &= \prod_{j=0}^{n-1} [\![x_j]\!]^{\alpha^{kj}} \\
&= \prod_{j=0}^{n/2-1} [\![x_{2j}]\!]^{\alpha^{k2j}} [\![x_{2j+1}]\!]^{\alpha^{k(2j+1)}} \\
&= \prod_{j=0}^{n/2-1} [\![x_{2j}]\!]^{(\alpha^2)^{kj}} \left(\prod_{j=0}^{n/2-1} [\![x_{2j+1}]\!]^{(\alpha^2)^{kj}} \right)^{\alpha^k}.
\end{aligned}$$

We define $[\![e'_k]\!] = \prod_{j=0}^{n/2-1} [\![e_j]\!]^{\alpha^{2kj}}$ and $[\![o'_k]\!] = \prod_{j=0}^{n/2-1} [\![o_j]\!]^{\alpha^{2kj}}$, where $[\![e_k]\!] = [\![x_{2k}]\!]$ and $[\![o_k]\!] = [\![x_{2k+1}]\!]$ for $0 \leq k < n/2$. The lists $\{[\![e'_k]\!]\}_{k=0}^{n/2}$ and $\{[\![o'_k]\!]\}_{k=0}^{n/2}$ are both valid DFTs of encrypted values w.r.t. α^2 of the lists $\{[\![e_k]\!]\}_{k=0}^{n/2}$ and $\{[\![o_k]\!]\}_{k=0}^{n/2}$, respectively. Note that α^2 is an $n/2$-th root of unity.

Since lists $\{[\![e'_k]\!]\}_{k=0}^{n/2}$ and $\{[\![o'_k]\!]\}_{k=0}^{n/2}$ are DFTs w.r.t. an $n/2$-th root of unity, they have period $n/2$ and thus,

$$\begin{aligned}
[\![x'_k]\!] &= [\![e'_k]\!][\![o'_k]\!]^{\alpha^k} \text{ for } 0 \leq k < n/2, \\
[\![x'_{k+n/2}]\!] &= [\![e'_k]\!][\![o'_k]\!]^{-\alpha^k} \text{ for } 0 \leq k < n/2.
\end{aligned}$$

This way, we have reduced the problem of the computation of a DFT on a list of length n to solving two DFTs of lists of half the length plus one extra exponentiation.[2] We note that the two half-length DFTs can be computed in parallel because they are independent of each other. Therefore, the total number of exponentiations has been reduced from about n^2 to about $2(n/2)^2 + n/2$ which asymptotically means an improvement by a factor of 2.

When n is an integral power of 2, we can apply this method in a recursive manner, splitting the list in even and odd elements. We get that the number of exponentiations is $\log n$ for half of the elements in the transformed list, yielding a total of $(n/2) \log n$ exponentiations overall.

[2] More precisely, once $[\![o_k]\!]^{\alpha^k}$ has been computed, $[\![o_k]\!]^{-\alpha^k}$ may be obtained by computing some modular inversions.

Algorithm 3.2 DFT of encrypted values using Cooley-Tukey FFT for n even

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$, $n = 2\ell$
Output: $\{[\![x'_k]\!]\}_{k=0}^{n-1}$ such that $\{[\![x'_k]\!]\}_{k=0}^{n-1} = \{[\![x_k]\!]\}_{k=0}^{n-1}$
  for $k = 0$ to $n/2 - 1$ **do**
    $[\![e_k]\!] = [\![x_{2k}]\!]$
    $[\![o_k]\!] = [\![x_{2k+1}]\!]$
  end for
  compute
    $\{[\![e'_k]\!]\}_{k=0}^{n/2-1} = \mathrm{DFT}(\{[\![e_k]\!]\}_{k=0}^{n/2-1})$
    $\{[\![o'_k]\!]\}_{k=0}^{n/2-1} = \mathrm{DFT}(\{[\![o_k]\!]\}_{k=0}^{n/2-1})$
  for $k = 0$ to $n/2 - 1$ **do**
    $[\![x'_k]\!] = [\![e'_k]\!][\![o'_k]\!]^{\alpha^k}$
    $[\![x'_{k+n/2}]\!] = [\![e'_k]\!][\![o'_k]\!]^{-\alpha^k}$
  end for
  return $\{[\![x'_k]\!]\}_{k=0}^{n-1}$

General Composite n

The general Cooley-Tukey decomposition of the computation, for $n = n_1 n_2$ with $n_1, n_2 > 1$, uses $k = n_2 k_1 + k_2$ and $j = n_1 j_2 + j_1$ for indices $0 \leq k_1, j_1 < n_1$ and $0 \leq k_2, j_2 < n_2$

$$\begin{aligned}
[\![x'_{n_2 k_1 + k_2}]\!] &= \prod_{j_1=0}^{n_1-1} \prod_{j_2=0}^{n_2-1} [\![x_{n_1 j_2 + j_1}]\!]^{\alpha^{(n_2 k_1 + k_2)(n_1 j_2 + j_1)}} \\
&= \prod_{j_1=0}^{n_1-1} \left[\left(\prod_{j_2=0}^{n_2-1} [\![x_{n_1 j_2 + j_1}]\!]^{\alpha^{n_1 j_2 k_2}} \right)^{\alpha^{j_1 k_2}} \right]^{\alpha^{n_2 j_1 k_1}} .
\end{aligned}$$

As α is an n-th root of unity modulo q, and $n = n_1 n_2$, it follows that α^{n_1} is an n_2-th root of unity. Therefore, all interior products (indexed by j_2) are DFTs of lists of n_2 elements. There are n_1 in total. These intermediate transforms are later raised to some constants known as *twiddle factors*. The new list is combined with the outer product which represents n_2 DFTs of length n_1. Hence, we have reduced the problem of a DFT of length $n = n_1 n_2$ into a problem of n_1 DFTs of length n_2 and n_2 DFTs of length n_1 plus n exponentiations. All this roughly amounts to $n_1 {n_2}^2 + {n_1}^2 n_2 + n$ exponentiations, meaning a reduction from $n_1^2 n_2^2$ to $n_1 {n_2}^2 + {n_1}^2 n_2$ exponentiations. Algorithm 3.3 summarizes this method.

3.3.2 Bluestein's FFT Algorithm

Unfortunately, Cooley-Tukey's technique does not consider the case when n is prime. For the complex field there exist other FFT algorithms that reduce the computation of a DFT of prime length n to the computation of other DFTs of composite length, allowing the application of, for example, the above mentioned FFT algorithms. In the following, we sketch Bluestein's FFT algorithm [Blu70] that works in the case that n is odd.

Algorithm 3.3 DFT of encrypted values using General Cooley-Tukey FFT

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$, $n = n_1 n_2$
Output: $\{[\![x'_k]\!]\}_{k=0}^{n-1}$ such that $\{[\![x'_k]\!]\}_{k=0}^{n-1} = \{[\![x_k]\!]\}_{k=0}^{n-1}$
  for $j_1 = 0$ to $n_1 - 1$ **do**
    for $j_2 = 0$ to $n_2 - 1$ **do**
      $[\![y_{j_1,j_2}]\!] = [\![x_{n_1 j_2 + j_1}]\!]$
    end for
    compute $\{[\![y'_{j_1,j_2}]\!]\}_{j_2=0}^{n_2-1} = \mathrm{DFT}(\{[\![y_{j_1,j_2}]\!]\}_{j_2=0}^{n_2-1})$
  end for
  for $k_2 = 0$ to $n_2 - 1$ **do**
    for $j_1 = 0$ to $n_2 - 1$ **do**
      $[\![\hat{y}_{j_1,k_2}]\!] = [\![y_{j_1,k_2}]\!]^{\alpha^{j_1 k_2}}$ $\rightsquigarrow$ These are the *twiddle* factors.
    end for
    compute $\{[\![\hat{x}_{j_1,k_2}]\!]\}_{j_1=0}^{n_1-1} = \mathrm{DFT}(\{[\![\hat{y}_{j_1,k_2}]\!]\}_{j_1=0}^{n_1-1})$
  end for
  for $k_1 = 0$ to $n_1 - 1$ **do**
    for $k_2 = 0$ to $n_2 - 1$ **do**
      $[\![x'_{n_2 k_1 + k_2}]\!] = [\![\hat{x}_{k_1,k_2}]\!]$
    end for
  end for
  return $\{[\![x'_k]\!]\}_{k=0}^{n-1}$

Convolutions of Homomorphic Encryptions

For later use in the exposition of Bluestein's algorithm for FFT adapted to the case of homomorphic encryptions, we explain how to compute convolutions of encrypted values using properties of the DFT.

Suppose we have a list of homomorphic encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and a list of *known* values $\{a_k\}_{k=0}^{n-1}$ all in $\mathbb{Z}_q$. Encryptions of the convolution of the lists $\{x_k\}_{k=0}^{n-1}$ and $\{a_k\}_{k=0}^{n-1}$ can be computed using homomorphic properties as follows:

$$[\![z_k]\!] = \prod_{j=0}^{n-1} [\![x_j]\!]^{a_{k-j}}, \tag{3.4}$$

where k ranges from 0 to $n-1$. The straightforward way of computing a convolution clearly requires $O(n^2)$ exponentiations.

Algorithm 3.4 gives a more efficient way of computing convolutions of encrypted values. Its correctness is based on Prop. 3.6. The overall number of exponentiations reduces to the work required for the two DFTs under encryptions and n exponentiations for the intermediate point-wise multiplication. If n is (highly) composite we can take advantage of the Cooley-Tukey FFT of the two DFTs needed. In particular, if n is an integral power of two, get that the number of exponentiations is proportional to $n \log n$.

Parameter Requirements

At a high-level, Bluestein's FFT algorithm reduces the computation of the DFT of a list of odd length n to the computation of two DFTs of length $m \geq 2n - 1$. Obviously, the

Algorithm 3.4 Computing convolutions of encrypted values

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}, \{a_k\}_{k=0}^{n-1}$
Output: $\{[\![z_k]\!]\}_{k=0}^{n-1}$ such that $\{[\![z_k]\!]\}_{k=0}^{n-1} = \text{CONV}(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{a_k\}_{k=0}^{n-1})$
  compute
    $\{[\![x'_k]\!]\}_{k=0}^{n-1} = \text{DFT}(\{[\![x_k]\!]\}_{k=0}^{n-1})$
    $\{a'_k\}_{k=0}^{n-1} = \text{DFT}(\{a_k\}_{k=0}^{n-1})$
  for $k = 0$ to $n-1$ **do**
    $[\![z'_k]\!] = [\![x'_k]\!]^{a'_k}$
  end for
  compute $\{[\![z_k]\!]\}_{k=0}^{n-1} = \text{DFT}^{-1}(\{[\![z'_k]\!]\}_{k=0}^{n-1})$
  return $\{[\![z_k]\!]\}_{k=0}^{n-1}$

computation of the DFT on lists of length m requires that an m-th root of unity exists, yielding the condition that $m \mid q-1$. Furthermore, m is required to be *highly* composite so that Cooley-Tukey's method gives us some computational savings. In particular, it is highly desired that m is an integral power of 2.

Hence, q, n and m should be such that n is odd, $n \mid q-1$, $m \geq 2n-1$ and $m \mid q-1$. Thus, it should hold that $q = \text{lcm}(m,n)\ell + 1$ for some integer ℓ. Since we compute two DFTs of length m, we may want to ensure that $m = 2^k$ in which case we request that $q = 2^k n\ell + 1$, for an integer k such that $2^k \geq 2n-1$. For a discussion on the feasibility of these parameters, see [BA01] and references therein.

The Algorithm

Let $\{[\![x_k]\!]\}_{k=0}^{n-1}$ be the list of encryptions for which we have to compute the DFT on. Consider the case that n is odd (in particular that n is prime). Then we have the following.

$$\begin{aligned}
[\![x'_k]\!] &= \prod_{j=0}^{n-1} [\![x_j]\!]^{\alpha^{kj}} \\
&= \prod_{j=0}^{n-1} [\![x_j]\!]^{\alpha^{-(k-j)^2/2 + k^2/2 + j^2/2}} \qquad (3.5) \\
&= \left(\prod_{j=0}^{n-1} ([\![x_j]\!]^{\alpha^{-j^2/2}})^{\alpha^{(k-j)^2/2}} \right)^{\alpha^{-k^2/2}} . \qquad (3.6)
\end{aligned}$$

We start by expanding the expression of the DFT of encrypted values. Then, in Eq. (3.5) the product kj can be written as $kj = -(k-j)^2/2 + k^2/2 + j^2/2$ which Eq. (3.6) yields by associating the exponents accordingly.

Defining $[\![y_j]\!] = [\![x_j]\!]^{\alpha^{-j^2/2}}$ and $a_j = \alpha^{j^2/2}$ we take $\{[\![z_k]\!]\}_{k=0}^{n-1} = \text{CONV}(\{[\![y_k]\!]\}_{k=0}^{n-1}, \{a_k\}_{k=0}^{n-1})$ as in Eq. (3.4). This merely says that Eq. (3.6) can be rewritten as follows

$$[\![x'_k]\!] = [\![z_k]\!]^{\alpha^{-k^2/2}}. \qquad (3.7)$$

Therefore, we have to compute the convolution of encrypted values between $\{[\![y_k]\!]\}_{k=0}^{n-1}$ and $\{a_k\}_{k=0}^{n-1}$. It is worth noticing that $\alpha^{1/2}$ exists only if α has odd order.

Algorithm 3.5 DFT of encrypted values using Bluestein's FFT

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$, n is odd, $m \geq 2n-1$ with $m \mid q-1$.
Output: $\{[\![x'_k]\!]\}_{k=0}^{n-1}$ such that $\{[\![x'_k]\!]\}_{k=0}^{n-1} = \text{DFT}(\{[\![x_k]\!]\}_{k=0}^{n-1})$
  for $k = 0$ to $n-1$ **do**
    $[\![\hat{y}_k]\!] = [\![x_k]\!]^{-\alpha^2/2}$
    $\hat{a}_k = \alpha^{j^2/2}$
  end for
  for $k = n$ to $m-n-1$ **do**
    $[\![\hat{y}_k]\!] = \mathsf{E}(0)$
    $\hat{a}_k = 0$
  end for
  for $k = m-n$ to $m-1$ **do**
    $[\![\hat{y}_k]\!] = \mathsf{E}(0)$
    $\hat{a}_k = \alpha^{(m-k)^2/2}$
  end for
  compute $\{[\![z_k]\!]\}_{k=0}^{m-1} = \text{CONV}(\{[\![\hat{y}_k]\!]\}_{k=0}^{m-1}, \{\hat{a}_k\}_{k=0}^{m-1})$
  for $k = 0$ to $n-1$ **do**
    $[\![x'_k]\!] = [\![z_k]\!]^{\alpha^{-k^2/2}}$
  end for
  return $\{[\![x'_k]\!]\}_{k=0}^{n-1}$

If we apply Algorithm 3.4 at this stage, we would have traded the problem of computing the DFT of length n (our ultimate goal) to that of computing two DFTs of the same length which makes no sense. Instead, we trade the problem for that of computing a convolution on lists of length m for which a much more efficient FFT algorithm may be achieved. In order to do this, the lists $\{[\![y_k]\!]\}_{k=0}^{n-1}$ and $\{a_k\}_{k=0}^{n-1}$ are zero-padded in a special way such that we can compute the original convolution of length n via the new convolution of length m, where we additionally require that $m \geq 2n-1$.

The list $\{[\![y_k]\!]\}_{k=0}^{n-1}$ is zero-padded into $\{[\![\hat{y}_k]\!]\}_{k=0}^{m-1}$ in the usual way. Namely,

$$[\![\hat{y}_k]\!] = \begin{cases} [\![y_k]\!], & \text{if } 0 \leq k < n, \\ \mathsf{E}(0), & \text{if } n \leq k < m. \end{cases}$$

Recall that $\mathsf{E}(0)$ is a deterministic encryption of 0.

On the other hand, the list $\{a_k\}_{k=0}^{n-1}$ is padded into $\{\hat{a}_k\}_{k=0}^{m-1}$ differently.

$$\hat{a}_k = \begin{cases} a_k, & \text{if } 0 \leq k < n, \\ a_{m-k}, & \text{if } m-n \leq k < m, \\ 0, & \text{if } n \leq k < m-n. \end{cases}$$

The list $\{\hat{a}_k\}_{k=0}^{m-1}$ consists of a copy of the list $\{a_k\}_{k=0}^{m-1}$ at the beginning and in reversed order from the end. Clearly, the list $\hat{a}$ is well-defined if $m \geq 2n-1$.

Algorithm 3.4 can be used to efficiently compute $\{[\![\hat{z}_k]\!]\}_{k=0}^{m-1} = \text{CONV}(\{[\![\hat{y}_k]\!]\}_{k=0}^{m-1}, \{\hat{a}_k\}_{k=0}^{m-1})$. The correctness of Bluestein's algorithm is based on the following fact.

It is easy to see that $\{[\![\hat{z}_k]\!]\}_{k=0}^{n-1} = \{[\![z_k]\!]\}_{k=0}^{n-1}$. For all $0 \leq k < n$, we have:

$$\begin{aligned}
[\![\hat{z}_k]\!] &= \prod_{j=0}^{m-1} [\![\hat{y}_j]\!]^{\hat{a}_{k-j}} \\
&= \prod_{j=0}^{n-1} [\![\hat{y}_j]\!]^{\hat{a}_{k-j}} && (3.8) \\
&= \prod_{j=0}^{k} [\![y_j]\!]^{\hat{a}_{k-j}} \prod_{j=k+1}^{n-1} [\![y_j]\!]^{\hat{a}_{m-(j-k)}} && (3.9) \\
&= \prod_{j=0}^{k} [\![y_j]\!]^{a_{k-j}} \prod_{j=k+1}^{n-1} [\![y_j]\!]^{a_{j-k}} && (3.10) \\
&= \prod_{j=0}^{n-1} [\![y_j]\!]^{a_{k-j}} && (3.11) \\
&= [\![z_k]\!].
\end{aligned}$$

In Eq. (3.8) we "throw away" the tail of the product since $[\![\hat{y}_j]\!] = \mathsf{E}(0)$ for $k+1 \leq j < m$. For Eq. (3.9) note that $k+1 \leq j < m$ implies that $k - j \leq 0$ and since $j - k \leq m$ it holds that $\hat{a}_{k-j} = \hat{a}_{m-(j-k)}$. In Eq. (3.10) we use the definition of $\hat{a}$ observing that $m - n \leq m - (j-k) < m$. Finally, Eq. (3.11) follows from the fact that by definition $a_k = a_{-k}$.

Following this reasoning, we have that we can compute Eq. (3.7) efficiently if m is highly composite. All these steps are summarized in Algorithm 3.5. This way, one is able to compute efficiently a DFT efficiently for odd list lengths n, in particular when n is prime.

Chapter 4

General Verifiable Rotations

The DFT-based protocol presented in the previous chapter puts some constraints on the parameters involved. This is a consequence of the requirement of the existence of a root of unity. As a result, this limits the applicability of the approach. For example, if an adequate cryptosystem is already set up with message space $\mathbb{Z}_q$ for prime q, the length of the list of homomorphic encryptions to be rotated is restricted to all n for which an n-th root of unity modulo q exists. Of course, depending on the application some padding techniques or other tricks may be applied to the list that is going to be rotated. In some settings, if n is fixed, we have to set up a cryptosystem such that the plaintext is $\mathbb{Z}_q$ with q prime such that n is a divisor of $q-1$. Besides these two limitations, another drawback is that the application of the DFT using homomorphic properties requires some additional computations before and after the rotation takes place.

In this chapter we present a solution that does not put restrictions on the length n of the list to be rotated, once the cryptosystem is set up. In fact, this result applies to any homomorphic cryptosystem. We use a completely different approach which yields overall $O(n)$ computational complexity. Though the solution presented is not a standard Σ-protocol, we are able to show that the protocols are honest verifier zero-knowledge with witness-extended emulation.

4.1 Background

We start presenting the Schwartz-Zippel lemma as the core tool for the soundness of our protocols. This lemma is commonly used to test equality of polynomials probabilistically.

Lemma 4.1 (Schwartz-Zippel) *Let $p \in \mathbb{F}[x_1, \ldots, x_m]$ be a non-zero multivariate polynomial of degree d over a field $\mathbb{F}$ and let S be a finite, non-empty subset of $\mathbb{F}$ of at least d elements. Then $\mathbb{P}[p(z_1, \ldots, z_m) = 0] \leq d/|S|$, where all z_i, for $1 \leq i \leq m$, are randomly chosen from S.*

In other words, if a polynomial p evaluated at a random point from a finite set gives zero, it means that p is the zero polynomial except with some error probability. Clearly, checking equality of two polynomials s and t over a field $\mathbb{F}$ is equivalent to decide if the difference polynomial $p = s - t$ is the zero-polynomial.

The zero-polynomial test is formalized in the following proposition.

Proposition 4.2 *Let $p \in \mathbb{F}[x_1, \ldots, x_m]$ be a polynomial over $\mathbb{F}$ of degree at most d. Given S a finite non-empty subset of $\mathbb{F}$ with at least d elements, and $\beta_1, \ldots, \beta_m$ randomly selected from S.*

1. *If* $p = 0$ *then* $\mathbb{P}[p(\beta_1, \ldots, \beta_m) = 0] = 1$;
2. *If* $p \neq 0$ *then* $\mathbb{P}[p(\beta_1 \ldots, \beta_m) = 0] \leq d/|S|$.

Proof. (1) holds trivially. For (2) the Schwartz-Zippel lemma is applied to p since it has degree $k \leq d$. Hence $\mathbb{P}[p(\beta_1, \ldots, \beta_m) = 0] \leq k/|S| \leq d/|S|$. ■

The test gives a false positive (i.e., $p(\beta) = 0$ but $p \neq 0$) with probability at most $d/|S|$. To make the test accurate enough we either take $S \subset \mathbb{F}$ to be large enough or, when this is not possible, we repeat the test independently as many times as necessary to make the probability of failure very small.

Applications

As an application of this probabilistic test consider the problem of checking if a list of field elements consists of all zero elements. More specifically, given a list $\{z_k\}_{k=0}^{n-1}$ of n elements in the field $\mathbb{F}$, we want to check that $z_k = 0$ for all $0 \leq k < n$. For the Schwartz-Zippel test, let S be a finite subset of $\mathbb{F}$.

We define the polynomial $p(x_0, x_1, \ldots, x_{n-1}) = \sum_{j=0}^{n-1} z_j x_j$. Clearly, $p = 0$ if and only if $z_k = 0$ for all $0 \leq k < n$. Using the probabilistic test given by Prop. 4.2 we get that the failure probability of the test is upper-bounded by $1/|S|$ since p has degree 1.

Note that the polynomial may be defined differently. For example, consider the univariate polynomial $p(x) = \sum_{j=0}^{n-1} z_j x^j$. Once more, $z_k = 0$ for all $0 \leq k < n$ if and only if p is the zero-polynomial. Since p has degree at most $n - 1$, the failure probability is bounded above by $(n-1)/|S|$.

The all-zero list test can be used to check whether two lists $\{x_k\}_{k=1}^{n-1}$ and $\{y_k\}_{k=1}^{n-1}$ of n field elements are identical. It suffices to define $z_k = x_k - y_k$ for all $0 \leq k < n - 1$.

For the rest of this chapter, we use the all-zero test on the finite field $\mathbb{Z}_q$ with q prime. The set S will be chosen to be the whole space $\mathbb{Z}_q$.

4.2 Proof of Multiple Encryptions of 0

In this section we study a very related problem to the all-zero list test given above. Namely, we describe a protocol to prove in zero-knowledge that a list of homomorphic ciphertexts all are encryptions of 0. The soundness of this protocol is based on the Schwartz-Zippel lemma, and formally proved by showing a witness-extender emulator. This protocol and the ideas used in the witness-extended emulation are a stepping stone for the proof of rotation presented later in this chapter.

More concretely, this section treats an efficient proof of knowledge for the following relation:

$$R_{\text{B-ZERO}} = \{(\{[\![z_k]\!]\}_{k=0}^{n-1}; \{s_k\}_{k=0}^{n-1}) : [\![z_k]\!] = \mathsf{E}(0, s_k) \text{ for all } 0 \leq k < n\}.$$

A proof of knowledge for such a relation can be obtained by a Σ-protocol resulting of an AND-composition of Σ-protocols for R_{ZERO}. Namely, showing that $([\![z_k]\!]; s_k) \in R_{\text{ZERO}}$ for all $0 \leq k < n$, suffices to show that $(\{[\![z_k]\!]\}_{k=0}^{n-1}; \{s_k\}_{k=0}^{n-1}) \in R_{\text{B-ZERO}}$.

Such an approach requires 3 rounds of interaction and the computational work is n times that of a Σ-protocol for R_{ZERO}. That is, $O(n)$ computations.

Protocol 4.1 Honest verifier zero-knowledge proof of knowledge for relation $R_{\text{B-ZERO}}$

Common Input : $\{[\![z_k]\!]\}_{k=0}^{n-1}$
Prover's Private Input : $\{s_k\}_{k=0}^{n-1}$

Prover		**Verifier**
		$\beta \in_R \mathbb{Z}_q$
	$\xleftarrow{\quad\beta\quad}$	
$t = \sum_{j=0}^{n-1} \beta^j s_j$	Everyone computes: $[\![z]\!] = \prod_{j=0}^{n-1} [\![z_j]\!]^{\beta^j}$	
	Run Σ-protocol for $R_{\text{ZERO}}([\![z]\!]; t)$	

Protocol 4.1 gives a solution approaching the problem differently. It requires 4 rounds but the computational work of a *single* invocation of a Σ-protocol for R_{ZERO}. Soundness of this protocol relies on Prop. 4.2: it is like doing an all-zero list test under encrypted values.

Theorem 4.3 *Protocol 4.1 is a complete, honest verifier zero-knowledge proof of knowledge with witness-extended emulation for relation* $R_{\text{B-ZERO}}$.

Proof. We have to show that the protocol is complete, honest verifier zero-knowledge and it has witness-extended emulation. Completeness follows immediately by simple inspection. The remaining two properties are proved in the following.

Honest Verifier Zero-Knowledge. Algorithm 4.1 describes a simulator $\mathcal{S}_{\text{B-ZERO}}$ for Protocol 4.1 that produces conversation transcripts that are indistinguishable from the ones in an execution of the protocol assuming an honest verifier.

The simulator $\mathcal{S}_{\text{ZERO}}$ of the Σ-protocol for relation R_{ZERO} is used as a subroutine. Recall that as it is honest verifier zero-knowledge, the simulator $\mathcal{S}_{\text{ZERO}}$ produces a tuple $\mathtt{tr}_{\text{ZERO}} = (a, \lambda, u)$ statistically indistinguishable from a tuple representing a real world transcript. Using this fact, it is easy to conclude that $(\beta, \mathtt{tr}_{\text{ZERO}})$ is indistinguishable from a conversation transcript of Protocol 4.1.

Witness-Extended Emulation. Algorithm 4.2, denoted as $\mathcal{W}_{\text{B-ZERO}}$, is a witness-extended emulator for Protocol 4.1. It follows the structure of a generic witness-extended emulation. The prover is first run on random challenges and then the emulator attempts to extract a witness only if the first conversation is accepting.

For the extraction process, the emulation makes use of extractor $E_{\text{B-ZERO}}$ described in Algorithm 4.3. This extractor needs n witnesses of R_{ZERO} which are obtained by the special soundness extractor E_{ZERO} of its Σ-protocol. We note that extractor $E_{\text{B-ZERO}}$ only works under the condition that all β_i's are different.

We show that $\mathcal{W}_{\text{B-ZERO}}$ satisfies the sufficient conditions given in Lemma 2.6 for a witness-extended emulator. In fact, Lemma 2.6(ii) is satisfied straightforwardly since the algorithm always copies the result of the interaction with P^*. We now prove that $\mathcal{W}_{\text{B-ZERO}}$ outputs either $\perp$ or a valid witness. This enables us to validate Lemma 2.6(iii).

Algorithm 4.1 ($\mathcal{S}_{\text{B-ZERO}}$) Simulator for Protocol 4.1

Input: $\{[\![z_k]\!]\}_{k=0}^{n-1}$
  pick $\beta \in \mathbb{Z}_q$
  $[\![z]\!] = \prod_{j=0}^{n-1} [\![z_j]\!]^{\beta^j}$
  run $\mathtt{tr}_{\text{ZERO}} \leftarrow \mathcal{S}_{\text{ZERO}}([\![z]\!])$
  return $(\beta, \mathtt{tr}_{\text{ZERO}})$

Lemma 2.6(i) and (iv) are proved in Lemmas 4.5 and 4.6 respectively. ■

Before going into further details we define some notation and establish some conventions. Since P^* is a deterministic algorithm, we define ϵ to be the probability that $P^*(x, s)$ gives an accepting conversation when it runs the protocol on randomly chosen challenges (β, λ). This way, we can define a $q \times 2^k$ $\{0,1\}$-matrix S such that $S_{\beta,\lambda} = 1$ if and only if P^* gives an accepting conversation when executed on challenges β and λ. We denote with ϵ_β the probability that P^* gives an accepting proof on random λ, given that it has received β as initial challenge. By definition, ϵ is the proportion of 1's in S whereas ϵ_β is the proportion of 1's in S_β, the β-th row of S.

The specification of $\mathcal{W}_{\text{B-ZERO}}$ can be expressed in terms of matrix S. In the first steps (line 1 through 4), matrix S is probed at a random position (β_0, λ_0). If a 0 is hit, the algorithm halts copying the transcript of the protocol and there is no witness. If a 1 is hit then it will do the following. Row S_{β_0} is probed until a 1 is hit (lines 5 through 8). After this, the following is repeated $n-1$ times. The matrix is probed on random positions until a 1 is hit. This is done in the first **repeat** loop (lines 10 through 14). Then, row S_{β_i} is probed until a 1 is hit (lines 15 through 18).

Extractor $\mathcal{W}_{\text{B-ZERO}}$ knows positions (β_i, λ_i) and (β_i, λ_i') for which $S_{\beta_i,\lambda_i} = S_{\beta_i,\lambda_i'} = 1$, for $0 \leq i < n$. Note that some 1's might have hit the same row, and some rows might have been overlapped. Note that the order in which these random queries are executed does not matter as every time independently random challenges are drawn. The emulator is able to extract a witness when $\lambda_i \neq \lambda_i'$ for all $0 \leq i < n$ (i.e. there is no collusion in the same row), and if $\beta_i \neq \beta_j$ for $i \neq j$ (i.e. there is no row overlapping).

For later use in Lemma 4.6, we now argue that Algorithm 4.3 always gives a valid witness whenever it outputs a witness.

Lemma 4.4 *Let $\{s_k\}_{k=0}^{n-1}$ be the output of Algorithm 4.3 on inputs $\{\beta_i\}_{i=0}^{n-1}, \{t_i\}_{i=0}^{n-1}$ such that $\beta_i \neq \beta_j$ for all $i \neq j$ and $\prod_{j=0}^{n-1} [\![z_j]\!]^{\beta_i^j} = E(0, t_i)$ for all $0 \leq i < n$. Then, $[\![z_k]\!] = E(0, s_k)$, for all $0 \leq k < n$.*

Proof. The core of Algorithm 4.3 is explained in the following. Consider the matrix B defined by $B_{ij} = \beta_i^j$, for $0 \leq i, j < n$. By construction, B is a Vandermonde matrix since $\beta_i \neq \beta_j$ for all $i \neq j$. As B is an invertible matrix, so is its transpose B^T. Hence, there exist vectors d_k such that $B^T d_k$ is the $(k+1)$-th unit vector, for $0 \leq k < n-1$. This actually means that $\sum_{i=0}^{n-1} B_{ji}^T d_{k,i} = \sum_{i=0}^{n-1} B_{ij} d_{k,i} = 1$ only when $j = k$ and it is 0 otherwise. Vectors d_k can be computed using basic linear algebra.

Algorithm 4.2 ($\mathcal{W}_{\text{B-ZERO}}$) Witness-extended emulator for Protocol 4.1

Input: $x = \{[\![z_k]\!]\}_{k=0}^{n-1}$
  pick $\beta_0 \in_R \mathbb{Z}_q$
  pick $\lambda_0 \in_R \{0,1\}^k$
  run $(\beta_0, a_0, \lambda_0, u_0) \leftarrow \langle P^*(x,s), \tilde{V}(\beta_0, \lambda_0)\rangle$
  if $\langle(\beta_0, a_0, \lambda_0, u_0), x\rangle = 1$ **then**
    repeat
      pick $\lambda_0' \in_R \{0,1\}^k$
      run $(\beta_0, a_0, \lambda_0', u_0') \leftarrow \langle P^*(x,s), \tilde{V}(\beta_0, \lambda_0')\rangle$
    until $\langle(\beta_0, a_0, \lambda_0', u_0'), x\rangle = 1$
    for $i = 1$ to $n-1$ **do**
      repeat
        pick $\beta_i \in_R \mathbb{Z}_q$
        pick $\lambda_i \in_R \{0,1\}^k$
        run $(\beta_i, a_i, \lambda_i, u_i) \leftarrow \langle P^*(x,s), \tilde{V}(\beta_i, \lambda_i)\rangle$
      until $\langle(\beta_i, a_i, \lambda_i, u_i), x\rangle = 1$
      repeat
        pick $\lambda_i' \in_R \{0,1\}^k$
        run $(\beta_i, a_i, \lambda_i', u_i') \leftarrow \langle P^*(x,s), \tilde{V}(\beta_i, \lambda_i')\rangle$
      until $\langle(\beta_i, a_i, \lambda_i', u_i'), x\rangle = 1$
    end for
    if $\lambda_i \neq \lambda_i'$ for all i **and** $\beta_i \neq \beta_j$ for $i \neq j$ **then**
      for $i = 0$ to $n-1$ **do**
        $[\![z^{(i)}]\!] = \prod_{j=0}^{n-1} [\![z_j]\!]^{\beta_i^j}$
        run $t_i \leftarrow E_{\text{ZERO}}([\![z^{(i)}]\!], a_i, \lambda_i, u_i, \lambda_i', u_i')$
      end for
      run $\{s_k\}_{k=0}^{n-1} \leftarrow E_{\text{B-ZERO}}(\{\beta_i\}_{i=0}^{n-1}, \{t_i\}_{i=0}^{n-1})$
      return $((\beta_0, a_0, \lambda_0, u_0), \{s_k\}_{k=0}^{n-1})$
    else
      return $((\beta_0, a_0, \lambda_0, u_0), \perp)$
    end if
  else
    return $((\beta_0, a_0, \lambda_0, u_0), \perp)$
  end if

Algorithm 4.3 ($E_{\text{B-ZERO}}$) Extractor for Protocol 4.1

Input: $\{\beta_i\}_{i=0}^{n-1}, \{t_i\}_{i=0}^{n-1}$ where $\beta_i \neq \beta_j$ for $i \neq j$.
Output: $\{s_k\}_{k=0}^{n-1}$.
  find $d_k \in \mathbb{Z}_q^n$ such that $\sum_{i=0}^{n-1} \beta_i^j d_{k,i} = 1$ if $j = k$ and $\sum_{i=0}^{n-1} \beta_i^j d_{k,i} = 0$ otherwise
  $s_k = \sum_{j=0}^{n-1} t_j d_{k,j}$ for $0 \leq k < n$
  return $\{s_k\}_{k=0}^{n-1}$

Now,

$$\begin{aligned}
[\![z_k]\!] &= \prod_{i=0}^{n-1}\left(\prod_{j=0}^{n-1}[\![z_j]\!]^{B_{ij}}\right)^{d_{k,i}} \\
&= \prod_{i=0}^{n-1}\left(\prod_{j=0}^{n-1}[\![z_j]\!]^{\beta_i^j}\right)^{d_{k,i}} \\
&= \prod_{i=0}^{n-1}\left(\mathsf{E}(0,t_i)\right)^{d_{k,i}} \\
&= \mathsf{E}(0,\sum_{i=0}^{n-1} t_i d_{k,i}) \text{ (using homomorphic properties).}
\end{aligned}$$

From this equality, we can see that $[\![z_k]\!] = \mathsf{E}(0,\sum_{j=0}^{n-1} t_j d_{k,j})$ for $0 \leq k < n$. The output of Algorithm 4.3 is $s_k = \sum_{j=0}^{n-1} t_j d_{k,j}$. ■

The proof that Algorithm 4.3 gives a valid witness follows by analyzing the context in which the algorithm is executed. Observe that for all $0 \leq i < n$, $([\![z^{(i)}]\!]; t_i) \in R_{\text{ZERO}}$, and thus,

$$[\![z^{(i)}]\!] = \mathsf{E}(0,t_i), \text{ for } 0 \leq i < n, \tag{4.1}$$

because they are extracted using the special soundness extractor E_{ZERO} of a Σ-protocol for relation R_{ZERO}.

Moreover, we have that

$$[\![z^{(i)}]\!] = \prod_{j=0}^{n-1}[\![z_j]\!]^{\beta_i^j} \text{ for } 0 \leq i < n. \tag{4.2}$$

Combining Eq. (4.1) and Eq. (4.2) yields

$$\prod_{j=0}^{n-1}[\![z_j]\!]^{\beta_i^j} = \mathsf{E}(0,t_i).$$

Thus, Lemma 4.4 tells us that the output of extractor $E_{\text{B-ZERO}}$ defined in Algorithm 4.3 produces a valid witness for relation $R_{\text{B-ZERO}}$.

Lemma 4.5 *Emulator $\mathcal{W}_{\text{B-ZERO}}$ runs in expected polynomial-time.*

Proof. Before going into the computation of the overall running time, we focus our attention to the **for** loop of lines 9 through 19 of Algorithm 4.2. Algorithm 4.4 represents a single iteration of that loop. Let T' denote the random variable that counts the number of queries to P^* in Algorithm 4.4. We calculate $\mathbb{E}[T']$.

In Algorithm 4.4, there are two **repeat** loops. The expected number of queries to P^* in the first loop is $1/\epsilon$ since it is like sampling the matrix S until a 1 is hit. In the second loop the expected time is $1/\epsilon_\beta$ given that the 1 found in the first loop hits row β of matrix S.

In this way,

$$\begin{aligned}\mathbb{E}[T'] &= \sum_{v=0}^{q-1} \mathbb{P}[\beta = v]\mathbb{E}[T' \mid \beta = v] \\ &= \sum_{v=0}^{q-1} \frac{\epsilon_v}{q\epsilon}(\frac{1}{\epsilon} + \frac{1}{\epsilon_v}) \\ &= \frac{1}{\epsilon} + \sum_{v=0}^{q-1} \frac{1}{q\epsilon} \\ &= \frac{2}{\epsilon}.\end{aligned}$$

Note that in this context $\mathbb{P}[\beta = v]$ is the probability that after probing random positions in S until hitting a 1, that 1 belongs to row v. This probability can be calculated using matrix S. That is, in row v there are $2^k\epsilon_v$ 1's out of a total of $2^k q\epsilon$ in the whole matrix. Thus, $\mathbb{P}[\beta_i = v] = \epsilon_v/q\epsilon$.

The expected number of invocations to P^* the **for** loop of lines 9 through 19 of $\mathcal{W}_{\text{B-ZERO}}$ is $2(n-1)/\epsilon$. This is because Algorithm 4.4 is executed $n-1$ independent times.

We now calculate $\mathbb{E}[T]$ where T is the random variable counting the number of calls to P^* made by $\mathcal{W}_{\text{B-ZERO}}$. In order to shorten the notation, $\mathtt{tr}_i$ will denote the transcript $(v_i, a_i, \lambda_i, t_i) \leftarrow \langle P^*(x,s), \tilde{V}(v_i, \lambda_i)\rangle$.

$$\begin{aligned}\mathbb{E}[T] &= \sum_{v_0=0}^{q-1} \mathbb{P}[\beta_0 = v_0]\mathbb{E}[T \mid \beta_0 = v_0] \\ &= \frac{1}{q}\sum_{v_0=0}^{q-1} (\mathbb{P}[\langle \mathtt{tr}_0, x\rangle = 1 \mid \beta_0 = v_0]\mathbb{E}[T \mid \beta_0 = v_0 \wedge \langle \mathtt{tr}_0, x\rangle = 1] \\ &\qquad +\mathbb{P}[\langle \mathtt{tr}_0, x\rangle = 0 \mid \beta_0 = v_0]\mathbb{E}[T \mid \beta_0 = v_0 \wedge \langle \mathtt{tr}_0, x\rangle = 0]) \\ &= \frac{1}{q}\sum_{v_0=0}^{q-1} (\epsilon_{v_0}(1 + \frac{1}{\epsilon_{v_0}} + (n-1)\frac{2}{\epsilon}) + (1-\epsilon_{v_0})) \\ &= 2 + 2(n-1)\sum_{v_0=0}^{q-1} \frac{\epsilon_{v_0}}{q\epsilon} \\ &= 2n.\end{aligned}$$

Here $\mathbb{P}[\beta_0 = v_0] = 1/q$ as β_0 is picked at random from $\mathbb{Z}_q$. Overall, an average of $2n$ invocations of P^* are required. Since P^* runs in strict polynomial-time, the running time of $\mathcal{W}_{\text{B-ZERO}}$ is thus expected polynomial. ■

Lemma 4.6 *Let $P^*(x,s)$ be given and let $(\mathtt{tr}, w) \leftarrow \mathcal{W}_{\text{B-ZERO}}(P^*(x,s), x)$. Then $\mathbb{P}[(x;w) \in R_{\text{B-ZERO}}] \cong \epsilon$.*

Proof. This lemma shows that $\mathcal{W}_{\text{B-ZERO}}$ gives a valid witness with essentially the same probability as P^* would when running the protocol. Basically, we have to compute the probability of success of the emulator where 'success' means that $\mathcal{W}_{\text{B-ZERO}}$ manages to

Algorithm 4.4 Auxiliary Subroutine of $\mathcal{W}_{\text{B-ZERO}}$

repeat
  pick $\beta \in_R \mathbb{Z}_q$
  pick $\lambda \in_R \{0,1\}^k$
  run $(\beta, a, \lambda, u) \leftarrow \langle P^*(x,s), \tilde{V}(\beta,\lambda)\rangle$
until $\langle(\beta, a, \lambda, u), x\rangle = 1$
repeat
  pick $\lambda' \in_R \{0,1\}^k$
  run $(\beta, a, \lambda', u') \leftarrow \langle P^*(x,s), \tilde{V}(\beta,\lambda')\rangle$
until $\langle(\beta, a, \lambda', u'), x\rangle = 1$

extract a witness for x. This happens when the conditions of line 16 are true. That is, $\lambda_i \neq \lambda_i'$ and $\beta_i \neq \beta_j$ for all $i \neq j$, $1 \leq i, j \leq n$. We denote this event with *succ*. By construction, it immediately follows that $\mathbb{P}[succ] \leq \epsilon$.

If ϵ happens to be negligible then the lemma trivially holds. In the following, we argue that the lemma also holds when ϵ is not negligible.

We observe that the event *succ* can be explained in terms of matrix S. We probe matrix S until we hit $2n$ 1's. Then the probability of *succ* is the probability that there is no collision within rows β_i's (i.e., $\gamma_i \neq \gamma_i'$), and that all β_i's are different. We separate our analysis of the event *succ* in these two events.

First, let C_v be the event that there is no collision in row v. More specifically, looking at Algorithm 4.4 we want to know what is the probability that given that $\beta = v$ in the first **repeat** loop, it holds that $\lambda \neq \lambda'$. Since 1 out of $2^k\epsilon_v$ 1's gives a collision in row S_v, we conclude that $\mathbb{P}[C_v] = 1 - 1/2^k\epsilon_v$.

Secondly, for the probability of different β_i's we consider the Algorithm 4.5 which is a subroutine of $\mathcal{W}_{\text{B-ZERO}}$. It comprises all statements where different β_i's are probed until a total of n 1's are hit in matrix S.

Let D denote the event that the β_i's selected in Algorithm 4.5 are all different. Then,

$$\begin{aligned}
\mathbb{P}[D] &= \sum_{v_0} \mathbb{P}[\beta_0 = v_0]\mathbb{P}[D \mid \beta_0 = v_0] \\
&= \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}\Big(\sum_{v_1 \notin \{v_0\}} \mathbb{P}[\beta_1 = v_1]\mathbb{P}[D \mid \beta_0 = v_0 \wedge \beta_1 = v_1 \wedge v_1 \neq v_0]\Big) \\
&\;\;\vdots \\
&= \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}\Big(\sum_{v_1 \notin \{v_0\}} \frac{\epsilon_{v_1}}{q\epsilon}\Big(\sum_{v_2 \notin \{v_0, v_1\}} \frac{\epsilon_{v_2}}{q\epsilon}\big(\cdots\big(\sum_{v_{n-1} \notin \{v_i\}_{i=0}^{n-2}} \frac{\epsilon_{v_{n-1}}}{q\epsilon}\big)\cdots\big)\Big)\Big).
\end{aligned}$$

This probability is computed by conditioning on the different rows v_i of S that are being hit pondered with the probability that row v_i is selected. This long expression for this probability is left as is. Below, we use it to compute the probability of the event *succ*.

Below, we analyze the probability of the event *succ*, combining the two parts dis-

Algorithm 4.5 Auxiliary Subroutine of $\mathcal{W}_{\text{B-ZERO}}$

for $i = 0$ to $n-1$ **do**
  repeat
    pick $\beta_i \in_R \mathbb{Z}_q$
    pick $\lambda_i \in_R \{0,1\}^k$
    run $(\beta_i, a_i, \lambda_i, u_i) \leftarrow \langle P^*(x,s), \tilde{V}(\beta_i, \lambda_i)\rangle$
  until $\langle(\beta_i, a_i, \lambda_i, u_i), x\rangle = 1$
end for
return $\{\beta_i\}_{i=0}^{n-1}$

cussed above.

$$
\begin{aligned}
\mathbb{P}[succ] &= \mathbb{P}[\langle \mathtt{tr}_0, x\rangle = 1]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1] + \mathbb{P}[\langle \mathtt{tr}_0, x\rangle = 0]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 0]) \\
&= \epsilon \sum_{v_0} \mathbb{P}[\beta_0 = v_0 \mid \langle \mathtt{tr}_0, x\rangle = 1]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0] \\
&= \epsilon \sum_{v_0} \mathbb{P}[\beta_0 = v_0]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0] \\
&= \epsilon \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0] \\
&= \epsilon \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}\mathbb{P}[C_{v_0} \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0 \wedge C_{v_0}] \\
&= \epsilon \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}\mathbb{P}[C_{v_0}](\sum_{v_1 \notin \{v_0\}} \mathbb{P}[\beta_1 = v_1] \cdot \\
&\qquad \mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0 \wedge C_{v_0} \wedge \beta_1 = v_1]) \\
&= \epsilon \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_0}})(\sum_{v_1 \notin \{v_0\}} \frac{\epsilon_{v_1}}{q\epsilon} \cdot \\
&\qquad \mathbb{P}[C_{v_1}]\mathbb{P}[succ \mid \langle \mathtt{tr}_0, x\rangle = 1 \wedge \beta_0 = v_0 \wedge C_{v_0} \wedge \beta_1 = v_1 \wedge C_{v_1}]) \\
&= \epsilon \sum_{v_0} \frac{\epsilon_{v_0}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_0}})\Bigg(\sum_{v_1 \notin \{v_0\}} \frac{\epsilon_{v_1}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_1}})\Big(\cdots \\
&\quad (\sum_{v_{n-2} \notin \{v_i\}_{i=0}^{n-3}} \frac{\epsilon_{v_{n-2}}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_{n-2}}})(\sum_{v_{n-1} \notin \{v_i\}_{i=0}^{n-2}} \frac{\epsilon_{v_{n-1}}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_{n-1}}}))) \cdots \Big)\Bigg)
\end{aligned}
$$

To evaluate this formula we define the following sequence:

$$f_m = \sum_{v_{n-m} \notin \{v_i\}_{i=0}^{n-m-1}} \frac{\epsilon_{v_{n-m}}}{q\epsilon}(1 - \frac{1}{2^k\epsilon_{v_{n-m}}})f_{m-1}.$$

where $f_0 = 1$, and f_m is a function of $v_0, \ldots, v_{n-m-1}$. By construction, $\mathbb{P}[succ] = \epsilon f_n$.

Claim 4.7 *If ϵ is not negligible, then $f_m \geq B_m$ for $0 \leq m \leq n$, where*

$$B_m = 1 - m\sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon}.$$

with $B_m > 0$.

Note that B_m is also a function of $v_0, \ldots, v_{n-m-1}$.

Proof. Let us first argue that $B_m > 0$. Assume the opposite, then

$$1 - m \sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon} \leq 0$$

$$1 \leq m \sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q\epsilon} + \frac{m(m-1)}{2q\epsilon} + \frac{m}{2^k\epsilon}$$

$$\epsilon \leq m \sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q} + \frac{m(m-1)}{2q} + \frac{m}{2^k}.$$

This says that ϵ is bounded above by a negligible number which contradicts the assumption that ϵ is not negligible. Hence B_m must be positive.

We prove that $f_m \geq B_m$ by induction on m. The bound clearly holds for $m = 0$.

For any m, $0 \leq m < n$ we have:

$$f_{m+1} = \sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \frac{\epsilon_{v_{n-m-1}}}{q\epsilon}\left(1 - \frac{1}{2^k\epsilon_{v_{n-m-1}}}\right) f_m$$

$$\geq \sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \frac{\epsilon_{v_{n-m-1}}}{q\epsilon}\left(1 - \frac{1}{2^k\epsilon_{v_{n-m-1}}}\right)\left(1 - m \sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon}\right) \quad (4.3)$$

$$\geq \sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \frac{\epsilon_{v_{n-m-1}}}{q\epsilon}\left(1 - m \sum_{i=0}^{n-m-1} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon} - \frac{1}{2^k\epsilon_{v_{n-m-1}}}\right) \quad (4.4)$$

$$= \left(\sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \frac{\epsilon_{v_{n-m-1}}}{q\epsilon}\right)\left(1 - m \sum_{i=0}^{n-m-2} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon}\right) -$$

$$-m \sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \left(\frac{\epsilon_{v_{n-m-1}}}{q\epsilon}\right)^2 - \sum_{v_{n-m-1} \notin \{v_i\}_{i=0}^{n-m-2}} \frac{1}{2^k q\epsilon}$$

$$\geq \left(1 - \sum_{i=0}^{n-m-2} \frac{\epsilon_{v_i}}{q\epsilon}\right)\left(1 - m \sum_{i=0}^{n-m-2} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{m(m-1)}{2q\epsilon} - \frac{m}{2^k\epsilon} - \frac{m}{q\epsilon}\right) - \frac{1}{2^k\epsilon} \quad (4.5)$$

$$\geq 1 - (m+1) \sum_{i=0}^{n-(m+1)-1} \frac{\epsilon_{v_i}}{q\epsilon} - \frac{(m+1)m}{2q\epsilon} - \frac{m+1}{2^k\epsilon}. \quad (4.6)$$

Eq. (4.3) follows from the induction hypothesis and because B_m is positive. To get Eq. (4.4) and (4.5) we discard all positive terms after distributing the multiplication. Moreover, in Eq. (4.5) we note that $\epsilon_v^2 \leq \epsilon_v$ because $\epsilon_v \leq 1$, and therefore $\sum_v \epsilon_v^2 \leq \sum_v \epsilon_v$. Discarding all positive terms once more yields Eq. (4.6).

This finishes the proof of Claim 4.7. ■

Under the condition that ϵ is not negligible we see that $f_n \geq B_n$ where $B_n = 1 - n(n-1)/2q\epsilon - n/2^k\epsilon$ with $B_n > 0$. This added to the fact that $\mathbb{P}[succ] = \epsilon f_n$ allows us to conclude that

$$\mathbb{P}[succ] \geq \epsilon\left(1 - \frac{n(n-1)}{2q\epsilon} - \frac{n}{2^k\epsilon}\right) = \epsilon - \frac{n(n-1)}{2q} - \frac{n}{2^k}.$$

Protocol 4.2 Honest verifier zero-knowledge proof of knowledge for relation R_{ROT}

Common Input : $\{[\![x_k]\!]\}_{k=0}^{n-1}$, $\{[\![y_k]\!]\}_{k=0}^{n-1}$.
Prover's Private Input : $\{s_k\}_{k=0}^{n-1}$, rotation offset r with $0 \leq r < n$.

Prover		**Verifier**
		$\beta \in_R \mathbb{Z}_q$
	β $\longleftarrow$	
	Everyone computes:	
$t = \sum_{j=0}^{n-1} \beta^j s_j$,	$\{[\![z_k]\!] = \prod_{j=0}^{n-1}([\![y_{j+k}]\!]/[\![x_j]\!])^{\beta^j}\}_{k=0}^{n-1}$	
	Run Σ-protocol for $R_{\text{OR-ZERO}}(\{[\![z_k]\!]\}_{k=0}^{n-1}; r, t)$	

On the other hand, we know that $\mathbb{P}[succ] \leq \epsilon$ and thus $\epsilon - n(n-1)/2q - n/2^k \leq \mathbb{P}[succ] \leq \epsilon$ meaning that $\mathbb{P}[succ] \cong \epsilon$ for non-negligible ϵ.

Once the event *succ* happens, it follows from Lemma 4.4 that the extractor $E_{\text{B-ZERO}}$ of Algorithm 4.3 gives a valid witness. Thus, $\mathbb{P}[(x; w) \in R_{\text{B-ZERO}}] \cong \epsilon$ which finishes the proof of Lemma 4.6. ■

4.3 General Proof of Rotation

Protocol 4.2 gives a solution for the relation of rotation of homomorphic encryptions R_{ROT}. The structure is similar to that of Protocol 4.1: the first message is a challenge and the validity of the proof is tested via a Σ-protocol.

The intuition of the approach is as follows. First note that if the two lists of homomorphic encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and $\{[\![y_k]\!]\}_{k=0}^{n-1}$ are a rotation by r positions and re-randomization of each other then for all $0 \leq k < n$ it holds that $[\![y_{k+r}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k)$ for randomizers s_k. This means that $[\![y_{k+r}]\!]/[\![x_k]\!] = \mathsf{E}(0, s_k)$ for all $0 \leq k < n$. This can be proved using the proof of multiple encryptions of 0 given by Protocol 4.1 since $(\{[\![y_{k+r}]\!]/[\![x_k]\!]\}_{k=0}^{n-1}; \{s_k\}_{k=0}^{n-1}) \in R_{\text{B-ZERO}}$. Doing this requires to compute publicly the encryption $[\![z_r]\!] = \prod_{j=0}^{n-1}([\![y_{j+r}]\!]/[\![x_j]\!])^{\beta^j}$ where the respective secret for the prover is $t = \sum_{j=0}^{n-1} \beta^j s_j$ (see Protocol 4.1).

Unfortunately, the rotation offset r is exposed since $[\![z_r]\!]$ must be publicly computed. Protocol 4.2 avoids that by computing *all* $[\![z_k]\!]$ for $0 \leq k < n$ for each possible rotation offset. By means of an OR-composition of Σ-protocols it can be proved that one of the n possible rotation offsets is used while keeping the actual rotation offset in secret. Hence, we have the proof for rotation.

Algorithm 4.6 (ZLIST) Computing list $\{[\![z_k]\!]\}_{k=0}^{n-1}$ of Protocol 4.2 efficiently

Input: β, $\{[\![x_k]\!]\}_{k=0}^{n-1}$, $\{[\![y_k]\!]\}_{k=0}^{n-1}$.
Output: $\{[\![z_k]\!]\}_{k=0}^{n-1}$ such that $[\![z_k]\!] = \prod_{j=0}^{n-1}([\![y_{j+r}]\!]/[\![x_j]\!])^{\beta^j}$.

$[\![G]\!] = \prod_{j=0}^{n-1}[\![x_j]\!]^{\beta^j}$
$[\![H_0]\!] = \prod_{j=0}^{n-1}[\![y_j]\!]^{\beta^j}$
$[\![z_0]\!] = [\![H_0]\!]/[\![G]\!]$
for $i = 1$ to $n-1$ **do**
  $[\![H_i]\!] = ([\![H_{i-1}]\!][\![y_{i-1}]\!]^{\beta^n-1})^{\beta^{-1}}$
  $[\![z_i]\!] = [\![H_i]\!]/[\![G]\!]$
end for
return $\{[\![z_k]\!]\}_{k=0}^{n-1}$

4.3.1 Performance Analysis

Protocol 4.2 requires 4 rounds of interaction. As for the computational complexity, taking into account only the Σ-protocol, the solution requires $O(n)$ computation as it is an n-way OR-composition of Σ-protocols that require $O(1)$ computation each.

Before the Σ-protocol for $R_{\text{OR-ZERO}}$ is performed both the prover *and* potential verifiers must compute the list of encryptions $\{[\![z_k]\!]\}_{k=0}^{n-1}$. Note that it naively requires the computation of n times n-way exponentiations, yielding a total of $O(n^2)$ computation. Fortunately, due to the special form of the exponents we can reduce the number of exponentiations to proportional to n using the observation below. The overall computational complexity of Protocol 4.2 now becomes $O(n)$.

Proposition 4.8 *Let $\{[\![y_k]\!]\}_{k=0}^{n-1}$ be a list of n homomorphic encryptions and let $\beta \in \mathbb{Z}_q$. Then the lists of encryptions $\{[\![H_k]\!]\}_{k=0}^{n-1}$ and $\{[\![\tilde{H}_k]\!]\}_{k=0}^{n-1}$ are identical, where*

$$[\![H_i]\!] = \prod_{j=0}^{n-1}[\![y_{j+i}]\!]^{\beta^j}, \text{ for } 0 \leq i < n, \tag{4.7}$$

$$[\![\tilde{H}_0]\!] = [\![H_0]\!];\ [\![\tilde{H}_i]\!] = ([\![\tilde{H}_{i-1}]\!][\![y_{i-1}]\!]^{\beta^n-1})^{\beta^{-1}}, \text{ for } 1 \leq i < n. \tag{4.8}$$

Proof. By induction on i. $[\![H_0]\!] = [\![\tilde{H}_0]\!]$, so we only need to prove the induction step.

Assume that $[\![H_i]\!] = [\![\tilde{H}_i]\!]$ for $0 \leq i < n$. Then,

$$\begin{aligned}
[\![H_{i+1}]\!] &= \prod_{j=0}^{n-1} [\![y_{j+i+1}]\!]^{\beta^j} \\
&= [\![y_i]\!]^{\beta^{n-1}} \prod_{j=0}^{n-2} [\![y_{j+i+1}]\!]^{\beta^j} \\
&= (\prod_{j=0}^{n-2} [\![y_{j+i+1}]\!]^{\beta^{j+1}})^{\beta^{-1}} [\![y_i]\!]^{\beta^{n-1}} \\
&= (\prod_{j=1}^{n-1} [\![y_{j+i}]\!]^{\beta^j})^{\beta^{-1}} [\![y_i]\!]^{\beta^{n-1}} \\
&= ([\![H_i]\!] / [\![y_i]\!])^{\beta^{-1}} [\![y_i]\!]^{\beta^{n-1}} \\
&= ([\![\tilde{H}_i]\!] / [\![y_i]\!])^{\beta^{-1}} [\![y_i]\!]^{\beta^{n-1}} \\
&= ([\![\tilde{H}_i]\!] [\![y_i]\!]^{\beta^n - 1})^{\beta^{-1}} \\
&= [\![\tilde{H}_{i+1}]\!].
\end{aligned}$$

■

Computing list $\{[\![H_k]\!]\}_{k=0}^{n-1}$ via $\{[\![\tilde{H}_k]\!]\}_{k=0}^{n-1}$ as in Eq. (4.8) requires only $O(n)$ exponentiations. This fact can be used in Protocol 4.2 to compute the list $\{[\![z_k]\!]\}_{k=0}^{n-1}$ using a linear number of exponentiations. Algorithm 4.6 shows an efficient way to do this.

Proposition 4.9 *Algorithm 4.6 needs $O(n)$ exponentiations to compute the list $\{[\![z_k]\!]\}_{k=0}^{n-1}$ of Protocol 4.2.*

The result follows by observing that $[\![H_k]\!] = \prod_{j=0}^{n-1} [\![y_{j+k}]\!]^{\beta^j}$, for $0 \leq k < n$ using Prop. 4.8.

4.3.2 Security Analysis

Theorem 4.10 *Protocol 4.2 is a complete, honest verifier zero-knowledge proof of knowledge with witness-extended emulation for relation R_{ROT}.*

Proof. We proceed in a similar fashion as in Theorem 4.3. Indeed, completeness and zero-knowledgeness follow in the same way. We now address witness-extended emulation.

Algorithm 4.7 gives a full description of $\mathcal{W}_{\text{ROT}}$, a witness-extended emulator for Protocol 4.2. Roughly speaking, it follows the same strategy of Algorithm 4.2 with a few particular features that we describe below.

We first argue that $\mathcal{W}_{\text{ROT}}$ extracts a witness for R_{ROT} on encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and $\{[\![y_k]\!]\}_{k=0}^{n-1}$ as inputs. It needs n witnesses for $R_{\text{OR-ZERO}}$ obtained on different challenges β_i's. These witnesses are obtained using the special soundness property of the Σ-protocol for relation $R_{\text{OR-ZERO}}$ by means of its extractor $E_{\text{OR-ZERO}}$. Since we are dealing with an OR-composition of Σ-protocols, its extractor delivers a witness for (one of) the valid instance(s) of the composition.

In order to conclude a successful extraction of the witness for the relation R_{ROT}, we need n witnesses of $R_{\text{OR-ZERO}}$ where the *same* rotation offset r is obtained. We note that P^*

may give valid proofs for $R_{\text{OR-ZERO}}$ using different offsets each time. Think of the list of encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ having all equal plaintexts for which the new list of encryptions $\{[\![y_k]\!]\}_{k=0}^{n-1}$ is computed. Any rotation offset can be used to prove a rotation between the two lists of encryptions.

Algorithm 4.7 ($\mathcal{W}_{\text{ROT}}$) Witness-extended emulator for Protocol 4.2

Input: $x = (\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1})$
  pick $\beta_0 \in_R \mathbb{Z}_q$
  pick $\lambda_0 \in_R \{0,1\}^k$
  run $(\beta_0, a_0, \lambda_0, u_0) \leftarrow \langle P^*(x,s), \tilde{V}(\beta_0, \lambda_0)\rangle$
  if $\langle(\beta_0, a_0, \lambda_0, u_0), x\rangle = 1$ **then**
    for $i = 1$ to $n(n-1)$ **do**
      repeat
        pick $\beta_i \in_R \mathbb{Z}_q$
        pick $\lambda_i \in_R \{0,1\}^k$
        run $(\beta_i, a_i, \lambda_i, u_i) \leftarrow \langle P^*(x,s), \tilde{V}(\beta_i, \lambda_i)\rangle$
      until $\langle(\beta_i, a_i, \lambda_i, u_i), x\rangle = 1$
    end for
    for $i = 0$ to $n(n-1)$ **do**
      repeat
        pick $\lambda_i' \in_R \{0,1\}^k$
        run $(\beta_i, a_i, \lambda_i', u_i') \leftarrow \langle P^*(x,s), \tilde{V}(\beta_i, \lambda_i')\rangle$
      until $\langle(\beta_i, a_i, \lambda_i', u_i'), x\rangle = 1$
    end for
    if $\lambda_i \neq \lambda_i'$ for all i **and** $\beta_i \neq \beta_j$ for $i \neq j$ **then**
      $A = \emptyset$
      for $i = 0$ to $n(n-1)$ **do**
        $\{[\![z_k^{(i)}]\!]\}_{k=0}^{n-1} = \text{ZLIST}(\beta_i, \{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1})$
        run $(r_i, t_i) \leftarrow E_{\text{OR-ZERO}}(\{[\![z_k^{(i)}]\!]\}_{k=0}^{n-1}, a_i, \lambda_i, u_i, \lambda_i', u_i')$
        $A \leftarrow A \cup \{(r_i, t_i)\}$
      end for
      Let i_k be such that $\{(r_{i_k}, t_{i_k})\}_{k=0}^{n-1} \subset A$ with $r = r_{i_k}$ for $0 \leq k < n$.
      run $\{s_k\}_{k=0}^{n-1} \leftarrow E_{\text{B-ZERO}}(\{\beta_{i_k}\}_{k=0}^{n-1}, \{t_{i_k}\}_{k=0}^{n-1})$
      return $((\beta_0, a_0, \lambda_0, u_0), (r, \{s_k\}_{k=0}^{n-1}))$
    else
      return $((\beta_0, a_0, \lambda_0, u_0), \perp)$
    end if
  else
    return $((\beta_0, a_0, \lambda_0, u_0), \perp)$
  end if

To cover the possibility of having different rotation offsets extracted, emulator $\mathcal{W}_{\text{ROT}}$ collects a total of $n(n-1)+1$ valid witnesses for $R_{\text{OR-ZERO}}$. After this many steps it is guaranteed that there exists an offset r such that there are n witnesses for that particular offset r.

Let the n witnesses of $R_{\text{OR-ZERO}}$ be $\{(r, t_{i_j})\}_{j=0}^{n-1}$ with their respective challenges $\{\beta_{i_j}\}_{j=0}^{n-1}$.

Then

$$[\![z_r^{(i_j)}]\!] = \mathsf{E}(0, t_{i_j}), \tag{4.9}$$

where by definition

$$[\![z_r^{(i_j)}]\!] = \prod_{l=0}^{n-1} \left(\frac{[\![y_{l+r}]\!]}{[\![x_l]\!]} \right)^{\beta_{i_j}^l}. \tag{4.10}$$

Algorithm 4.3 is applied to inputs $\{\beta_{i_j}\}_{j=0}^{n-1}$ and $\{t_{i_j}\}_{j=0}^{n-1}$ which satisfy Eqs. (4.9) and (4.10). Therefore, the output $\{s_k\}_{k=0}^{n-1}$ satisfies

$$\frac{[\![y_{k+r}]\!]}{[\![x_k]\!]} = \mathsf{E}(0, s_k),$$

meaning that the list $\{[\![y_k]\!]\}_{k=0}^{n-1}$ is a rotation and re-randomization of the list $\{[\![x_k]\!]\}_{k=0}^{n-1}$ by r positions with $\{s_k\}_{k=0}^{n-1}$ as the used randomizers.

We show that the hypotheses of Lemma 2.6 are satisfied by $\mathcal{W}_{\text{ROT}}$ following similar arguments as in Theorem 4.3. Indistinguishability of transcripts (Lemma 2.6(ii)) and that $\mathcal{W}_{\text{ROT}}$ outputs accepting transcripts when it gives valid witnesses (Lemma 2.6(iii)) are straightforward to check. In the following two claims we analyze Lemma 2.6(i) and (iv) respectively.

Claim 4.11 *Emulator $\mathcal{W}_{\text{ROT}}$ runs in expected polynomial-time.*

In fact, as in Lemma 4.5 it can be shown that the number of invocations of P^* is $2(n(n-1)+1)$, thus yielding overall expected polynomial-time.

Claim 4.12 *If $(t\boldsymbol{r}, w) \leftarrow \mathcal{W}_{\text{ROT}}(x)$, then $\mathbb{P}[(x; w) \in R_{\text{ROT}}] \cong \epsilon$.*

Here ϵ is defined as before, being the probability that P^* gives an accepting proof on randomly selected challenges. In line with the reasoning used in Lemma 4.6 it is now easy to show that if ϵ is not negligible then

$$\epsilon - \frac{n^4/2 - n^3 + n^2 - n/2}{q} - \frac{n(n-1)+1}{2^k} \leq \mathbb{P}[(x; w) \in R_{\text{ROT}}] \leq \epsilon,$$

whereas for negligible ϵ it is true that $0 \leq \mathbb{P}[(x; w) \in R_{\text{ROT}}] \leq \epsilon$. In both cases we can conclude that $\mathbb{P}[(x; w) \in R_{\text{ROT}}] \cong \epsilon$.

Thus, $\mathcal{W}_{\text{ROT}}$ has witness-extended emulation. ■

There are various ways in which the performance of $\mathcal{W}_{\text{ROT}}$ can be optimized without spoiling its properties. For instance, emulator $\mathcal{W}_{\text{ROT}}$ could keep track of all extracted rotation offsets in the witnesses for $R_{\text{OR-ZERO}}$, and as soon as it gets n with the same rotation offset proceed to extract the witness for R_{ROT}. In the worst case, emulator $\mathcal{W}_{\text{ROT}}$ will run loops of $n(n-1)+1$ iterations.

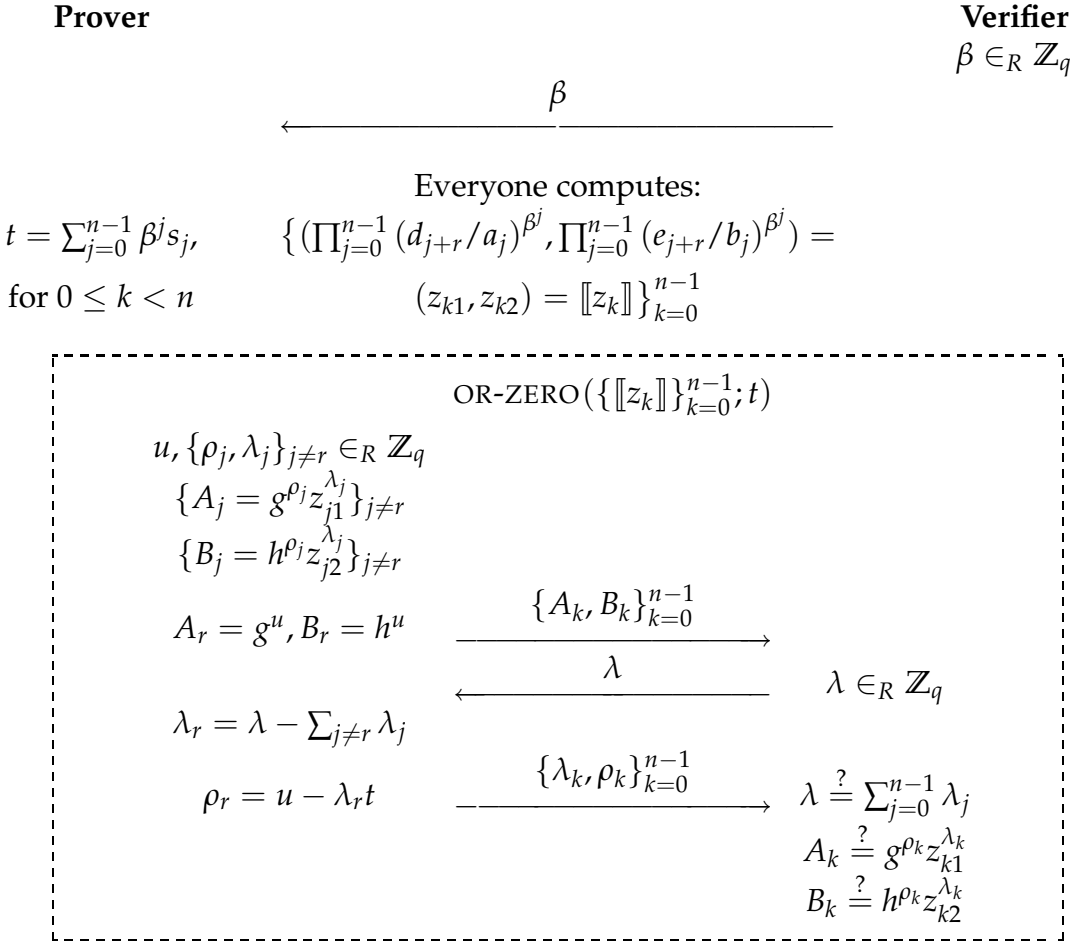
4.3.3 Rotation of ElGamal Encryptions

Protocol 4.3 presents the complete proof for the case of ElGamal encryptions. Protocol OR-ZERO is the Σ-protocol of the OR-composition of relation R_{ZERO}.

The full description of the protocol shows that the Σ-protocol OR-ZERO requires a handful of exponentiations. Also, the use of Algorithm 4.6 helps to keep the computation of the list $\{[\![z_k]\!]\}_{k=0}^{n-1}$ low as well.

Protocol 4.3 Honest verifier zero-knowledge proof of knowledge for R_{ROT} for homomorphic ElGamal cryptosystem

Common Input	:	$\{(a_k, b_k) = [\![x_k]\!]\}_{k=0}^{n-1}$, $\{(d_k, e_k) = [\![y_k]\!]\}_{k=0}^{n-1}$.
Prover's Private Input	:	$\{s_k\}_{k=0}^{n-1}$, offset r with $0 \leq r < n$.



4.4 Related Work and Efficiency Comparison

There exist techniques in the literature to show that a rotation has been applied. Reiter and Wang [RW04] present the first protocol for shuffles restricted to a rotation. Their approach is rather inefficient as they need *four* sequential invocations to a protocol for general shuffling. De Hoog *et al.* [dHSŠV09] present a general protocol following a "two-stage paradigm", analogous to that for general shuffling in [Nef01, Gro03, GI08]. The high-level idea of all these shuffling protocols is to first give a protocol to show that a list of *known* values is committed in a shuffled (or, rotated) manner and use it as a stepping-stone to present the general solution.

Recently, Terelius and Wikström [TW10] have proposed a completely different and interesting approach to prove that a shuffle uses a permutation in a restricted set. The restricted set is the automorphism group of a partially oriented hypergraph. The approach works by first letting the shuffler commit to its permutation and then prove both that the permutation belongs to the automorphism group of a hypergraph and that the shuffle was done according the committed permutation. The idea of committing to a

Protocol	Prove	Verify	Rounds
Loop permutations [RW04]	$30n$	$28n$	28
DFT-based (Protocol 3.1)	$5n$	$4n$	3
General (Protocol 4.3)	$9n$	$9n$	4
General [dHSŠV09]	$9n$	$9n$	6
Graph automorphism [TW10]	$11n$	$9n$	5

Table 4.1: Computational and round complexities for different protocols for rotation.

permutation, first introduced in [Wik09], allows us to move the commitment stage to a pre-computation phase, while checking that the shuffle is done according to the committed matrix requires few computations in the on-line phase. In the particular case that the (hyper)graph has as automorphism group the set of all rotations, provides a protocol that is overall slightly less efficient than our solutions. The technique is attractive, though, since it allows to prove a shuffle using a permutation from many interesting families of permutations.

The solutions presented in this thesis are very efficient and the performance figures are comparable to those for proving a general shuffle. In fact, the DFT-based solution (Protocol 3.1) is slightly more efficient than Groth's shuffling protocol [Gro03]. The protocol for rotation presented in this chapter is as efficient as the general protocol for rotations in [dHSŠV09]. Our protocol, however, proves a rotation in *one go*, without any auxiliary protocols, which results in a protocol with 4 rounds instead of 6 rounds.

Table 4.1 presents a comparison of the computational complexities for various protocols for rotation. In all cases, we assume that the protocols are based on homomorphic ElGamal and Pedersen commitments. Additionally, for the protocol in [RW04] we assume that they use the proof of shuffle by Groth [Gro03] which is the computationally most efficient to the best of our knowledge.

Chapter 5

Verifiable Multiply Fragile Shuffles

We consider a generalization of the problem of rotation in a shuffle of encryptions. Namely, we look at zero-knowledge protocols to show that the shuffle applied belongs to a multiply fragile set of permutations. A set of permutations is k-fragile, as defined by Reiter and Wang [RW04], if any k input-output correspondences of a permutation uniquely identifies a permutation within the set. Shuffles showing that a permutation is k-fragile may be applied to construct fragile mixes.

In this chapter we present the first protocols for proving in zero-knowledge that a 2-fragile and 3-fragile permutation is applied in a shuffle. Concretely, we give protocols for the affine transformation and the Möbius transformation on a projective field. Interestingly, the two main proofs presented in this chapter can be expressed in terms of proofs of rotations.

Note that since the set of affine transformations and the set of Möbius transformations are both subgroups of the symmetric group, one can think of a cascade of these sets of permutations. In this chapter we analyze how such cascades can be constructed. In particular, we see how the DFT-based solution can be based such that the average computation per shuffler is amortized over the cascade length.

Besides of the protocol development, we point out the link between the definitions of fragile and transitive set of permutations. In fact, we show that under some conditions both definitions are equivalent. The idea of a transitive set of permutations is a very well studied problem in combinatorics. In fact, we review some (non-)existence results regarding multiply transitive set of permutations.

5.1 Fragile Permutations

Consider the following threat in the context of cascades of shufflers (a.k.a. mix-networks). After a cascade has been executed and successfully verified, no one can prevent any of the shufflers from disclosing partial information about the permutation they have used. For instance, all shufflers may disclose input-output correspondences throughout the cascade in order to track where a certain input element ends after the cascade of shuffles. This passive attack seriously compromises the secrecy that a cascade is supposed to provide, and is clearly undetectable by the users of the cascade.

With this attack in mind, Reiter and Wang [RW04] defined the concept of *fragile mixing* as a way to discourage shufflers from revealing any information about the permutation used in a shuffle. In a fragile mix, some of the shufflers are required to shuffle using a

	α	β	δ	γ
π_1	β	α	γ	δ
π_2	δ	β	α	γ
π_3	α	γ	δ	β
π_4	γ	δ	β	α

Figure 5.1: A Latin square on alphabet $A = \{\alpha, \beta, \delta, \gamma\}$.

permutation belonging to a restricted set. The distinguishing feature of a k-fragile permutation set is that revealing k correspondences between input and output messages suffices to reveal the permutation that has been used. A shuffler using such a fragile permutation is able to disclose limited information about the permutation before the permutation is fully revealed. In fact, once the permutation is revealed its reputation as shuffler may be affected and therefore, the shuffler will be deterred from revealing "too much" information about the permutation used in a fragile-mix.

Before going into the formal definitions, we give some common notation used throughout this chapter. The set $\mathbb{S}_n$ denotes the symmetric group on a set X of n elements. With $\mathbb{S}_X$ we denote the symmetric group of permutations on the set X where the set X is explicitly indicated.

Definition 5.1 *The subset F of $\mathbb{S}_n$ is* k-fragile *if for any set $D \subset X$ of k elements, and $\pi, \pi' \in F$ it holds that $\pi(x) = \pi'(x)$ for all $x \in D$ only when $\pi = \pi'$.*

The definition says that any element of the k-fragile set F is uniquely identified by a collection of k input-to-output mappings.

A trivial example of a 1-fragile set of permutations is a singleton set. We present in the following some other more elaborated examples of 1-fragile sets of permutations.

- **Rotation.** The set of the n cyclic shifts of $\mathbb{S}_{\mathbb{Z}_n}$, $F_{\text{ROT}} = \{\pi_r \in \mathbb{S}_{\mathbb{Z}_n} : \pi_r(x) = x + r \bmod n, r \in \mathbb{Z}_n\}$.

- **Latin square.** A *Latin square* is a $n \times n$ array which contains elements from an alphabet of n symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column. If each column is associated with a unique symbol of the alphabet then the set of rows of that table are seen as the images of the permutations on the alphabet (see Fig. 5.1). That set of permutations is fragile. Also, multiplication tables of a group operation can be seen as a Latin squares.

- **Scaling.** This set comprises all permutations on $\mathbb{Z}_n^*$ that rescale the positions by an element in $\mathbb{Z}_n^*$. More specifically, $F_{\text{SCL}} = \{\pi_a \in \mathbb{S}_{\mathbb{Z}_n^*} : \pi_a(x) = ax \bmod n, a \in \mathbb{Z}_n^*\}$.

- **One-time pad.** Consider $X = \{0,1\}^\ell$. A permutation is obtained by bit-wise XORing every element in X w.r.t. an ℓ-bit key. $F_{\text{OTP}} = \{\pi_K \in \mathbb{S}_X : \pi_K(x) = x \oplus K, K \in X\}$.

Proposition 5.2 *Let $F \subset \mathbb{S}_n$ be a k-fragile set of permutations. Then $|F| \leq n!/(n-k)!$.*

Proof. From the definition of fragility we can uniquely identify every permutation of F with a subset of k elements in X. We can associate $\pi \in F$ with $(\pi(x_1), \ldots, \pi(x_k))$ for a fixed tuple $(x_1, \ldots, x_k)$ of elements in X. Since F is k-fragile, any other permutation $\pi' \in F$ cannot be such that $\pi'(x_j) = \pi(x_j)$. This upper-bounds the size of F by the number of ordered subsets of k elements from X, so by $n!/(n-k)!$. ■

Observe moreover that if F is k-fragile, then so is F' with $F' \subset F$. A way to check whether a given set of permutations $F \subset \mathbb{S}_n$ is k-fragile is as follows. Clearly, if $|F| > n!/(n-k)!$ then F is not k-fragile. Otherwise, we take any subset $D \subset X$ of k elements and map it using each permutation in F. The k-fragility means that all images must be different.

Proposition 5.3 *A subset $F \in \mathbb{S}_n$ is k-fragile if and only if for every k distinct elements $x_0, \ldots, x_{k-1} \in X$ and $y_0, \ldots, y_{k-1} \in X$ there exist at most one permutation $\pi \in F$ such that $\pi(x_j) = y_j$ for all $0 \leq j < k$.*

Proof. Let $x_0, \ldots, x_{k-1} \in X$ and $y_0, \ldots, y_{k-1} \in X$ be two lists of non-repeating elements such that there are at least two permutations π and π' in F such that $\pi(x_j) = \pi'(x_j) = y_j$. Then, F is not k-fragile.

If F is not k-fragile, then there exists a set $D \subset F$ of k elements for which there are two permutations π and π' with $\pi(x) = \pi'(x)$ for all $x \in D$, yet $\pi \neq \pi'$. Define $\{x_0, \ldots, x_{k-1}\} = D$. Taking $y_j = \pi(x_j)$ for all $0 \leq j < k$, we have that π and π' are two different permutations satisfying $\pi(x_j) = y_j$, which yields a contradiction. ■

5.1.1 Transitive Sets of Permutations

The problem of identifying permutations in a set given k input-output correspondences has been extensively studied in combinatorics. A set of permutations is k-transitive if *at least* one permutation is identified when any k input-output correspondences is fixed.

Definition 5.4 *A subset H of $\mathbb{S}_n$ is called* k-transitive *if for every k distinct elements $x_0, \ldots, x_{k-1}$ and non-repeating $y_0, \ldots, y_{k-1}$ of X there exists a permutation $\pi \in H$ such that $\pi(x_j) = y_j$ for all $0 \leq j < k$. If H is k-transitive and is a subgroup of $\mathbb{S}_n$ then H is called a* k-transitive group.

We note that the definition can be applied to any group action. Instead of $\mathbb{S}_n$ we could consider a group G acting on the set X. Then a set $H \subset G$ is k-transitive if for any non-repeating elements $x_0, \ldots, x_{k-1}$ and $y_0, \ldots, y_{k-1}$ of X there exists $h \in H$ such that $h \cdot x_j = y_j$ for all $0 \leq j < k$, where $\cdot$ denotes the group action of G on X. In our definition we use group $G = \mathbb{S}_n$ acting on X which defines the set of all permutations on the set X.

From Prop. 5.3 we see the connection of the definitions of k-transitive set and k-fragile set. Given k input-output correspondences we have that a k-transitive set has at least one permutation that passes through all k correspondences whereas a k-fragile set has at most one permutation matching these k correspondences. In fact, we see in the following that all properties of k-transitivity are symmetric to those of k-fragility. We also point out under which conditions transitivity and fragility are equivalent definitions.

Proposition 5.5 *Let $H \subset \mathbb{S}_n$ is a k-transitive set of permutations. Then $|H| \geq n!/(n-k)!$.*

Proof. Let k non-repeating and fixed elements $x_0, \ldots, x_{k-1} \in X$ be given. Since H is k-transitive, for every k-tuple $(y_0, \ldots, y_{k-1})$ of non-repeating elements of X there must exist at least one permutation $\pi \in H$ such that $\pi(x_j) = y_j$, for all $0 \leq j < k$. Clearly, each of such permutations is different, and hence $|H| \geq n!/(n-k)!$. ■

Definition 5.6 *A subset H of $\mathbb{S}_X$ is called* sharply k-transitive *if H is k-transitive and for every subset $D \subset X$ of k-elements, if there exist permutations $\pi, \pi' \in H$ such that $\pi(x) = \pi'(x)$, for all $x \in D$, then $\pi = \pi'$.*

It follows in a straightforward way that a sharply k-transitive set is k-fragile. More precisely, a sharply k-transitive set is a k-transitive set and a k-fragile set at the same time.

The concepts of k-fragility and k-transitivity are not equivalent. In fact, there exist k-fragile sets of permutation that are not k-transitive. It suffices to consider a singleton set $F = \{\pi\}$ for some $\pi \in \mathbb{S}_n$. The set F is k-fragile, although it is not k-transitive. On the other hand, there exist k-transitive sets that are not k-fragile. As a counterexample, consider the a k-sharply transitive set $F \subset \mathbb{S}_n$ of size $n!/(n-k)!$. By definition, F is k-transitive and k-fragile. Define the set F' by $F' = F \cup \{\pi\}$ where $\pi \in \mathbb{S}_n \backslash F$. This set F' is k-transitive, however, since the size of F' is larger that the maximal possible for k-fragility it clearly cannot be k-fragile.

In the following proposition, we discuss the equivalence between k-fragility and k-transitivity.

Proposition 5.7 *Let H be a subset of $\mathbb{S}_n$ that has $n!/(n-k)!$ elements. H is k-fragile if and only if H is k-transitive.*

Proof. Let $x_0, \ldots, x_{k-1}$ and $y_0, \ldots, y_{k-1}$ be two sets of k different elements of X. On one hand, observe that $y_0, \ldots, y_{k-1}$ is one of the $n!/(n-k)!$ ordered combinations of k distinct elements chosen from the set X. On the other hand, since there are $n!/(n-k)!$ permutations in F there are $n!/(n-k)!$ possible *different* mappings of $x_0, \ldots, x_{k-1}$, otherwise k-fragility and the maximal size of F cannot happen simultaneously. Therefore, there exists a permutation $\pi \in F$ such that $\pi(x_j) = y_j$ for all $0 \leq j < n$.

Suppose that D is a subset of X with k elements on which the k-fragility condition fails. That is, there are two different permutations π and π' that coincide in D. Since H is k-transitive a counting argument implies that $|H| > n!/(n-k)!$ which yields a contradiction. ■

Corollary 5.8 *H is a sharply k-transitive set if and only if H has $n!/(n-k)!$ elements and is k-fragile.*

5.1.2 Basic Sharply Transitive Permutation Sets

We consider trivial examples of sharply transitive sets.

Proposition 5.9 *The set $\mathbb{S}_n$ of all permutations on X is sharply n-transitive and sharply $(n-1)$-transitive.*

Proof. A permutation in $\mathbb{S}_n$ becomes uniquely defined when $n-1$ or n input-output correspondences are given, which proves that it is n and $(n-1)$-fragile. The set $\mathbb{S}_n$ is sharply $(n-1)$-transitive and sharply n-transitive since it is of maximal size $n!$. ■

Proposition 5.10 *The alternating group $A_n \subset \mathbb{S}_n$ is a sharply $(n-2)$-transitive set.*

Proof. First, note that A_n is $(n-2)$-fragile. Given $n-2$ input-to-output correspondences, the two remaining ones are already determined in order to get even parity. Moreover, $|A_n| = n!/2$ which is the maximal size for an $(n-2)$-fragile set of permutations. Thus, A_n is sharply $(n-2)$-transitive. ■

A simple example of a sharply 1-transitive set is the set of all rotations of n elements, the set F_{ROT} defined above. The set F_{SCL} of scaling is another example of sharply 1-transitive set. They are both 1-fragile and of maximal size.

Proposition 5.11 *All sharply 1-transitive sets are Latin squares, and vice versa.*

On the one hand, the set permutations induced by a Latin square is fragile and of maximal size. On the other hand, any sharply 1-transitive set of permutations can be seen as a Latin square.

5.1.3 Affine Transformation

The affine transformation over $\mathbb{Z}_n$ is a function $f : \mathbb{Z}_n \to \mathbb{Z}_n$ parametrized by $a \in \mathbb{Z}_n^*$ and $b \in \mathbb{Z}_n$ defined as

$$f(x) = ax + b \bmod n.$$

Clearly, affine transformations are invertible, the inverse function is given by $f^{-1}(x) = (x-b)a^{-1} \bmod n$. Notice that since f is invertible, it follows that $f \in \mathbb{S}_n$.

Consider the set of affine transformations on $\mathbb{Z}_n$ denoted as $F_{\text{AFF}}(n) = \{\pi_{a,b} \in \mathbb{S}_n : \pi_{a,b}(x) = ax + b \bmod n, a \in \mathbb{Z}_n^*, b \in \mathbb{Z}_n\}$.

Proposition 5.12 *If n is prime, the set $F_{\text{AFF}}(n) \subset \mathbb{S}_n$ is sharply 2-transitive.*

Proof. We show that set $F_{\text{AFF}}(n)$ is 2-fragile using Proposition 5.3. Let x_0, x_1 and y_0, y_1 be two pairs of different elements in $\mathbb{Z}_n$. We have to see that there exists at most one permutation in F_{AFF} linking them. This is, find $a \in \mathbb{Z}_n^*$ and $b \in \mathbb{Z}_n$ such that

$$\begin{cases} ax_0 + b = y_0 \\ ax_1 + b = y_1 \end{cases}$$

The unique solution for this system of equations is $a = (y_0 - y_1)(x_0 - x_1)^{-1} \bmod n$ and $b = y_0 - (y_0 - y_1)(x_0 - x_1)^{-1}x_0 \bmod n$ which always exist if n is prime.

Also, note that $|F_{\text{AFF}}(n)| = n(n-1)$ which is the maximal size of a 2-fragile set. Therefore, $F_{\text{AFF}}(n)$ for prime n is sharply 2-transitive. ■

5.1.4 Möbius Transformation

Consider the projective line of a finite field $\mathbb{F}_q$ denoted with $\overline{\mathbb{F}}_q$. The set $\overline{\mathbb{F}}_q$ is the set $\mathbb{F}_q$ augmented with the point at infinity. For simplicity of exposition, we work in the field $\mathbb{Z}_n$, n prime.

The Möbius transformation over $\mathbb{Z}_n$, with n prime, is a function $f : \overline{\mathbb{Z}}_n \to \overline{\mathbb{Z}}_n$ parametrized by $a, b, c, d \in \mathbb{Z}_n$ such that

$$f(x) = \frac{ax + b}{cx + d},$$

where $ad - bc \neq 0$. The fraction is understood as operations over the projective line $\overline{\mathbb{Z}}_n$. Namely, when $c \neq 0$ the function is defined such that $f(\infty) = a/c$ and $f(-d/c) = \infty$. On the other hand, if $c = 0$ then $f(\infty) = \infty$.

We normalize the representation of a Möbius transformation. Given $f(x) = (ax + b)/(cx + d)$, if $c \neq 0$ we divide c out, getting

$$f(x) = \frac{\frac{a}{c}x + \frac{b}{c}}{x + \frac{c}{d}}.$$

In case $c = 0$, we divide by d, to get $f(x) = (a/d)x + b/d$ which is an affine map over $\overline{\mathbb{Z}}_n$. Therefore, from now on we only consider Möbius transformations parametrized by $a, b, c, d \in \mathbb{Z}_n$ such that $ad - bc \neq 0$, $c \in \{0, 1\}$, and if $c = 0$ then $d = 1$.

The Möbius map f with normalized parameters $a, b, c, d \in \mathbb{Z}_n$ with $c = 1$ can be decomposed in simpler, elementary functions. Namely, consider the functions f_1, f_2, f_3, f_4 defined as follows.

- $f_1(x) = x + d$ (cyclic rotation by d positions),
- $f_2(x) = 1/x$ (inversion),
- $f_3(x) = -(ad - b)x$ (scaling by non-zero factor $-(ad - b)$),
- $f_4(x) = x + a$ (cyclic rotation by a positions).

It can be easily checked that $f = f_4 \circ f_3 \circ f_2 \circ f_1$. Since we have an affine mapping when $c = 0$, it follows that Möbius transformations are invertible, and thus they are a subset of $\mathbb{S}_{n+1}$.

Furthermore, note that if f is a Möbius transformation with parameters $a, b, c, d \in \mathbb{Z}_n$, then f^{-1} is a Möbius transformation as well. In fact, if $c = 0$ then $f^{-1}(x) = a^{-1}x - a^{-1}b$ and if $c = 1$, then $f^{-1}(x) = (-dx + b)/(x - a)$.

Consider the set $F_{\text{MÖB}}(n)$, for n prime, defined as follows:

$$\begin{aligned} F_{\text{MÖB}}(n) = \quad & \{\pi \in \mathbb{S}_{\overline{\mathbb{Z}}_n} : \pi(x) = \tfrac{ax+b}{cx+d}, a, b, c, d \in \mathbb{Z}_n, ad - bc \neq 0, \\ & c \in \{0, 1\}, \text{ if } c = 1 \text{ then } d = 0\}. \end{aligned}$$

Proposition 5.13 *If n is prime, the set $F_{\text{MÖB}}(n) \subset \mathbb{S}_{n+1}$ is sharply 3-transitive.*

To prove this property, we need the following lemma.

Lemma 5.14 *Given non-repeating elements $x_0, x_1, x_2 \in \overline{\mathbb{Z}}_n$, there exists a Möbius transformation f such that $f(x_0) = 0$, $f(x_1) = 1$, and $f(x_2) = \infty$.*

	a	b	c	d
$x_i \in \mathbb{Z}_n$	$x_1 - x_2$	$x_0(x_2 - x_1)$	$x_1 - x_0$	$x_2(x_0 - x_1)$
$x_0 = \infty, x_1, x_2 \in \mathbb{Z}_n$	0	$x_2 - x_1$	-1	x_2
$x_1 = \infty, x_0, x_2 \in \mathbb{Z}_n$	1	$-x_0$	1	$-x_2$
$x_2 = \infty, x_0, x_1 \in \mathbb{Z}_n$	-1	x_0	0	$x_2 - x_1$

Table 5.1: Values of parameters a, b, c, d for the Möbius transformation $f(x) = (ax + d)/(cx + d)$ that passes through the points $(x_0, 0)$, $(x_1, 1)$, and (x_2, ∞).

Proof. We consider the following 4 cases: x_0, x_1, x_2 are all in $\mathbb{Z}_n$, or one of them is ∞. Parameters a, b, c, d up to normalization of the Möbius transformation f satisfying $f(x_0) = 0$, $f(x_1) = 1$, and $f(x_2) = \infty$ are depicted in Table 5.1.

It can be checked that a, b, c, d stated in the table are such that $ad - bc \neq 0$. The way to deduce those values is by noting that $f(x) = (x - x_0)(x_1 - x_2)/(x - x_2)(x_1 - x_0)$ is the function fulfilling the requirement that $f(x_0) = 0$, $f(x_1) = 1$, and $f(x_2) = \infty$ which is a Möbius transformation in all cases. ■

Proof of Prop. 5.13. We show that $F_{\text{MÖB}}(n)$ is a 3-fragile set of maximal size.

We first establish the size of the set $F_{\text{MÖB}}(n)$. Namely, we have to count the number of normalized Möbius transformations in $\overline{\mathbb{Z}}_n$. If $c = 0$ then $d = 1$, so we have to see all possible combinations of $a, b \in \mathbb{Z}_n$ with $a \neq 0$. This gives $n(n-1)$ possible permutations. When $c = 1$, we have to count how many $a, b, d \in \mathbb{Z}_n$ satisfy $ad - b \neq 0$. There are, say, n possible values for each b and d. There is only one value of a that makes $ad - b = 0$, thus there are $n^2(n-1)$ possible permutations in this case. Summing up both cases of c, we have $n(n-1) + n^2(n-1) = (n+1)n(n-1)$ different permutations in $F_{\text{MÖB}}(n)$ which is the maximal allowed size for a 3-fragile set.

Secondly, we prove that $F_{\text{MÖB}}(n)$ is 3-fragile by means of Prop. 5.3. Let x_0, x_1, x_2 and y_0, y_1, y_2 be two triples of non-repeating elements from $\overline{\mathbb{Z}}_n$. Then, according to Lemma 5.14 there exist Möbius transformations f_x and f_y such that $f_x(\{x_0, x_1, x_2\}) = \{0, 1, \infty\}$ and $f_y(\{y_0, y_1, y_2\}) = \{0, 1, \infty\}$. Now note that, the Möbius transformation $g = f_y^{-1} \circ f_x$ satisfies $g(x_i) = y_i$ for $i = 0, 1, 2$. ■

5.1.5 Multiply Sharply Transitive Sets

In the previous sections, we have given examples of sharply 1, 2 and 3-transitive sets of permutations. The natural question that arises is if we can find sharply transitive sets of higher degree, beyond the trivial examples of A_n and $\mathbb{S}_n$. Some existence results are known in the literature. They are included for completeness of the discussion on fragility.

The first distinction that is commonly made has to do with whether the sets of permutations are subgroups of the symmetric group or not. In fact, the set of all rotations, all scalings, all affine transformations, and all Möbius transformations are sharply transitive *groups*. Latin squares are not necessarily groups.

Sharply multiply transitive groups are fully classified. Sharply 1-transitive groups are the groups in which the only permutation fixing points is the identity [Rob95]. A comprehensive classification of all sharply 2-transitive and sharply 3-transitive groups has been made by Zassenhaus. Roughly speaking, all sharply 2-transitive groups are

Group	Order		Transitivity
M_{11}	7920	$= 2^4 \cdot 3^2 \cdot 5 \cdot 11$	sharply 4-transitive
M_{12}	95040	$= 2^6 \cdot 3^3 \cdot 5 \cdot 11$	sharply 5-transitive
M_{22}	443520	$= 2^7 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	3-transitive
M_{23}	10200960	$= 2^7 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 23$	4-transitive
M_{24}	244823040	$= 2^{10} \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 23$	5-transitive

Table 5.2: Mathieu Groups with order and transitivity.

related to the group of affine transformations, and sharply 3-transitivity relates to the Möbius transformations. For more details, we refer to the reader to [Ker74, Rob95, DM96] and references therein.

Interestingly, Jordan [Jor72] proved that there are no sharply k-transitive groups, for $k \geq 4$, other than the alternating and symmetric groups, with the exception of two sporadic groups, the Mathieu groups M_{11} and M_{12}.

Theorem 5.15 ([Jor72]) *Suppose that $k \geq 4$ and let G be a sharply k-transitive group of $\mathbb{S}_n$ which is neither $\mathbb{S}_n$ nor A_n. Then either $k = 4$, $n = 11$, and G is the Mathieu group M_{11}, or $k = 5$, $n = 12$, and G is the Mathieu group M_{12}.*

The groups $M_{11} \subset \mathbb{S}_{11}$ and $M_{12} \subset \mathbb{S}_{12}$ were discovered by Mathieu in 1861. This French mathematician was interested in finding multiply transitive groups. For illustration purposes, Table 5.2 gives some information about some transitive groups of permutations found by Mathieu.

A classification of all sharply k-transitive groups is summarized in the following theorem included in [Pas92].

Theorem 5.16 ([Pas92]) *The following list classifies all sharply k-transitive groups, $k \geq 2$:*

- *The symmetric group $\mathbb{S}_n$ is sharply n and $(n-1)$-transitive.*
- *The alternating group $A_n \subset \mathbb{S}_n$ is sharply $(n-2)$-transitive.*
- *Affine groups of fields or near fields are sharply 2-transitive.*
- *$PGL(2, n) \subset \mathbb{S}_{n+1}$ is sharply 3-transitive.*
- *The subgroup $G = \langle PSL(2, n^2), \sigma \rangle$ of $P\Gamma L(2, n^2) \subset \mathbb{S}_{n^2+1}$ with n odd and σ defined on $PG(2, n^2)$ as $\sigma(x, y) = (x^n, y^n)$ is sharply 3-transitive.*
- *The Mathieu group $M_{11} \subset \mathbb{S}_{11}$ is sharply 4-transitive.*
- *The Mathieu group $M_{12} \subset \mathbb{S}_{12}$ is sharply 5-transitive.*

For a detailed definition of these groups, including notation, we refer the reader to [DM96].

For sharply multiply transitive *sets* the picture is not as complete as for groups, but there are some results as well. For instance, Bonisoli and Quattrocchi [BQ00] present the following results. Here, an invertible set of permutations contains the inverses of all its elements.

Theorem 5.17 ([BQ00]) *The following statements hold.*

- *If G is a sharply 4-transitive set with the identity, then $G = M_{11}$.*
- *If G is a sharply 5-transitive set with the identity, then $G = M_{12}$.*
- *If G is a sharply k-transitive with $k \geq 4$ and G is an invertible set, then G is a group.*
- *There is no sharply 6-transitive set in $\mathbb{S}_{13}$.*

In particular, for $k \geq 4$, all sharply k-transitive invertible sets of permutations are $\mathbb{S}_n$, A_n or the sporadic Mathieu groups M_{11} for $k = 4$ and M_{12} for $k = 5$.

Quistorff [Qui04] shows the following non-existence result for sharply k-transitivity for large k.

Theorem 5.18 ([Qui04]) *If $4 \leq n - k \leq k$ then there is no sharply k-transitive set of permutation in $\mathbb{S}_n$.*

This states that for $k \geq 4$ and $n/2 \leq k \leq n - 4$ there is no sharply k-transitive set. A nice compilation and detail treatment of the classification of fragile sets, we refer the reader to [Rob95, DM96].

5.2 Shuffling according to an Affine Transformation

In the previous section we showed the connection between the definition of fragile sets of permutations and sharply transitive sets of permutations. In this section, we design protocols to show that an affine transformation is applied as a permutation in the application of a shuffle. This way, we achieve a 2-fragile shuffle.

As a stepping stone, we first describe how to prove that a scaling is performed in a shuffle, using our solutions for rotations. It uses a generic conversion which requires no extra cost at all. With this proof as a building block, we later show how to prove an affine transformation in a shuffle.

5.2.1 Scaling Homomorphic Encryptions

We show how to construct a proof for the relation of scaling. That is, a zero-knowledge protocol for the relation R_{SCL} defined as follows:

$$R_{\text{SCL}} = \{(\{[\![x_k]\!]\}_{k\in\mathbb{Z}_n^*}, \{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}; a, \{s_k\}_{k\in\mathbb{Z}_n^*}) : [\![y_{ak}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k), \text{ for } k \in \mathbb{Z}_n^*, a \in \mathbb{Z}_n^*\}.$$

The zero-knowledge proof of knowledge for this relation may be expressed in terms of a proof of rotation for some values of n, using the following well-known result from number theory.

Theorem 5.19 *The multiplicative group $\mathbb{Z}_n^*$ is cyclic if and only if $n = 2, 4, p^\ell$, or $2p^\ell$ for an odd prime p and non-negative integer ℓ.*

In particular, this enables us to say that the multiplicative *cyclic* group $\mathbb{Z}_n^*$ and the additive group $\mathbb{Z}_{\phi(n)}$ are isomorphic where the isomorphism is given by the map $f : \mathbb{Z}_{\phi(n)} \to \mathbb{Z}_n^*$ defined by $f(k) = g^k \bmod n$ where g is a generator of $\mathbb{Z}_n^*$. Here, ϕ denotes the Euler-ϕ function.

Algorithm 5.1 Scaling of Homomorphic Encryptions using Rotations

Input: $\{[\![x_k]\!]\}_{k\in\mathbb{Z}_n^*}$, $a \in \mathbb{Z}_n^*$, generator g of $\mathbb{Z}_n^*$.
Output: $\{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}$ such that $y_k = x_{ak}$.

Phase 1. The following re-ordering is applied to the inputs.

for $k = 0$ to $\phi(n) - 1$ **do**
  $[\![z_k]\!] = [\![x_{g^k}]\!]$
end for

Phase 2 (Rotation). The rotation offset $\log_g a$ is applied to the list $\{[\![z_k]\!]\}_{k=0}^{\phi(n)-1}$.

for $k = 0$ to $\phi(n) - 1$ **do**
  pick $s_k \in_R \mathcal{R}$
  $[\![w_{k+\log_g a}]\!] = [\![z_k]\!]\mathsf{E}(0, s_k)$
end for

Phase 3. Revert re-ordering of Step 1.

for $k = 0$ to $\phi(n) - 1$ **do**
  $[\![y_{g^k}]\!] = [\![w_k]\!]$
end for
return $\{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}$

For the rest of this section we work with n such that $\mathbb{Z}_n^*$ is cyclic. Given a list of $\phi(n)$ homomorphic encryptions, the scaling is done as in Algorithm 5.1. In particular, when n is a prime number, we perform a scaling on a list of homomorphic encryptions of $n-1$ elements.

Proposition 5.20 *Algorithm 5.1 performs a scaling of ciphertexts.*

Proof. This can be verified in a straightforward way as follows. Consider $k \in \mathbb{Z}_n^*$ we have,

$$x_k = z_{\log_g k} = w_{\log_g k + \log_g a} = y_{g^{\log_g k + \log_g a}} = y_{ak}.$$

■

We explain below how Algorithm 5.1 can be used to prove in zero-knowledge that a list of homomorphic encryptions $\{[\![x_k]\!]\}_{k\in\mathbb{Z}_n^*}$ is rescaled into a list $\{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}$. A shuffler executes Algorithm 5.1 on $\{[\![x_k]\!]\}_{k\in\mathbb{Z}_n^*}$, and outputs $\{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}$. Since Phases 1 and 3 only re-order of lists of encryptions according to an agreed upon generator g of $\mathbb{Z}_n^*$, these steps can be done publicly. In fact, the lists $\{[\![z_k]\!]\}_{k=0}^{\phi(n)-1}$ and $\{[\![w_k]\!]\}_{k=0}^{\phi(n)-1}$ are obtained by respectively renaming $[\![z_k]\!] = [\![x_{g^k}]\!]$ and $[\![y_{g^k}]\!] = [\![w_k]\!]$ for all $0 \le k < \phi(n)$. Therefore, verifiability of the scaling is obtained by giving a zero-knowledge proof of knowledge that

$$(\{[\![z_k]\!]\}_{k=0}^{\phi(n)-1}, \{[\![w_k]\!]\}_{k=0}^{\phi(n)-1}; \log_g a, \{s_k\}_{k=0}^{\phi(n)-1}) \in R_{\text{ROT}}.$$

This way, it follows that $(\{[\![x_k]\!]\}_{k\in\mathbb{Z}_n^*}, \{[\![y_k]\!]\}_{k\in\mathbb{Z}_n^*}; a, \{t_k\}_{k\in\mathbb{Z}_n^*}) \in R_{\text{SCL}}$, where $t_{g^k} = s_k$ for $0 \le k < \phi(n)$. In other words, we have traded the proof of scaling in $\mathbb{Z}_n^*$ with a proof of rotation of a list in $\mathbb{Z}_{\phi(n)}$.

Algorithm 5.2 Shuffling Encryptions using an affine transformation

Input: $\{[\![x_k]\!]\}_{k=0}^{n-1}$, $a \in \mathbb{Z}_n^*$, $b \in \mathbb{Z}_n$, n prime.
Output: $\{[\![y_k]\!]\}_{k=0}^{n-1}$ such that $y_k = x_{ak+b}$.

pick $\{\{s_k^{(i)}\}_{k=0}^{n-1}\}_{i=1}^{2}$ at random from $\mathcal{R}$

Phase 1 (Scaling). Applying scaling on input list.

for $k = 1$ to $n-1$ **do**
  $[\![z_{ak}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k^{(1)})$
end for
$[\![z_0]\!] = [\![x_0]\!]$

Phase 2 (Rotation). A rotation is applied to resulting list.

for $k = 0$ to $n-1$ **do**
  $[\![y_{k+b}]\!] = [\![z_k]\!]\mathsf{E}(0, s_k^{(2)})$
end for
return $\{[\![y_k]\!]\}_{k=0}^{n-1}$

Remark 5.21 *Scaling can be performed directly by adopting an approach along the lines of the general protocol for rotation (Protocol 4.2).*

Thus, scaling can be performed on lists of *any* length $\phi(n)$ indexed by $\mathbb{Z}_n^*$, not only for those where $n = 2, 4, p^\ell, 2p^\ell$. However, later in this chapter we will only need to apply scaling on lists of encryptions indexed by $\mathbb{Z}_n^*$ with n prime.

Remark 5.22 *Phase 2 of Algorithm 5.1 can be proved correct in zero-knowledge using the approach based on DFT (Algorithm 3.1) if some mild conditions on the system parameters are satisfied.*

In fact, let $\mathbb{Z}_q$ with q prime be the plaintext of the underlying homomorphic cryptosystem. Then it must hold that $\phi(n) \mid q-1$ so that a rotation of $\phi(n)$ elements can be performed using DFT. In the particular case that n is prime, an $(n-1)$-st root of unity modulo q must exist.

5.2.2 Shuffles using an Affine Transformation

We focus our attention on how to prove that a list of n encryptions has been shuffled using an affine transformation. More specifically, we provide a protocol for the following relation.

$$\begin{aligned} R_{\text{AFF}} \;=\; & \{(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; a, b, \{s_k\}_{k=0}^{n-1}) : \\ & [\![y_{ak+b}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k), \text{ for all } 0 \le k < n,\ a \in \mathbb{Z}_n^*\}. \end{aligned}$$

We assume that n is prime since affine transformations over $\mathbb{Z}_n$ form a sharply 2-transitive group. Algorithm 5.2 presents an approach to shuffle a list of encryptions of length n, with n prime, according to an affine map. The approach is based on the decomposition of an affine transformation into a rotation and scaling. It is straightforward to see that

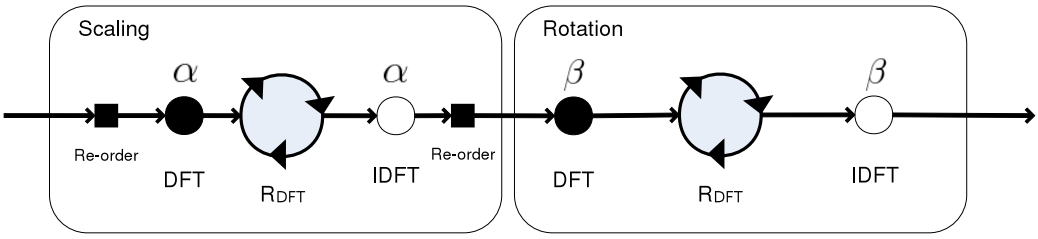


Figure 5.2: Applying an affine transformation using the DFT-based approach.

the algorithm shuffles the list of encryptions according to an affine transformation since $x_k = z_{ak} = y_{ak+b}$, for $0 \leq k < n$.

If a shuffler executes Algorithm 5.2 on a input list of encryptions $\{[\![x_k]\!]\}_{k=0}^{n-1}$ and produces $\{[\![y_k]\!]\}_{k=0}^{n-1}$, then a shuffle according to an affine transformation has been performed. Due to the blinding process in both the scaling and the rotation phases, the applied affine transformation is hidden. The shuffler proves that an affine transformation is linking input and output ciphertexts by releasing the lists of ciphertexts $\{[\![z_k]\!]\}_{k=0}^{n-1}$ and $\{[\![y_k]\!]\}_{k=1}^{n-1}$, and giving zero-knowledge proofs for R_{SCL} and R_{ROT} such that:

$$(\{[\![x_k]\!]\}_{k=1}^{n-1}, \{[\![z_k]\!]\}_{k=1}^{n-1}; a, \{s_k^{(1)}\}_{k=1}^{n-1}) \in R_{\text{SCL}} \text{ and} (\{[\![z_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; b, \{s_k^{(2)}\}_{k=0}^{n-1}) \in R_{\text{ROT}}.$$

Therefore, $(\{[\![x_k]\!]\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; a, b, \{u_k\}_{k=0}^{n-1}) \in R_{\text{AFF}}$, where $u_0 = s_0^{(2)}$ and $u_k = s_k^{(1)} + s_{ak}^{(2)}$ for $1 \leq k < n$.

5.2.3 Performance Analysis

The zero-knowledge proof for R_{SCL} is reduced to prove two rotations, one rotation of a list of $n-1$ encryptions and the other of a list of n elements. Thus the overall complexity is that of these two rotations.

If the approach for rotation based on DFT (Algorithm 3.1) is used for both rotations, some constraints are imposed on the parameters. Namely, if $\mathbb{Z}_q$ for prime q is the message space of the cryptosystem, we require prime numbers n, q be such that $n \mid q-1$ *and* $n-1 \mid q-1$ so that the corresponding roots of unity exist. Performing a parallel composition of both Σ-protocols, the resulting protocol for affine transformation is a Σ-protocol as well. Fig. 5.2 sketches the order in which DFT and inverse DFT must be performed where α is an $(n-1)$-st root of unity and β is an n-th root of unity.

Since one of the rotations is over n elements, FFT for n prime may be applicable. To scale $n-1$ elements, a rotation of $n-1$ is required. If $n-1$ is highly composite then an efficient FFT algorithm is possible as well.

Protocol 4.2 (or Protocol 4.3, for ElGamal encryptions) can be freely used, putting no constraints in the system parameters. Furthermore, we have that the resulting protocol for affine transformation is very efficient since $O(n)$ computation is required as two proofs of rotation on n encryptions are needed.

5.3 Shuffling according to a Möbius Transformation

In this section we focus our attention on a proof of a shuffle in which a Möbius transformation is applied. Suppose n is prime and that we work on the projective line $\overline{\mathbb{Z}}_n$.

$$R_{\text{MÖB}} = \{(\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, \{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}; a, b, c, d, \{s_k\}_{k=0}^{n-1}) : \\ [\![y_{\frac{ak+b}{ck+d}}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k), \text{ for all } k \in \overline{\mathbb{Z}}_n, ad - bc \neq 0, c \in \{0,1\}, c = 0 \Rightarrow d = 1\}.$$

Algorithm 5.3 transforms a list of $n+1$ homomorphic encryptions $\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ into a list $\{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ using a Möbius transform. It is based on the property that Möbius transformations can be splitted into elementary, simpler functions (see Section 5.1.4). The algorithm takes as input two rotation offsets r_1 and r_2, a scaling factor s and a bit e.

Proposition 5.23 *Algorithm 5.3 shuffles the input list of encryptions $\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ according to a Möbius transformation.*

Proof. We show that the resulting re-ordering is indeed a Möbius transformation. Let $r_1, r_2 \in \mathbb{Z}_n$, $s \in \mathbb{Z}_n^*$ and $e \in \{0,1\}$. If $e = 0$ then the shuffling map applied to the list of encryptions is an affine transformation over $\overline{\mathbb{Z}}_n$ defined by $f(x) = sx + sr_1 + r_2$. In particular, $x_\infty = t_\infty = v_\infty = u_\infty = y_\infty$.

If $e = 1$, the Möbius transformation that is applied is given by $f(x) = (r_2x + r_1r_2 + s)/(x + r_1)$. This mapping represents a valid Möbius transformation since $r_2r_1 - r_1r_2 - s \neq 0$ as the scaling factor s is not allowed to be 0. ■

We observe that *any* Möbius transformation can be 'reached' in the shuffle produced by Algorithm 5.3. In fact, if an affine transformation is applied such that $f(x) = ax + b$, then $r_1 = 0, e = 0, s = a$ and $r_2 = b$. If, on the other hand, a Möbius transformation $f(x) = (ax + b)/(x + d)$ with normalized parameters a, b, d is applied, then $r_1 = d, e = 1, s = -(ad - b)$ and $r_2 = a$. This is obtained from the decomposition of Möbius transformations discussed in Section 5.1.4.

5.3.1 Proof of Shuffle using a Möbius Transformation

In order to construct a proof for relation $R_{\text{MÖB}}$ on lists of homomorphic encryptions $\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ and $\{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$, it suffices to verify each of the phases of Algorithm 5.3. Namely, for the auxiliary lists of encryptions $\{[\![t_k]\!]\}_{k=0}^{n-1}$, $\{[\![v_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ and $\{[\![u_k]\!]\}_{k=1}^{n-1}$, the following relations are proved.

- $(\{x_k\}_{k=0}^{n-1}, \{[\![t_k]\!]\}_{k=0}^{n-1}; r_1, \{s_k^{(1)}\}_{k=0}^{n-1}) \in R_{\text{ROT}}$,
- $(\{[\![t_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, \{[\![v_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, id, inv; \pi, \{s_k^{(2)}\}_{k\in\overline{\mathbb{Z}}_n}) \in R_{\text{OR-PERM}}$ (see Section 2.3.2),
- $(\{[\![v_k]\!]\}_{k=1}^{n-1}, \{[\![u_k]\!]\}_{k=1}^{n-1}; s, \{s_k^{(3)}\}_{k=1}^{n-1}) \in R_{\text{SCL}}$
- $(\{u_k\}_{k=0}^{n-1}, \{[\![y_k]\!]\}_{k=0}^{n-1}; r_2, \{s_k^{(4)}\}_{k=0}^{n-1}) \in R_{\text{ROT}}$,

where *id* denotes the identity permutation and *inv* denotes the inverse permutation.

Therefore, we have that $(\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, \{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}; a, b, c, d, \{s_k\}_{k\in\overline{\mathbb{Z}}_n}) \in R_{\text{MÖB}}$ where the following holds. If $(\{[\![t_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, \{[\![v_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, inv; \{s_k^{(2)}\}_{k\in\overline{\mathbb{Z}}_n}) \in R_{\text{PERM}}$, then $a = r_2, b = r_1 r_2 + s, c = 1$, and $d = r_1$. Moreover, $s_\infty = s_\infty^{(2)} + s_0^{(4)}$, $s_{-r_1} = s_0^{(2)} + s_{-r_1}^{(1)}$, and $s_k = s_k^{(1)} + s_{k+r_1}^{(2)} + s_{\frac{1}{k+r_1}}^{(3)} + s_{\frac{s}{k+r_1}}^{(4)}$.

If $(\{[\![t_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, \{[\![v_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, id; \{s_k^{(2)}\}_{k\in\overline{\mathbb{Z}}_n}) \in R_{\text{PERM}}$, then we have that $a = s, b = sr_1 + r_2, c = 0$ and $d = 1$, with $s_\infty = s_\infty^{(2)}$, $s_{-r_1} = s_{-r_1}^{(1)} + s_0^{(2)} + s_0^{(4)}$ and for all other cases $s_k = s_k^{(1)} + s_{k+r_1}^{(2)} + s_{k+r_1}^{(3)} + s_{sk+sr_1}^{(4)}$.

Algorithm 5.3 Shuffling Permutations according to the Möbius Transformation

Input: $\{[\![x_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}, r_1, r_2 \in \mathbb{Z}_n, s \in \mathbb{Z}_n^*, e \in \{0,1\}$
Output: $\{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$ such that $y_{f(k)} = x_k$ for Möbius transformation f.

pick $\{\{s_k^{(i)}\}_{k=0}^{n-1}\}_{i=1,2,4}, s_\infty^{(2)}, \{s_k^{(3)}\}_{k=1}^{n-1}$ at random from $\mathcal{R}$

Phase 1 (Rotation)

for $k = 0$ to $n-1$ **do**
  $[\![t_{k+r_1}]\!] = [\![x_k]\!]\mathsf{E}(0, s_k^{(1)})$
end for
$[\![t_\infty]\!] = [\![x_\infty]\!]$

Phase 2 (Conditional Inversion)

Case $e = 0$**:**

for $k = 0$ to $n-1$ **do**
  $[\![v_k]\!] = [\![t_k]\!]\mathsf{E}(0, s_k^{(2)})$
end for
$[\![v_\infty]\!] = [\![t_\infty]\!]\mathsf{E}(0, s_\infty^{(2)})$

Case $e = 1$**:**

for $k = 1$ to $n-1$ **do**
  $[\![v_{1/k}]\!] = [\![t_k]\!]\mathsf{E}(0, s_k^{(2)})$
end for
$[\![v_0]\!] = [\![t_\infty]\!]\mathsf{E}(0, s_\infty^{(2)})$
$[\![v_\infty]\!] = [\![t_0]\!]\mathsf{E}(0, s_0^{(2)})$

Phase 3 (Scaling)

for $k = 1$ to $n-1$ **do**
  $[\![u_{sk}]\!] = [\![v_k]\!]\mathsf{E}(0, s_k^{(3)})$
end for
$[\![u_0]\!] = [\![v_0]\!]$
$[\![u_\infty]\!] = [\![v_\infty]\!]$

Phase 4 (Rotation)

for $k = 0$ to $n-1$ **do**
  $[\![y_{k+r_2}]\!] = [\![u_k]\!]\mathsf{E}(0, s_k^{(4)})$
end for
$[\![y_\infty]\!] = [\![u_\infty]\!]$
return $\{[\![y_k]\!]\}_{k\in\overline{\mathbb{Z}}_n}$

5.3.2 Selecting a Random Möbius Transformation

So far we have seen that Algorithm 5.3 gives a procedure to shuffle a list of homomorphic encryptions according to a Möbius transformation. Proving that this is the case is not an issue, we basically use previously presented proofs to verify that all steps in Algorithm 5.3 are followed correctly.

We describe now how a shuffler can select a random Möbius transformation using Algorithm 5.3. Naturally, the shuffler can simply enumerate all $(n+1)n(n-1)$ possible Möbius maps and pick a random number between 1 and $(n+1)n(n-1)$ to select one of the transformations. Then the parameters $a, b, c, d \in \mathbb{Z}_n$ of the picked transformation are translated into $r_1, r_2 \in \mathbb{Z}_n$, $s \in \mathbb{Z}_n^*$ and $e \in \{0,1\}$ for Algorithm 5.3.

Note that the alternative of selecting $r_1, r_2 \in \mathbb{Z}_n$, $s \in \mathbb{Z}_n^*$ and $e \in \{0,1\}$ independently and uniformly at random does not necessarily yield a Möbius transformation selected uniformly at random among all possible transformations. Indeed, if e is uniformly selected at random, then it is assigning affine and non-affine transformations with 50% probability. However, there are $n(n-1)$ affine transformations versus $n^2(n-1)$ non-affine ones.

The following selection of r_1, r_2, s, and e does yield a random Möbius map. Draw the offsets r_1, r_2 independently at random from $\mathbb{Z}_n$, select scaling factor s independently at random from $\mathbb{Z}_n^*$, and let bit e be 0 with probability $1/(n+1)$ and 1 with probability $n/(n+1)$.

The intuition of the correctness of the argument is as follows. If $e = 0$ then the Möbius map applied with Algorithm 5.3 is the affine transformation $f_0(x) = sx + (sr_1 + r_2)$ whose parameters are random since $a = s$ is a random non-zero scaling factor and $b = sr_1 + r_2$ is a random offset. In the case $e = 1$, the Möbius transformation applied is $f_1(x) = (r_2x + r_1r_2 + s)/(x + r_1)$ which is a non-affine map, all its parameters $a = r_2, b = r_1r_2 + s$, and $d = r_1$ are randomly chosen. Therefore, the biased election of bit e balances the choices among all possible Möbius transformations.

As a little optimization of the method to select a uniformly random Möbius transformation described above, note that in case $e = 0$ then the choice of r_1 does not influence the distribution of the affine map. Thus, r_2 and s are selected independently and uniformly at random, e is selected independently to be 0 with probability $1/(n+1)$ and 1 with probability $n/(n+1)$. If $e = 1$ then r_1 is drawn uniformly, else r_1 may equal any constant (no further randomness is needed).

5.4 Multiply Fragile Cascades

Since affine and Möbius transformations are closed under composition, they can be composed in a cascade to produce a 2-fragile cascade and a 3-fragile cascade, respectively. The result is that the input and output ciphertexts of the cascade are shuffled according to an affine or Möbius transformation respectively. If all shufflers in the cascade prove that they use a fragile permutation and at least one of them keeps its permutation hidden, the permutation linking input and output ciphertexts is kept secret, and it retains the fragility.

We have seen that in the approaches for affine transformation (Algorithm 5.2) and Möbius transformation (Algorithm 5.3) all intermediate operations can be carried out by means of rotations. More concretely, proving an affine shuffle requires 2 rotations, while

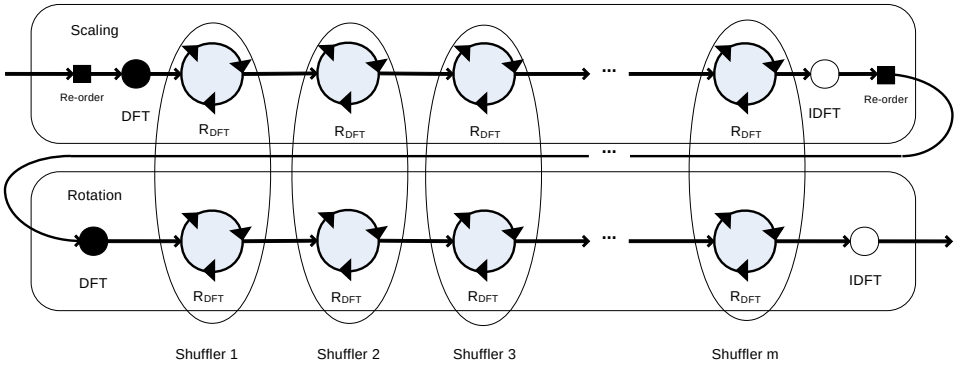


Figure 5.3: A cascade applying an affine transformation.

a Möbius transformation needs 3 rotations. In a cascade, this is repeated by each of the shufflers in the cascade.

In both cases, if the approach for general rotation as described in Chapter 4 is applied the computation per shuffler is $O(n)$. In the case of the approach based on DFT of Chapter 3 the average computation per shuffler is $O(n \log n)$ due to the fact that DFT and inverse DFT must be applied before and after each rotation. Here we assume that FFT is possible for both n-th and $(n-1)$-th root of unity.

In the following, we show how the computation in a cascade can be organized differently. In the case that the DFT approach is used, the computation per shuffler may be amortized over the length of the cascade so that the average work per rotator becomes linear.

5.4.1 Efficient Affine Cascade using DFT

Suppose that in a cascade with m shufflers an affine transformation is applied at any node in the cascade. Since any affine transformation can be splitted in a scaling phase and a rotation phase, the same can be done in the cascade. Namely, in the first stage, there is a sub-cascade of scalings, and in the second phase, there is a sub-cascade of rotators.

This gives some advantages in the DFT-based approach. Applying one affine transformation after the other means that each shuffler must perform two rotations on different roots of unity. Thus, a total of 4 DFTs under the encryptions must be performed per shuffler, meaning a total of $4m$ DFTs for the whole cascade.

By performing a sub-cascade of scalings followed by a sub-cascade of rotators, DFT and inverse DFT only need to be applied at the beginning and at the end of the sub-cascades respectively. Scaling and rotations within every sub-cascade are performed in the transformed domain. This is depicted in Fig. 5.3.

The computational complexity of each shuffler may be amortized over the length of the cascade assuming that FFT of both length n and $n-1$ is possible. There is only 4 FFTs required in total. The cost of proving relation R_{DFT} within the cascade requires $O(n)$ computation. We get an average computational work per shuffler proportional to $(2nm + 4n \log n)/m$. In particular, if the number of shufflers is $\Omega(\log n)$ we have that the average computational work per shuffler is $O(n)$.

An evident drawback of this approach is that every shuffler must act at two different points in time. The secrecy of the end-to-end affine permutation of the full cascade

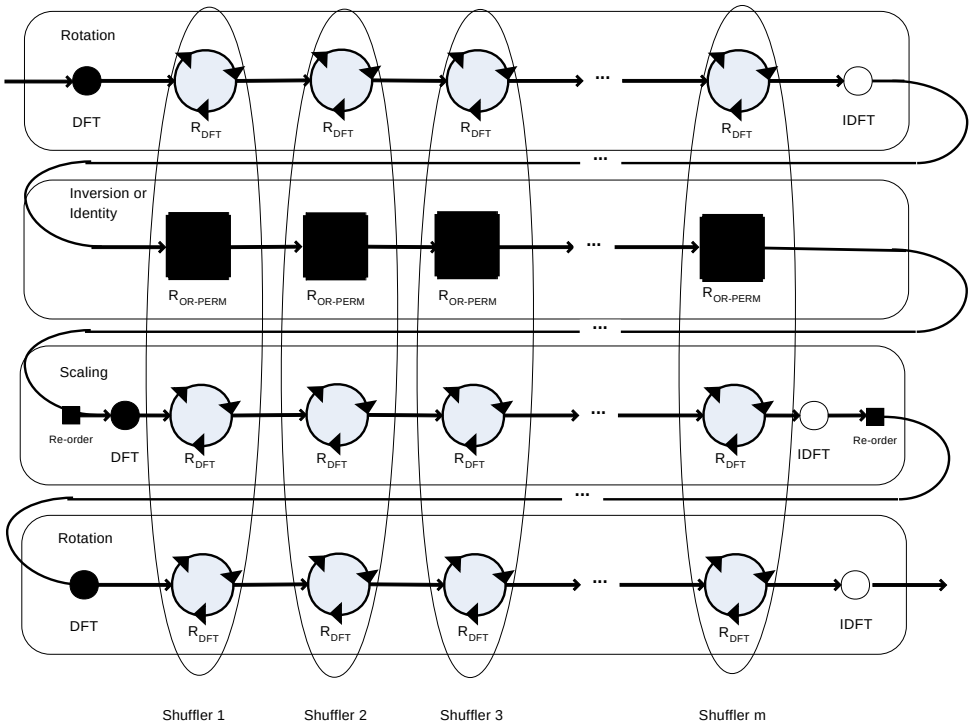


Figure 5.4: A cascade applying a Möbius transformation.

remains 2-fragile assuming that at least one shuffler keeps both its scaling factor and its rotation offset secret during the execution of the cascade.

Furthermore, this construction preserves a nice property of cascades. Namely, if at least one of the shufflers picks both its scaling factor and rotation offset at random, then the overall end-to-end affine transformation that links input and output ciphertexts of the cascade is random.

5.4.2 Efficient Möbius Cascade using DFT

The same idea as with affine cascades can be applied to a cascade of Möbius transformations. Following the idea of Algorithm 5.3, a cascade of shuffles using a Möbius transformation can be divided in 4 sub-cascades that proceed as in each phase of Algorithm 5.3.

As in the case of affine cascade, we observe that the overall Möbius transform can be decomposed in sub-cascades of rotations, conditional inversion, scalings and rotations. In the case that the DFT approach is applied, the sub-cascades using rotations and scalings need to perform DFT and inverse DFT at the beginning and at the end of the sub-cascades, respectively. Thus, only 6 DFTs under the encryptions for the entire Möbius cascade which may amortize the computation per shuffler over the number of shufflers.

As before, the drawback is that every shuffler must be active at four different points in time. Yet, the end-to-end Möbius transformation remains hidden as long as at least one shuffler does not disclose any information about its applied transformations.

Random Möbius transformation

Another drawback of this 4-way cascade is that it may be difficult to get a uniformly random Möbius transformation after the cascade. This is a consequence of the discussion in Section 5.3.2: even though the two rotation offsets and the scaling factor can be uniform assuming at least a honest shuffler throughout the sub-cascades, the distribution of the overall inversion/no-inversion bit is not biased towards the correct distribution in general.

If we assume that all the shufflers are honest, they can all agree to pick each particular bit e following a specific distribution that aggregated yields the desired $1/(n+1)$, $n/(n+1)$ distribution (see Section 5.3.2). If no such an assumption is made, the cascade as is does not withstand malicious shufflers who may choose the bit e from any distribution and thus affecting the distribution of the end-to-end Möbius transformation of the cascade.

This can be amended with the use of secure multiparty computation. An encrypted bit $[\![e]\!]$ can be jointly computed in such a way that that e is randomly chosen from the $1/(n+1)$, $n/(n+1)$ distribution [SS07, Sid07]. Then, the selection between the identity and inverse permutation is done using a multiplication gate per element of the list of encryptions (see Section 6.3.3 for a similar technique). This protocol requires interaction between the parties. However, it just produces an overhead of $O(n)$ computations.

Still, using a cascade of shufflers applying a full and random Möbius transformation at a time will give rise to an unknown *and* uniformly at random end-to-end Möbius transformation. In fact, the uniformity of the end-to-end transformation follows by assuming that one of the shufflers is honest. When we use DFT for the rotations at each step, the average computational cost is $O(n \log n)$ *per* shuffler.

Chapter 6

Integer Comparison

Yao's millionaires' problem [Yao82] is nowadays a classic problem in secure computation: Two millionaires want to compare their wealth and decide who is richer without giving away any other information, in particular, they do not want to disclose to each other how much their assets are worth. Solutions to this problem, usually referred to as integer comparison, are provided by the secure evaluation of the function $\text{GT}(x, y) = [x > y]$. The bracket notation $[B]$, for a condition B, is defined by $[B] = 1$ if B holds and $[B] = 0$ otherwise (this is called Iverson's convention; see [Knu97]).

In this chapter we discuss integer comparison protocols within the framework for secure multiparty computation based on threshold homomorphic cryptosystems put forth by Cramer *et al.* [CDN01]. We focus on the setting in which the inputs x and y are given as encrypted bits of their binary representation, $[\![x_{m-1}]\!], \ldots, [\![x_0]\!]$ and $[\![y_{m-1}]\!], \ldots, [\![y_0]\!]$, with $x = \sum_{i=0}^{m-1} x_i 2^i$, $y = \sum_{i=0}^{m-1} y_i 2^i$. The output is $[\![[x > y]]\!]$. Both inputs and output are available in encrypted form only, the actual values of x and y need not be known to any party. Furthermore, the setting is not limited to two parties.

Our solutions can be classified in two types. First, we present solutions that involve the evaluation of an arithmetic circuit composed of elementary gates as in [CDN01]. The intermediate multiplications of the circuit are performed on encrypted bits which allows us to apply conditional gates from [ST04]. We note that these circuits can be used to get unconditional security if encryptions are replaced by sharings as in [DFK+06].

Secondly, we make use of a more intricate approach. The protocols given here achieve constant rounds assuming that a fixed number of parties execute it. We give a complete description for the two-party setting, although the solutions generalize easily to a setting with more parties. The proofs of security use ideas from [ST06] in which a successful attacker to the protocol is reduced to an attacker of the semantic security of the underlying cryptosystem.

Furthermore, we present further investigations regarding integer comparison and the relation to other problems. Even though these interrelations are part of the folklore of integer comparison, we describe and point out some of them since they may help to achieve more efficient solutions.

6.1 Integer Comparison Circuits

We first recall the linear-depth circuit of [ST04] for computing $[x > y]$, using arithmetic gates only (addition, subtraction, multiplication). The circuit (or, oblivious program) is

fully described by the following recurrence:

$$t_0 = 0, \qquad t_{i+1} = (1 - (x_i - y_i)^2)t_i + x_i(1 - y_i),$$

where t_m is the output bit (hence $t_m = [x > y]$). Rather than starting from the most significant bit as one may intuitively do, this circuit computes $[x > y]$ starting from the least significant bit. The advantage of this approach is that the circuit comprises $2m - 1$ multiplication gates only (compared to about $3m$ multiplication gates when starting from the most significant bit, see [ST04]).

A disadvantage is that the depth of the circuit is m, hence inducing a critical path of m sequential secure multiplications. Clearly, the terms $x_0y_0, \ldots, x_{m-1}y_{m-1}$ can be computed in parallel, but the computation of $t_1, \ldots, t_m$ must be done sequentially.

6.1.1 Our Solution

We show how to reduce the depth of the circuit to roughly $\log m$ at the cost of a slight increase of the circuit size. The idea relies on the following simple but crucial property of integer comparison. Write $x = X_1X_0$ and $y = Y_1Y_0$ as bit strings, where $0 \leq |X_1| = |Y_1| \leq m$ and $0 \leq |X_0| = |Y_0| \leq m$. Then,

$$[x > y] = \begin{cases} [X_1 > Y_1], & X_1 \neq Y_1 \\ [X_0 > Y_0], & X_1 = Y_1, \end{cases}$$

which may be "arithmetized" as

$$[x > y] = [X_1 > Y_1] + [X_1 = Y_1][X_0 > Y_0]. \tag{6.1}$$

This property suggests a circuit that would first split the bit strings x and y in about equally long parts, compare these parts recursively and combine them to produce the final output. Note that in order to achieve this, we need to decide whether the most-significant parts of x and y are equal or not. Following the same recursive divide and conquer reasoning, the equality of bit-strings can be evaluated by first obtaining the result of the equality for the sub-strings. That is,

$$[x = y] = [X_1 = Y_1][X_0 = Y_0]. \tag{6.2}$$

Greater-than and equality comparisons on single bit strings are computed as follows. Given $|x| = |y| = 1$ we have that

$$\begin{aligned} [x > y] &= x(1 - y) = x - xy, \\ [x = y] &= 1 - (x - y)^2 = 1 - x + 2xy - y. \end{aligned}$$

Both comparisons are computed using a single multiplication, xy, combined with linear operations.

Given a bit-string $s = s_{m-1} \ldots s_0$, we let $s_{i,j}$ denote the substring of s of length j composed of the bits $s_{i+j-1} \ldots s_{i+1}s_i$. Let $t_{i,j}$ and $z_{i,j}$ stand for the value of $[x_{i,j} > y_{i,j}]$ and $[x_{i,j} = y_{i,j}]$, respectively. Expressed explicitly in terms of the bits of x and y, a full solution for $[x > y]$ is obtained by evaluating $t_{0,m}$ from the following recurrence relation.

$$\begin{aligned} t_{i,j} &= \begin{cases} x_i - x_iy_i, & j = 1, \\ t_{i+\ell,j-\ell} + z_{i+\ell,j-\ell}t_{i,\ell}, & j > 1, \end{cases} \\ z_{i,j} &= \begin{cases} 1 - x_i + 2x_iy_i - y_i, & j = 1, \\ z_{i+\ell,j-\ell}z_{i,\ell}, & j > 1, \end{cases} \end{aligned} \tag{6.3}$$

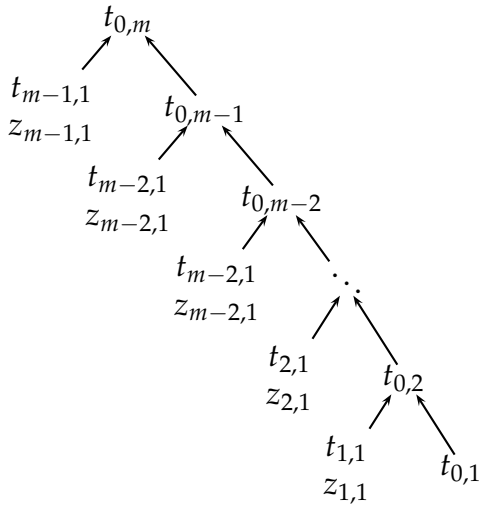


Figure 6.1: This is the diagram of recursion when $\ell = j - 1$. The resulting circuit has size $2m - 1$ and depth m.

where $1 \leq \ell \leq j - 1$.

Correctness of the computation is immediate from the recursion. The security follows assuming secure arithmetic gates as in the setting of Cramer *et al.* [CDN01]. Since multiplication gates work on bits, conditional gates of [ST04] can be used assuming threshold homomorphic ElGamal.

6.1.2 Performance Analysis

The performance parameters of an arithmetic circuit are the size, measured as the number of multiplication gates, and the depth which is associated with the critical path of multiplication gates (see Section 2.4.1).

We first focus on the depth of these circuits. From Eq. (6.3) it can be seen that taking $\ell \approx j/2$ at each recursive invocation, the depth of the resulting circuits is reduced to $\lceil \log m \rceil + 1$. In contrast, taking either $\ell = j - 1$ or $\ell = 1$, the depth of the circuits stretches to m. We note that these are respectively the msb-to-lsb and lsb-to-msb circuits of [ST04].

We turn our attention on the number of multiplication gates. Note that independently of how the recurrence relation is structured (i.e., which ℓ is chosen at each splitting step) the number of recursive steps is always the same. In fact, there are $j - 1$ recurrent calls to evaluate $t_{i,j}$ (resp. $z_{i,j}$). Also, the recurrence relation of $t_{i,j}$ (resp. $z_{i,j}$) will always evaluate the elements $t_{i',1}$ (resp. $z_{i',1}$), for all $i \leq i' < i + j$.

For computing equality of m-bit inputs, the value $z_{0,m}$ uses $m - 1$ recursive steps, each requiring one multiplication gate. Additionally, one multiplication is required for $z_{i,1}$, for all $0 \leq i < m$. Together this adds up to $2m - 1$ multiplication gates.

For greater-than comparison, the evaluation of $t_{i,j}$ requires two multiplications per recursive call. Each leaf $t_{i,1}$ requires one multiplication which can be "recycled" from the computation of $z_{k,1}$. Thus, the number of multiplication gates for $t_{0,m}$ is $3m - 2$.

Trade-offs between Size and Depth

The computation of the greater-than function using Eq. (6.1) tells us that equality of the least-significant part of the inputs is not necessary. This is easy to see in the recurrence relation of Eq. (6.3): the computation of $t_{0,j}$, for any $0 \leq j < m$, does not require the values $z_{0,j'}$ with $j' < j$. Hence, some multiplications can be avoided. The extreme case is taking $\ell = j - 1$ at each recursion step, getting a total of $2m - 1$ multiplication gates for the evaluation of $t_{0,m}$. The recurrence relation of this case is depicted in Fig. 6.1.

Therefore, the more recursive calls to $t_{0,j}$ for $j > 1$ the less invocations to $z_{0,j'}$ are needed which means less multiplication gates. However, we note that the more multiplication gates are avoided this way, the bigger the depth of the circuit is. In the case that $\ell = j - 1$ at each recursion step (Fig. 6.1) we have that the depth of the circuit is m. This clearly leads to a trade-off.

In the following we find an explicit formula for the size of the circuit based on the number of appearances of $t_{0,j}$ in the recurrence of Eq. (6.3). Also, we give a lower bound for the depth of the same circuit.

Proposition 6.1 *Consider a particular recurrence relation of the greater-than circuit of Eq.(6.1). Let d denote the depth and let r be the number of values of the form $t_{0,*}$ of the resulting circuit. Then $r \leq d$ and the size of the circuit is exactly $3m - 2 - r$.*

Proof sketch. The bound $r \leq d$ follows from the fact that all $t_{0,*}$ must be evaluated sequentially. The size of the circuit is analyzed by observing that m multiplication are needed for $t_{*,1}$ (which can be used to compute $z_{*,1}$ eventually), and then $2(m-1)$ for all the recursive splitting. Since all values of $z_{0,*}$ are not needed, we have a total of $2(m-1) + m - r = 3m - 2 - r$ multiplication gates. ■

This means that increasing the number r reduces the size of the circuit while it may increase the depth of the circuit. In particular, when $d = r$ depth and size are optimized. If $d = r$, saving one multiplication (i.e., increasing r) would inevitably increase the depth of the circuit.

This observation is useful in the following scenario. Suppose that the depth of the circuit does not need to be optimized. That is, the depth of the circuit is allowed to be larger than $\lceil \log m \rceil + 1$. This situation may occur when parties have to run certain tasks in parallel to the comparison protocol that require more rounds of interaction than the ones needed for the evaluation of the comparison circuit. Thus, the circuit for integer comparison can be 'stretched' so that the size of the circuit is reduced (i.e., less computation), by increasing the value of r as much as needed, so that the overall round complexity is not increased.

Comparison to Related Work

We compare our arithmetic circuit for bit-wise greater-than with the most efficient greater-than circuits in the literature. This is summarized in Table 6.1.

It can be seen clearly that going from linear to logarithmic depth, means an approximated increase of 50% in the multiplication gates. However, going from logarithmic to constant depth means a penalty of about 6 times more multiplication gates. On the other hand, note that the break-even point of the depth of the circuit happens when $\log m = 8$,

	Size	Depth
Logarithmic depth (Eq. (6.3))	$3m - \log m$	$\log m$
Linear depth [ST04]	$2m - 1$	m
Constant depth [DFK+06, NO07]	$19m$	8

Table 6.1: Performance of the circuits of bit-wise greater-than arithmetic circuits.

meaning that the bit-length of integers must be at least $m = 256$, which means comparing integers larger than 2^{256}. In practical situations one may need to compare integers of $m = 64$ bits for which our logarithmic circuit outperforms the constant depth circuit, especially if we consider the computation.

6.2 Constant Round 2-Party Protocol

In this section we seek to reduce the round complexity to a constant, adopting an approach quite different from the logarithmic depth circuit. We consider the problem of computing $[\![x > y]\!]$ in the two-party case given encryptions of the bit representation of x and y which may be unknown to both parties.

The main idea is to calculate the first position $\hat{k}$ where the bits of x and y differ, starting from the most-significant bit. The position $\hat{k}$ indicates whether $x > y$ or not, since $x_{\hat{k}} - y_{\hat{k}} = 1$ if and only if $x > y$. Position $\hat{k}$ will be determined as the unique index in a list of integers $\{\gamma_k\}_{k=0}^{m-1}$ satisfying $\gamma_{\hat{k}} = 0$ and γ_k is random for $k \neq \hat{k}$. Of course, position $\hat{k}$ leaks information and must be hidden. This is achieved by letting the parties randomly rotate the relevant lists. The output is obtained in encrypted form by using a kind of blinding of the shuffled lists at the end of the protocol.

6.2.1 Our Protocol

Protocol 6.1 presents our solution. We use boxes to depict the actual steps in the protocol of parties A and B, on the left hand side and right hand side respectively. The arrows indicate the messages that are sent over and give an indication of the sequence in which the steps must be performed. Steps that are not in boxes are either operations that can be done based on known values or subroutine calls to other protocols.

In the following we explain the role of each phase in the protocol. Phase 1 determines the position, $\hat{k}$, where the bits of x and y first differ starting from the most-significant bit. List $\{\gamma_k\}_{k=0}^{m-1}$ is such that $\gamma_{\hat{k}} = 0$ and $\gamma_k \neq 0$ for all $k \neq \hat{k}$.

In Phase 2, unknown random numbers are assigned to u_k for $k \neq \hat{k}$, $u_{\hat{k}}$ is set equal to $x_{\hat{k}} - y_{\hat{k}}$. Clearly, $u_{\hat{k}} = 1$ if $x > y$, and $u_{\hat{k}} = -1$ otherwise. Thus, the result of the comparison is actually stored in $u_{\hat{k}}$. Note that giving away $\hat{k}$ leaks information on the inputs.

Phase 3 hides position $\hat{k}$. This is achieved by randomly rotating the list $\{u_k\}_{k=0}^{m-1}$. Note that since all other positions encrypt random numbers, this suffices to conceal position $\hat{k}$. The list $\{v_k\}_{k=0}^{m-1}$ has a position k^* such that $v_{k^*} \in \{-1, 1\}$ contains the answer to the comparison, and v_k is random for all $k \neq k^*$.

Protocol 6.1 Two-Party Greater-Than Comparison

Input: $\{[\![x_k]\!]\}_{k=0}^{m-1}, \{[\![y_k]\!]\}_{k=0}^{m-1}, x_k, y_k \in \{0,1\}$.
Output: $[\![[x > y]]\!]$ where $x = \sum_{i=0}^{m-1} 2^i x_i$ and $y = \sum_{i=0}^{m-1} 2^i y_i$.

Party A	**Party** B
pick $\{\{s_{j,k}^A\}_{k=0}^{m-1}\}_{j=2}^{4}$: $s_{j,k}^A \in_R \mathcal{R}$	**pick** $\{\{s_{j,k}^B\}_{k=0}^{m-1}\}_{j=2}^{4}$: $s_{j,k}^B \in_R \mathcal{R}$
pick $\{t_{2,k}^A\}_{k=0}^{m-1}, t_4^A : t_{2,k}^A, t_4^A \in_R \mathcal{R}$	**pick** $\{t_{2,k}^B\}_{k=0}^{m-1}, t_4^B : t_{2,k}^B, t_4^B \in_R \mathcal{R}$
pick $\{r_k^A\}_{k=0}^{m-1}, r_k^A \in_R \mathcal{M}$	**pick** $\{r_k^B\}_{k=0}^{m-1}, r_k^B \in_R \mathcal{M}$
pick $0 \le r_A < m$ at random	**pick** $0 \le r_B < m$ at random
pick $s_A \in_R \{-1,1\}$	**pick** $s_B \in_R \{-1,1\}$

Phase 1 (Identify interesting position) Parties A and B do the following.

for $k = m-1$ down-to 0 **do**
  run $[\![x_k y_k]\!] \leftarrow \text{MULT}([\![x_k]\!], [\![y_k]\!])$
  $[\![f_k]\!] = [\![[x_k \neq y_k]]\!] = [\![(x_k - y_k)^2]\!] = [\![x_k + y_k - 2x_k y_k]\!]$
  $[\![\gamma_k]\!] = [\![1 + (\sum_{i=k+1}^{m-1} f_i) - f_k]\!]$
end for

Phase 2 (Blind all but interesting position)

for $k = 0$ to $m-1$ **do**		**for** $k = 0$ to $m-1$ **do**
$[\![r_k^A]\!] = \mathsf{E}(r_k^A, t_{2,k}^A)$		$[\![r_k^B]\!] = \mathsf{E}(r_k^B, t_{2,k}^B)$
$[\![u_k^A]\!] = [\![\gamma_k]\!]^{r_k^A} \mathsf{E}(0, s_{2,k}^A)$		$[\![u_k^B]\!] = [\![\gamma_k]\!]^{r_k^B} \mathsf{E}(0, s_{2,k}^B)$
zk-proof [$([\![\gamma_k]\!], [\![u_k^A]\!], [\![r_k^A]\!]; r_k^A, s_{2,k}^A, t_{2,k}^A) \in R_{\text{MULT}}$]	$\{[\![r_k^A]\!]\}_{k=0}^{m-1}$, $\{[\![u_k^A]\!]\}_{k=0}^{m-1}$ $\longrightarrow$	**zk-proof** [$([\![\gamma_k]\!], [\![u_k^B]\!], [\![r_k^B]\!]; r_k^B, s_{2,k}^B, t_{2,k}^B) \in R_{\text{MULT}}$]
end for	$\{[\![r_k^B]\!]\}_{k=0}^{m-1}$, $\{[\![u_k^B]\!]\}_{k=0}^{m-1}$ $\longleftarrow$	**end for**

Party A and B do the following:

for $k = 0$ to $m-1$ **do**
  $[\![u_k]\!] = [\![u_k^A]\!][\![u_k^B]\!][\![x_k - y_k]\!] = [\![u_k^A + u_k^B + (x_k - y_k)]\!]$
end for

Phase 3 (Rotation) Parties A and B in sequence rotate the list $\{[\![u_k]\!]\}_{k=0}^{m-1}$ into $\{[\![v_k]\!]\}_{k=0}^{m-1}$.

for $k = 0$ to $m-1$ **do**		
$[\![u'_{k+r_A}]\!] = [\![u_k]\!]\mathsf{E}(0, s_{3,k}^A)$		
end for	$\{[\![u'_k]\!]\}_{k=0}^{m-1}$ $\longrightarrow$	
zk-proof [$(\{[\![u_k]\!]\}_{k=0}^{m-1}, \{[\![u'_k]\!]\}_{k=0}^{m-1}; r_A, \{s_{3,k}^A\}_{k=0}^{m-1}) \in R_{\text{ROT}}$]		
		for $k = 0$ to $m-1$ **do**
	$\{[\![v_k]\!]\}_{k=0}^{m-1}$ $\longleftarrow$	$[\![v_{k+r_B}]\!] = [\![u'_k]\!]\mathsf{E}(0, s_{3,k}^B)$
		end for
		zk-proof [$(\{[\![u'_k]\!]\}_{k=0}^{m-1}, \{[\![v_k]\!]\}_{k=0}^{m-1}; r_B, \{s_{3,k}^B\}_{k=0}^{m-1}) \in R_{\text{ROT}}$]

Protocol 6.1 (continued)

Phase 4 (Hiding Comparison Output)

$[\![s_A]\!] = \mathsf{E}(s_A, t_4^A)$
for $k = 0$ to $m-1$ **do**
  $[\![v'_k]\!] = [\![v_k]\!]^{s_A}\mathsf{E}(0, s_{4,k}^A)$
  zk-proof [$([\![v_k]\!], [\![v'_k]\!], [\![s_A]\!]; s_A, s_{4,k}^A, t_4^A) \in R_{\text{PRV-MLT}}$]
end for
zk-proof [$([\![s_A]\!], -1, 1; s_A, t_4^A) \in R_{\binom{2}{1}\text{KNW}}$]

$[\![s_A]\!],\ \{[\![v'_k]\!]\}_{k=0}^{m-1}$ ⟶

$[\![s_B]\!] = \mathsf{E}(s_B, t_4^B)$
for $k = 0$ to $m-1$ **do**
  $[\![w_k]\!] = [\![v'_k]\!]^{s_B}\mathsf{E}(0, s_{4,k}^B)$
  zk-proof [$([\![v'_k]\!], [\![w_k]\!], [\![s_B]\!]; s_B, s_{4,k}^B, t_4^B) \in R_{\text{PRV-MLT}}$]
end for
zk-proof [$([\![s_B]\!], -1, 1; s_B, t_4^B) \in R_{\binom{2}{1}\text{KNW}}$]

⟵ $[\![s_B]\!],\ \{[\![w_k]\!]\}_{k=0}^{m-1}$

Phase 5 (Decryption) Party A and B do the following:

for $k = 0$ to $m-1$ **do**
  run $w_k \leftarrow \text{DECR}([\![w_k]\!])$
end for
find k^* such that $w_{k^*} \in \{-1, 1\}$
return $[\![(v_{k^*}+1)/2]\!]$

In order to achieve encrypted output, in Phase 4 every member of list $\{v_k\}_{k=0}^{m-1}$ is multiplied by an unknown random element $s \in \{-1, 1\}$ yielding list $\{w_k\}_{k=0}^{m-1}$. This is achieved by letting each party multiply, one after the other, list $\{v_k\}_{k=0}^{m-1}$ by random elements $s_A, s_B \in \{-1, 1\}$. That way w_{k^*} conceals v_{k^*}.

Finally, index k^* is determined in Phase 5 by revealing the values $\{w_k\}_{k=0}^{m-1}$. The output of the protocol is essentially v_{k^*} which is blinded in w_{k^*}. The value v_{k^*} is either -1 or 1, hence $(v_{k^*}+1)/2$ is either 0 or 1, and gives the bit of information about $x > y$. This linear transformation can be done for free using homomorphic properties.

Note that the position $\hat{k}$ such that $\gamma_{\hat{k}} = 0$ is guaranteed to exist only if x and y are different. We add some 'sentinels' to deal with the case that $x = y$. In fact, we define $f_{-1} = 1$ and $[\![u_{-1}]\!] = [\![u_{-1}^A + u_{-1}^B - 1]\!]$. The rest of the protocol is adapted accordingly.

Related Work

The approach of used in Protocol 6.1 resembles the protocol for conditional oblivious transfer of Blake and Kolesnikov [BK04]. Ever since, some protocols for integer comparison have been presented following more or less the same idea (e.g., [DGK07, RT09, GSV07]).

The differences of our solution with [BK04] include the fact that we allow for encrypted inputs, rather than private inputs. Accordingly, we use a (2,2)-threshold homomorphic cryptosystem instead of just a homomorphic cryptosystem, and multiplications

gates are used. Furthermore, the specific kind of blinding at the end of the protocol allows the extraction of the outcome of the integer comparison in encrypted form. As a further important difference, we can actually use homomorphic cryptosystems such as ElGamal since the multiplication gates work on bits (that is, conditional gates of [ST04] can be used) and decryption checks if encrypted values are in a two-value domain or not. In contrast, [BK04] makes essential use of Paillier.

Damgård *et al.* [DGK07] present protocol following a similar approach as ours but the main difference is the setting: one of the inputs is known to one of the parties, and the output of the comparison is delivered in the clear. The definition of the list $\{\gamma_k\}_{k=0}^{m-1}$ of Phase 1 is due to Damgård *et al.* [DGK07]. In fact, the list $\{\gamma_k\}_{k=0}^{m-1}$ grows linearly in m unlike exponential growth in [BK04, GSV07]. The advantage is that the protocol can be based on a cryptosystem with relatively small plaintext space.

Phase 3 is a common step in secure protocols for comparison following this approach (e.g., [BK04, ABFL06, DGK07, GSV07, RT09]) in which an unknown permutation of the elements of the list $\{u_k\}_{k=0}^{m-1}$ is applied. This is achieved in [BK04, ABFL06, DGK07, GSV07] by letting the parties sequentially apply verifiable shuffles. Note, however, that applying an unknown rotation produces the same effect as a general permutation. In the list $\{u_k\}_{k=0}^{m-1}$ all but one position are random numbers, the rotation suffices to achieve the goal of hiding position $\hat{k}$. This was first noted by Reistad and Toft [RT09], although their solution works in a complete different context than ours. Note that in general any 1-fragile permutation may be applied in order to get the effect of hiding position $\hat{k}$.

The protocol in the semi-honest case only needs multiplication gates in Phase 1. The rest of the protocol involves arithmetic operations over encryptions and randomized encryptions. At the decryption step in Phase 5, only random numbers are disclosed with the additional notice that the position where -1 or 1 leaks no information (due to rotation of Phase 3, and hiding in Phase 4). In order to withstand active adversaries, (non-interactive) proofs of knowledge are attached to each action performed by the parties.

6.2.2 Security Analysis

For the proof of security, we want to be able to simulate this protocol assuming that one of the participants is corrupted. The idea is to give the simulator the inputs $\{[\![x_k]\!]\}_{k=0}^{m-1}$ and $\{[\![y_k]\!]\}_{k=0}^{m-1}$ in such a way that a consistent view of the protocol can be constructed without making use of the private information of the honest party.

Simulation of Building Blocks

We review the simulation requirements for the building blocks, as discussed in Section 2.4.3. The simulator $\mathcal{S}_{\text{MULT}}$ needs encryptions $[\![x]\!]$ and $[\![y]\!]$ and $[\![xy]\!]$ in order to simulate a multiplication gate. The same holds for the simulator of the threshold decryption protocol. Given encryption $[\![x]\!]$ and plaintext x, simulator $\mathcal{S}_{\text{DECR}}([\![x]\!], x)$ gives a statistically indistinguishable transcript.

We use zero-knowledge proofs of knowledge for relations $R_{\text{PRV-MLT}}$, R_{ROT}, and $R_{\binom{2}{1}\text{KNW}}$. We use their simulators to generate an indistinguishable view. The witness-extended emulator for each of these proofs gives both an identical transcript and the witnesses for such relations.

Simulation of Protocol 6.1

We now turn our attention to the overall simulation strategy. We note that a problem arises when trying to simulate the multiplication gates in the first phase of the protocol. In order to simulate the multiplication gates, the simulator has to produce $[\![x_k y_k]\!]$ only given $[\![x_k]\!]$ and $[\![y_k]\!]$, which is impossible. We circumvent such problems by adopting the approach introduced in [ST06], in which a simulation for input/output of a special form (see Theorem 6.2 below) suffices to ensure integration with the framework of [CDN01]. This is a consequence of the fact that the security proof in this framework centers around the construction of a so-called *YAD^b distribution*, which is defined as a function of an encrypted bit $[\![b]\!]$.

The structure of the security proof in [CDN01] follows an ideal-model/real-model approach. The YAD^0 distribution is identical to the distribution of the ideal case, whereas the YAD^1 distribution is statistically indistinguishable from the distribution in the real case. Therefore, if an adversary can distinguish between the ideal/real cases, it implies that the adversary can distinguish the YAD^0 distribution from the YAD^1 distribution. But the choice between these two distributions is determined by the value of an encrypted bit b. It follows that the distinguisher for the ideal/real cases is a distinguisher for the underlying encryption scheme. This is done in a tight way, i.e., without loss in the success probability for the distinguisher. See [CDN01, Full version] for more details.

It is sufficient to show a simulation for inputs of a special form, namely, $[\![\tilde{x}]\!] = [\![(1-b)x^{(0)} + bx^{(1)}]\!]$, where $x^{(0)}$ and $x^{(1)}$ are given in the clear to the simulator, but b is only given in encrypted form $[\![b]\!]$. The values $x^{(0)}$ and $x^{(1)}$ correspond to the values arising in the YAD^0 and YAD^1 cases, respectively.

Theorem 6.2 *Given input values $\{x_k^{(0)}\}_{k=0}^{m-1}$, $\{y_k^{(0)}\}_{k=0}^{m-1}$, $\{x_k^{(1)}\}_{k=0}^{m-1}$ and $\{y_k^{(1)}\}_{k=0}^{m-1}$ and an encryption $[\![b]\!]$ with $b \in \{0,1\}$ Protocol 6.1 can be simulated statistically for inputs $\{[\![\tilde{x}_k]\!]\}_{k=0}^{m-1}$ and $\{[\![\tilde{y}_k]\!]\}_{k=0}^{m-1}$ where $[\![\tilde{x}_k]\!] = [\![(1-b)x_k^{(0)} + bx_k^{(1)}]\!]$ and $[\![\tilde{y}_k]\!] = [\![(1-b)y_k^{(0)} + by_k^{(1)}]\!]$.*

Proof. Let $\{x_k^{(0)}\}_{k=0}^{m-1}$, $\{y_k^{(0)}\}_{k=0}^{m-1}$, $\{x_k^{(1)}\}_{k=0}^{m-1}$ and $\{y_k^{(1)}\}_{k=0}^{m-1}$ be binary representations of $x^{(0)}$, $y^{(0)}$, $x^{(1)}$, $y^{(1)}$ respectively, and let the encryption $[\![b]\!]$ with $b \in \{0,1\}$ be given. The simulation is given in Algorithm 6.1 where party A^* is assumed to be corrupted.

The description of Algorithm 6.1 does not include some checks that are done during the execution, just as an honest party will always do in Protocol 6.1. That is, the simulator must check that the zero-knowledge proofs of knowledge are accepting. Otherwise, it means that the other party is corrupted and it is deviating from the protocol description. In fact, Algorithm 6.1 aborts if after running the witness-extended emulator of certain proof it got an invalid witness. For simplicity of presentation, these checks are left out.

We now briefly explain each step of the simulation.

1. The simulation for Phase 1 relies on the simulator for multiplication gates. Lists f and γ are prepared for the two sets of inputs, and the multiplication gate is simulated depending on the value of the bit b. This simulation gives a statistically indistinguishable transcript of the protocol.

2. In phase 2, party A^* gives list $\tilde{u}^A$ for which the blinding factors are extracted using the witness-extended emulator of the proof for relation $R_{\text{PRV-MLT}}$. A transcript of this conversation is also provided by the emulator. Note that, because of the properties

Algorithm 6.1 Simulator for Protocol 6.1

Input: $\{x_k^{(0)}\}_{k=0}^{m-1}, \{y_k^{(0)}\}_{k=0}^{m-1}, [\![b]\!]$
for $k = 0$ to $m-1$ **do**
$[\![\tilde{x}_k]\!] = [\![(1-b)x_k^{(0)} + bx_k^{(1)}]\!]$
$[\![\tilde{y}_k]\!] = [\![(1-b)y_k^{(0)} + by_k^{(1)}]\!]$
end for

Phase 1.

for $k = m-1$ down-to 0 **do**
print $\mathcal{S}_{\text{MULT}}([\![\tilde{x}_k]\!], [\![\tilde{y}_k]\!], [\![(1-b)x^{(0)}y^{(0)} + bx^{(1)}y^{(1)}]\!])$
end for
for $j = 0, 1$ **do**
for $k = m-1$ down-to 0 **do**
$f_k^{(j)} = [x_k^{(j)} \neq y_k^{(j)}]$
$\gamma_k^{(j)} = 1 + (\sum_{i=k+1}^{m-1} f_i^{(j)}) - f_k^{(j)}$
end for
let k_j s.t. $\gamma_{k_j}^{(j)} = 0$
end for
$\theta = k_1 - k_0$
for $k = 0$ to $m-1$ **do**
$[\![\tilde{\gamma}_k]\!] = [\![(1-b)\gamma_k^{(0)} + b\gamma_k^{(1)}]\!]$
end for

Phase 2.

Party A^* gives $\{[\![\tilde{r}_k^A]\!]\}_{k=0}^{m-1}, \{[\![\tilde{u}_k^A]\!]\}_{k=0}^{m-1}$
for $k = 0$ to $m-1$ **do**
run $(\mathtt{tr}, (\tilde{r}_k^A, s_{2,k}^A, t_{2,k}^A)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![\tilde{\gamma}_k]\!], [\![\tilde{u}_k^A]\!], [\![\tilde{r}_k^A]\!])$
print $\mathtt{tr}$
end for
pick $s_B^{(0)} \in_R \{-1, 1\}$
$s_B^{(1)} = (-1)^{[[x^{(0)} > y^{(0)}] \neq [x^{(1)} > y^{(1)}]]} s_B^{(0)}$
for $k = 0$ to $m-1$ **do**
pick $r_k^{B(0)} \in_R \mathcal{M}$
find $r_k^{B(1)}$ s.t. $s_B^{(0)}\big((\tilde{r}_k^A + r_k^{B(0)})\gamma_k^{(0)} + (x_k^{(0)} - y_k^{(0)})\big) = s_B^{(1)}\big((\tilde{r}_{k+\theta}^A + r_{k+\theta}^{B(1)})\gamma_{k+\theta}^{(1)} + (x_{k+\theta}^{(1)} - y_{k+\theta}^{(1)})\big)$
for $j = 0, 1$ **do**
$u_k^{B(j)} = r_k^{B(j)}\gamma_k^{(j)}$
end for
$[\![\tilde{r}_k^B]\!] = [\![(1-b)r_k^{B(0)} + br_k^{B(1)}]\!]$
$[\![\tilde{u}_k^B]\!] = [\![(1-b)u_k^{B(0)} + bu_k^{B(1)}]\!]$
print $[\![\tilde{r}_k^B]\!], [\![\tilde{u}_k^B]\!], \mathcal{S}_{\text{PRV-MLT}}([\![\tilde{\gamma}_k]\!], [\![\tilde{u}_k^B]\!], [\![\tilde{r}_k^B]\!])$
$[\![\tilde{u}_k]\!] = [\![\tilde{u}_k^A + \tilde{u}_k^B + (\tilde{x}_k - \tilde{y}_k)]\!]$
end for

Algorithm 6.1 (continued)

Phase 3.

Party A^* gives $\{[\![\tilde{u}'_k]\!]\}_{k=0}^{m-1}$
run $(\mathtt{tr}, (\tilde{r}_A, \{s^A_{3,k}\}_{k=0}^{m-1})) \leftarrow \mathcal{W}_{\mathrm{ROT}}(\{[\![\tilde{u}_k]\!]\}_{k=0}^{m-1}, \{[\![\tilde{u}'_k]\!]\}_{k=0}^{m-1})$
print $\mathtt{tr}$
pick $\tilde{k}^* \in_R \{0, \ldots, m-1\}$
find $r_B^{(0)}, r_B^{(1)}$ s.t. $\tilde{k}^* = k_0 + \tilde{r}_A + r_B^{(0)} = k_1 + \tilde{r}_A + r_B^{(1)}$
for $j = 0, 1$ **do**
  for $k = 0$ to $m-1$ **do**
    $v^{(j)}_{k+\tilde{r}_A+r_B^{(j)}} = u_k^{(j)}$
  end for
end for
for $k = 0$ to $m-1$ **do**
  pick $s^B_{3,k} \in_R \mathcal{R}$
  $[\![\tilde{v}_k]\!] = [\![(1-b)v_k^{(0)} + bv_k^{(1)}]\!]\mathsf{E}(0, s^B_{3,k})$
end for
print $\{[\![\tilde{v}_k]\!]\}_{k=0}^{m-1}, \mathcal{S}_{\mathrm{ROT}}(\{[\![\tilde{u}'_k]\!]\}_{k=0}^{m-1}, \{[\![\tilde{v}_k]\!]\}_{k=0}^{m-1})$

Phase 4.

Party A^* gives $[\![\tilde{s}_A]\!]$, $\{[\![\tilde{v}'_k]\!]\}_{k=0}^{m-1}$
run $(\mathtt{tr}, (\tilde{s}_A, t_4^A)) \leftarrow \mathcal{W}_{\binom{2}{1}\mathrm{KNW}}([\![\tilde{s}_A]\!], -1, 1)$
print $\mathtt{tr}$
for $k = 0$ to $m-1$ **do**
  run $(\mathtt{tr}, (\tilde{s}_A, s^A_{4,k}, t^A_{4,k})) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![\tilde{v}_k]\!], [\![\tilde{v}'_k]\!], [\![\tilde{s}_A]\!])$
  print $\mathtt{tr}$
end for
$[\![\tilde{s}_B]\!] = [\![(1-b)s_B^{(0)} + bs_B^{(1)}]\!]$
print $[\![\tilde{s}_B]\!], \mathcal{S}_{\binom{2}{1}\mathrm{KNW}}([\![\tilde{s}_B]\!], -1, 1)$
for $k = 0$ to $m-1$ **do**
  $\tilde{w}_k = \tilde{s}_A s_B^{(0)} v_k^{(0)}$
  pick $s^B_{4,k} \in_R \mathcal{R}$
  $[\![\tilde{w}_k]\!] = \mathsf{E}(\tilde{w}_k, s^B_{4,k})$
  print $[\![\tilde{w}_k]\!], \mathcal{S}_{\text{PRV-MLT}}([\![\tilde{v}'_k]\!], [\![\tilde{w}_k]\!], [\![\tilde{s}_B]\!])$
end for

Phase 5.

for $k = 0$ to $m-1$ **do**
  print $\mathcal{S}_{\mathrm{DECR}}([\![\tilde{w}_k]\!], \tilde{w}_k)$
end for

of the witness-extended emulator, the witness is given with the same probability as A^* would succeed in a real execution of the protocol. That is, if no witness is extracted, the simulation simply aborts (as honest party B would do in a real execution of the protocol). Although is not stated in the description of the simulation, this check is done at every witness-extended emulation call.

Also, Algorithm 6.1 is accommodating the values within the simulation. Since the last step require some decryptions, the simulator needs to know the plaintexts of the corresponding ciphertexts. Notice that after Phase 2 is finished, the lists $u^{(0)}$ and $u^{(1)}$ contain the same values up to a rotation with offset θ. The values in the list $\tilde{u}$ depend on the encrypted bit b, but independently of it, the distribution of the values is exactly the same as the distribution of the values of the list u in the protocol. The simulator for $R_{\text{PRV-MLT}}$ is invoked to simulate the computation of list $\tilde{u}^B$.

3. Next, party A^* gives a list of encryptions that is rotated if the witness-extended emulator gives a valid witness. Also, the rotation step of honest party B is simulated. Here, the rotation offsets are chosen especially to let the list $\tilde{v}$ have the same values no matter what the value of the encrypted bit b is.

4. As for phase 4, the blinding factor $\tilde{s}_A$ is extracted with the witness-extended emulator. The simulator also checks that $\tilde{s}_A$ is either -1 or 1. At the end, the blinding step is simulated accordingly. Additionally, a simulation for $R_{\binom{2}{1}\text{KNW}}$ is given for the fact that $\tilde{s}_B$ is either -1 or 1.

5. In phase 5 threshold decryption is simulated. This is possible due to the fact that the simulator knows the value $\tilde{w}_k$ of the corresponding ciphertext $[\![\tilde{w}_k]\!]$.

The values generated in this way by the simulator follow the same distribution as the values in the real protocol execution. Hence, an adversary cannot statistically distinguish them from the ones resulting in a real execution. This completes the proof of the theorem. ■

6.2.3 Variations

In this section we analyze different modifications of Protocol 6.1. In particular, we see how to adapt it to provide equality test and public output.

Equality Test

Protocol 6.1 assumes that $x \neq y$, so that the position $\hat{k}$ such that $x_{\hat{k}} \neq y_{\hat{k}}$ starting from the most significant bit is well-defined. We have discussed earlier how to modify the protocol to handle a possible equality of the inputs by using some sentinels. The same idea is applied to compute the bit saying whether $x = y$.

We modify Protocol 6.1 as follows. In Phase 1, we define $f_{-1} = 1$, while in Phase 2, we redefine $\{u_k\}_{k=-1}^{m-1}$ such that $u_k = u_k^A + u_k^B - 1$ for $0 \leq k < m$, and $u_{-1} = u_{-1}^A + u_{-1}^B + 1$. The rest of the protocol is adapted in accordance with these modifications.

We argue that the resulting protocol outputs $[\![[x = y]]\!]$. If $x = y$ then $\gamma_k = 1$ for $0 \leq k < m$ and $\gamma_{-1} = 0$. This means that u_k is random for all $0 \leq k < n$ and $u_{-1} = 1$. In

case $x \neq y$, there exists $\hat{k}$ with $0 \leq \hat{k} < m$ such that $\gamma_{\hat{k}} = 0$ and $\gamma_k \neq 0$ for $k \neq \hat{k}$. Hence, $u_{\hat{k}} = -1$ while u_k is random for all $k \neq \hat{k}$. Whether the list $\{u_k\}_{k=-1}^{m-1}$ has a 1 or -1 among otherwise random values tells if $x = y$ or not, respectively. Phases 3 and 4 of Protocol 6.1 hide any other information about this fact.

One Input is Public

Suppose now that, say, x is an integer that is known and wants to be compared with y which is given in bit-wise encrypted form, i.e., we are given encryptions $\{[\![y_k]\!]\}_{k=0}^{m-1}$. Some multiplication gates can be avoided with this configuration. Namely, the m multiplications for $x_k y_k$ of Phase 1 do not need to be computed with a multiplication protocol. Private multiplication can be used instead.

Note that the same observation applies to the circuit of Eq. (6.3) where all multiplications $x_k y_k$ can be replaced by private multiplications where the party knowing its local inputs computes these multiplications.

Public Output

In some applications, parties comparing two encrypted numbers want to know the result of the comparison right away after the computation. In that situation, Protocol 6.1 can be adapted to deliver the result of the comparison in the clear.

This is done by omitting Phase 4, and the parties directly decrypt the list of encryption $\{[\![v_k]\!]\}_{k=0}^{m-1}$ in order to find the unique position $\hat{k}$ such that $v_{\hat{k}} \in \{-1, 1\}$. $v_{\hat{k}} = -1$ means that $x \leq y$, while and $v_{\hat{k}} = 1$ means that $x > y$.

6.2.4 Optimized Protocols

We now look at different optimizations of Protocol 6.1. Note that we can optimize the round complexity by simply changing the order in which the different steps are executed. For instance, Phase 3 and 4 can be overlapped in one phase in which each party applies rotation (Phase 3) and the blinding factors (Phase 4). Table 6.2 uses this observation to compute the round complexity.

Public Output

We analyze other alternatives but the core idea of the protocols is the same. We first present Protocol 6.2 that performs a comparison with public output. Differences with Protocol 6.1 include the order in which the different phases are executed and the way that the list f is computed. Interestingly, no multiplication gates are needed, yielding a 3-round protocol.

Protocol 6.2 identifies position $\hat{k}$ in the first round in which $\gamma_{\hat{k}} = 0$ if and only if $x > y$. Party A applies a non-zero blinding factor and rotates the list of encrypted γ's. Party B reblinds and rotates the same list of γ's. If in the decryption step a unique zero is found, it means that $x > y$, otherwise $x \leq y$.

The protocol requires a cryptosystem with exponentially large plaintext to avoid a possible wrap-around of the list $\{\gamma_k\}_{k=0}^{m-1}$. Since the reblinding factors r_k^A and r_k^B are used in a multiplicative way, they must be non-zero. In addition, no full decryption is needed, but just zero/non-zero plaintext checking. Hence, homomorphic ElGamal is an

Protocol 6.2 Two-Party Greater-Than Comparison with Public Output

Input: $\{[\![x_k]\!]\}_{k=0}^{m-1}, \{[\![y_k]\!]\}_{k=0}^{m-1}, x_k, y_k \in \{0,1\}$.
Output: $[x > y]$.

Party A	**Party** B
pick $\{\{s_{j,k}^A\}_{k=0}^{m-1}\}_{j=1}^{2} : s_{j,k}^A \in_R \mathcal{R}$	**pick** $\{\{s_{j,k}^B\}_{k=0}^{m-1}\}_{j=1}^{2} : s_{j,k}^B \in_R \mathcal{R}$
pick $\{t_k^A\}_{k=0}^{m-1} : t_k^A \in_R \mathcal{R}$	**pick** $\{t_k^B\}_{k=0}^{m-1} : t_k^B \in_R \mathcal{R}$
pick $\{r_k^A\}_{k=0}^{m-1} : r_k^A \in_R \mathcal{M}\backslash\{0\}$	**pick** $\{r_k^B\}_{k=0}^{m-1}, r_k^B \in_R \mathcal{M}\backslash\{0\}$
pick $r_A \in_R \{0,\ldots,m-1\}$	**pick** $r_B \in_R \{0,\ldots,m-1\}$

$\delta_m = 0$
for $k = m-1$ down-to 0 **do**
  $[\![\delta_k]\!] = [\![3\delta_{k+1} + (x_k - y_k)]\!]$
  $[\![\gamma_k]\!] = [\![\delta_k - 1]\!]$
end for

for $k = m-1$ down-to 0 **do**
  $[\![r_k^A]\!] = \mathsf{E}(r_k^A, t_k^A)$
  $[\![u_k]\!] = [\![\gamma_k]\!]^{r_k^A}\mathsf{E}(0, s_{1,k}^A)$
  zk-proof [
  $([\![r_k^A]\!]; r_k^A, t_k^A) \in R_{\text{NO-ZERO}}$
  $([\![\gamma_k]\!], [\![u_k]\!], [\![r_k^A]\!]; r_k^A, s_{1,k}^A, t_k^A) \in R_{\text{PRV-MLT}}$]
  $[\![v_{k+r_A}^A]\!] = [\![u_k]\!]\mathsf{E}(0, s_{2,k}^A)$
end for
zk-proof [
$(\{[\![u_k]\!]\}_{k=0}^{m-1}, \{[\![v_k^A]\!]\}_{k=0}^{m-1}; r_A, \{s_{2,k}^A\}_{k=0}^{m-1}) \in R_{\text{ROT}}$]

$\{[\![r_k^A]\!]\}_{k=0}^{m-1}$
$\{[\![v_k^A]\!]\}_{k=0}^{m-1}$ $\longrightarrow$

for $k = 0$ to $m-1$ **do**
  $[\![r_k^B]\!] = \mathsf{E}(r_k^B, t_k^B)$
  $[\![v_k^B]\!] = [\![v_k^A]\!]^{r_k^B}\mathsf{E}(0, s_{1,k}^B)$
  zk-proof [
  $([\![r_k^B]\!]; r_k^B, t_k^B) \in R_{\text{NO-ZERO}}$
  $([\![v_k^A]\!], [\![v_k^B]\!], [\![r_k^B]\!]; r_k^A, s_{1,k}^B, t_k^B) \in R_{\text{PRV-MLT}}$]
  $[\![w_{k+r_B}]\!] = [\![v_k^B]\!]\mathsf{E}(0, s_{2,k}^B)$
end for
zk-proof [
$(\{[\![v_k^B]\!]\}_{k=0}^{m-1}, \{[\![w_k]\!]\}_{k=0}^{m-1}; r_B, \{s_{2,k}^B\}_{k=0}^{m-1}) \in R_{\text{ROT}}$]

$\{[\![r_k^B]\!]\}_{k=0}^{m-1}$
$\{[\![w_k]\!]\}_{k=0}^{m-1}$ $\longleftarrow$

for $k = 0$ to $m-1$ **do**
  run $w_k \leftarrow \text{DECR}([\![w_k]\!])$
end for
return $[\exists k^* : w_{k^*} = 0]$

applicable cryptosystem. Protocol 6.2 can be readily extended to the multiparty case, but since rotations must be applied sequentially, the number of rounds grows with the number of parties.

Encrypted Output

We observe that using the special blinding technique used in Phase 4 of Protocol 6.1, we can adapt Protocol 6.2 to deliver encrypted output. Protocol 6.3 depicts these changes. It adds a round at the beginning with a kind of blinding that produces a swap of the inputs. In fact, s is a randomly generated value from the two-value set $\{-1, 1\}$. If $s = 1$ then the order of the inputs is not changed, while $s = -1$ means that the input's order is swapped. Since parties generate s at random and they do not know what its actual value is, the result of the comparison that is stored in the bit b does not say anything about the result of the original comparison. The actual comparison result is corrected using b and $[\![s]\!]$. This trick, due to Toft [Tof09b], works in general in any greater-than comparison protocol (see Section 6.3.3). The resulting protocol has 4 rounds. In addition, no explicit multiplication gates are required.

Protocol 6.3 can also be extended to handle comparison in a multiparty setting. However, due to the sequential nature of both the blinding in the first round, and the rotations, the round complexity will grow linearly with the number of parties.

We give now a simulation of Protocol 6.3. Similarly as with Protocol 6.1 we give a simulation whose inputs depend on an encrypted bit $[\![b]\!]$. The simulation follows the same ideas and principles.

Theorem 6.3 *Given input values* $\{x_k^{(0)}\}_{k=0}^{m-1}$, $\{y_k^{(0)}\}_{k=0}^{m-1}$, $\{x_k^{(1)}\}_{k=0}^{m-1}$ *and* $\{y_k^{(1)}\}_{k=0}^{m-1}$ *and an encryption* $[\![b]\!]$ *with* $b \in \{0, 1\}$ *Protocol 6.3 can be simulated statistically for inputs* $\{[\![\tilde{x}_k]\!]\}_{k=0}^{m-1}$ *and* $\{[\![\tilde{y}_k]\!]\}_{k=0}^{m-1}$ *where* $[\![\tilde{x}_k]\!] = [\![(1-b)x_k^{(0)} + bx_k^{(1)}]\!]$ *and* $[\![\tilde{y}_k]\!] = [\![(1-b)y_k^{(0)} + by_k^{(1)}]\!]$.

Proof sketch. Algorithm 6.2 presents a simulator for Protocol 6.3 assuming that Party A^* is corrupted. It follows the same structure of Algorithm 6.1 and the same results follow. In fact, it works running the protocol for both sets of inputs $\{x_k^{(0)}\}_{k=0}^{m-1}$, $\{y_k^{(0)}\}_{k=0}^{m-1}$. The actual selection between one problem or the another is done by the selection with b given in encrypted form. One of the main goals is to be able to run the simulation of the threshold decryption which, as we know, requires ciphertexts and the plaintext encrypted in them. Thus, the simulator accommodates all internal values in such a way that eventually the two problems are equal at the decryption step, independently of bit b. ■

6.2.5 Performance Evaluation

We now present the performance figures of the protocol in this section. Furthermore, we put next to each other the arithmetic circuit with log-depth and Protocols 6.1 and 6.3. We add the linear depth circuit of [ST04] to complete the picture. In order to compare them, we assume that the 2-party setting where (2,2)-threshold homomorphic ElGamal is set up. We consider the number of rounds, the computational cost measured in exponentiations and the communication complexity calculated as the number of group elements exchanged among the parties. The resulting complexities figures are presented in Table 6.2.

Protocol 6.3 Improved Two-Party Greater-Than Comparison

Input: $\{[\![x_k]\!]\}_{k=0}^{m-1}, \{[\![y_k]\!]\}_{k=0}^{m-1}, x_k, y_k \in \{0,1\}$.
Output: $[\![[x > y]]\!]$.

Party A

pick $\{\{s_{j,k}^A\}_{k=0}^{m-1}\}_{j=1}^{3} : s_{j,k}^A \in_R \mathcal{R}$
pick $\{t_k^A\}_{k=0}^{m-1} : t_k^A \in_R \mathcal{R}$
pick $u_1^A, u_2^A \in_R \mathcal{R}$
pick $\{r_k^A\}_{k=0}^{m-1}, r_k^A \in_R \mathcal{M}\backslash\{0\}$
pick $r_A \in_R \{0, \ldots, m-1\}$
pick $s_A \in_R \{-1, 1\}$

Party B

pick $\{\{s_{j,k}^B\}_{k=0}^{m-1}\}_{j=1}^{3} : s_{j,k}^B \in_R \mathcal{R}$
pick $\{t_k^B\}_{k=0}^{m-1} : t_k^B \in_R \mathcal{R}$
pick $u_1^B \in_R \mathcal{R}$
pick $\{r_k^B\}_{k=0}^{m-1} : r_k^B \in_R \mathcal{M}\backslash\{0\}$
pick $r_B \in_R \{0, \ldots, m-1\}$
pick $s_B \in_R \{-1, 1\}$

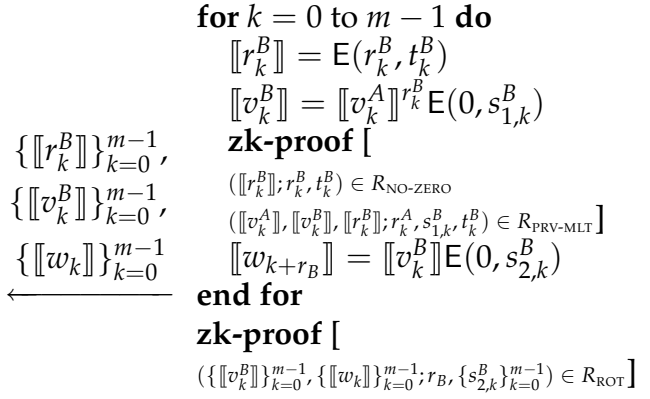
$[\![s_B]\!] = \mathsf{E}(s_B, u_1^B)$
zk-proof $[(\![s_B]\!], -1, 1; u_1^B) \in R_{\binom{2}{1}\text{KNW}}]$
for $k = 0$ to $m-1$ **do**
  $[\![g_k^B]\!] = [\![x_k - y_k]\!]^{s_B}\mathsf{E}(0, s_{3,k}^B)$
  zk-proof $[(\![x_k - y_k]\!], [\![g_k^B]\!], [\![s_B]\!]; s_B, s_{3,k}^B, u_1^B) \in R_{\text{PRV-MLT}}]$
end for

$\xleftarrow{[\![s_B]\!], \{[\![g_k^B]\!]\}_{k=0}^{m-1}}$

$[\![s_A]\!] = \mathsf{E}(s_A, u_1^A)$
$[\![s]\!] = [\![s_B]\!]^{s_A}\mathsf{E}(0, u_2^A)$
zk-proof $[(\![s_A]\!], -1, 1; u_1^A) \in R_{\binom{2}{1}\text{KNW}}$
$(\![s_B]\!], [\![s]\!], [\![s_A]\!]; s_A, u_2^A, u_1^A) \in R_{\text{PRV-MLT}}]$
for $k = 0$ to $m-1$ **do**
  $[\![g_k]\!] = [\![g_k^B]\!]^{s_A}\mathsf{E}(0, s_{3,k}^A)$
  zk-proof $[(\![g_k^B]\!], [\![g_k]\!], [\![s_A]\!]; s_A, s_{3,k}^A, u_1^A) \in R_{\text{PRV-MLT}}]$
end for
$\delta_m = 0$
for $k = m-1$ down-to 0 **do**
  $[\![\delta_k]\!] = [\![3\delta_{k+1} + g_k]\!]$
  $[\![\gamma_k]\!] = [\![\delta_k - 1]\!]$
  $[\![r_k^A]\!] = \mathsf{E}(r_k^A, t_k^A)$
  $[\![u_k]\!] = [\![\gamma_k]\!]^{r_k^A}\mathsf{E}(0, s_{1,k}^A)$
  zk-proof $[(\![r_k^A]\!]; r_k^A, t_k^A) \in R_{\text{NO-ZERO}}$
  $(\![\gamma_k]\!], [\![u_k]\!], [\![r_k^A]\!]; r_k^A, s_{1,k}^A, t_k^A) \in R_{\text{PRV-MLT}}]$
  $[\![v_{k+r_A}^A]\!] = [\![u_k]\!]\mathsf{E}(0, s_{2,k}^A)$
end for
zk-proof $[(\{[\![u_k]\!]\}_{k=0}^{m-1}, \{[\![v_k^A]\!]\}_{k=0}^{m-1}; r_A, \{s_{2,k}^A\}_{k=0}^{m-1}) \in R_{\text{ROT}}]$

$\xrightarrow{[\![s_A]\!], \{[\![g_k]\!]\}_{k=0}^{m-1}, \{[\![r_k^A]\!]\}_{k=0}^{m-1}, \{[\![u_k]\!]\}_{k=0}^{m-1}, [\![s]\!], \{[\![v_k^A]\!]\}_{k=0}^{m-1}}$

Protocol 6.3 (continued)



for $k = 0$ to $m-1$ **do**
  run $w_k \leftarrow \text{DECR}([\![w_k]\!])$
end for
$b = [\exists k^* : w_{k^*} = 0]$
return $[\![\frac{1-s}{2} b + \frac{1+s}{2}(1-b)]\!]$

Clearly, Protocol 6.3 outperforms all the others in this particular setting. However, as said before, the number of rounds starts growing when considering a setting in which more parties are involved in the execution of the protocol.

Table 6.3 shows the performance of protocol from a more general point of view. We assume an n-party setting in which integers of bit-length m are compared. A generic multiplication gate costs c computations (e.g., modular exponentiations), b units of communication (e.g., groups elements) and r rounds of interaction. We see now that one of the protocols may be the best in a performance parameter under certain circumstances. For example, if the setting considers a large number of parties and a somewhat short input length, we see that Protocol 6.3 may involve many more rounds than Protocol 6.1 and the arithmetic circuits. If multiplication gates are very efficient (that is, c and b are rather low), then the arithmetic circuits will become more efficient than the other two solutions. Protocol 6.3 will not take any advantage of the fact that multiplications are cheap. This scenario of cheap multiplication gates happens when, for instance, the multiplication gates can be replaced by private multiplier. This is the case when one of the inputs is public.

6.2.6 Yao's Garbled Circuit Approach

There are other approaches to solve secure comparison, and more generally any secure multiparty computation, that are outside the scope of this thesis. These approaches may be advantageous under different circumstances and thus can be applied in practice. The *garbled circuit* approach was introduced by Yao [Yao82] where it is shown how any function can be evaluated securely by two parties in the semi-honest model. Later, the approach has been extended to the malicious case [Lin03, Pin03, MNPS04, MF06, KS06b, LP07] and multiparty case [BMR90]. Yao's protocol requires oblivious transfer and secure pseudorandom generators as building blocks.

Roughly speaking, Yao's protocol works as follows. Assume two parties, A and B willing to securely evaluate $f(x, y)$ which is represented as a binary circuit. Party A has

Algorithm 6.2 Simulator for Protocol 6.3

Input: $\{x_k^{(0)}\}_{k=0}^{m-1}, \{y_k^{(0)}\}_{k=0}^{m-1}, [\![b]\!]$
  for $k = 0$ to $m-1$ **do**
    $[\![\tilde{x}_k]\!] = [\![(1-b)x_k^{(0)} + bx_k^{(1)}]\!]$
    $[\![\tilde{y}_k]\!] = [\![(1-b)y_k^{(0)} + by_k^{(1)}]\!]$
  end for
  pick $s_B^{(0)} \in_R \{-1, 1\}$
  $s_B^{(1)} = (-1)^{[[x^{(0)} > y^{(0)}] \neq [x^{(1)} > y^{(1)}]]} s_B^{(0)}$
  $[\![\tilde{s}_B]\!] = [\![(1-b)s_B^{(0)} + bs_B^{(1)}]\!]$
  print $[\![\tilde{s}_B]\!], \mathcal{S}_{\binom{2}{1}\text{KNW}}([\![\tilde{s}_B]\!], -1, 1)$
  for $k = 0$ to $m-1$ **do**
    $[\![\tilde{g}_k^B]\!] = [\![(1-b)(x_k^{(0)} - y_k^{(0)})s_B^{(0)} + b(x_k^{(1)} - y_k^{(1)})s_B^{(1)}]\!]$
    print $[\![\tilde{g}_k^B]\!], \mathcal{S}_{\text{PRV-MLT}}([\![\tilde{x}_k - \tilde{y}_k]\!], [\![\tilde{g}_k^B]\!], [\![\tilde{s}_B]\!])$
  end for
  Party A^* gives $[\![\tilde{s}_A]\!], \{[\![\tilde{g}_k]\!]\}_{k=0}^{m-1}, \{[\![\tilde{r}_k^A]\!]\}_{k=0}^{m-1}, \{[\![\tilde{u}_k]\!]\}_{k=0}^{m-1}, [\![\tilde{s}]\!], \{[\![\tilde{v}_k^A]\!]\}_{k=0}^{m-1}$
  run $(\mathtt{tr}_1, (\tilde{s}_A, u_1^A)) \leftarrow \mathcal{W}_{\binom{2}{1}\text{KNW}}([\![\tilde{s}_A]\!], -1, 1)$
  run $(\mathtt{tr}_2, (\tilde{s}_A, u_2^A, u_1^A)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![\tilde{s}_B]\!], [\![\tilde{s}]\!], [\![\tilde{s}_A]\!])$
  for $k = 0$ to $m-1$ **do**
    run $(\mathtt{tr}_{3,k}, (\tilde{s}_A, s_{3,k}^A, u_1^A)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![\tilde{g}_k^B]\!], [\![\tilde{g}_k]\!], [\![\tilde{s}_A]\!])$
  end for
  for $j = 0, 1$ **do**
    $\delta_m = 0$
    for $k = m-1$ down-to 0 **do**
      $\delta_k = 3\delta_{k+1} + \tilde{s}_A s_B^{(j)}(x_k^{(j)} - y_k^{(j)})$
      $\gamma_k^{(j)} = \delta_k - 1$
    end for
    if $s_B^{(0)} = 1$ **xor** $x^{(0)} > y^{(0)}$ **then**
      pick $k_j \in_R \{0, \ldots, m-1\}$
    else
      let k_j s.t. $\gamma_{k_j}^{(j)} = 0$
    end if
  end for
  $\theta = k_1 - k_0$
  for $k = m-1$ down-to 0 **do**
    $[\![\tilde{\gamma}_k]\!] = [\![(1-b)\gamma_k^{(0)} + b\gamma_k^{(1)}]\!]$
    run $(\mathtt{tr}_{4,k}, (\tilde{r}_k^A, t_k^A)) \leftarrow \mathcal{W}_{\text{NO-ZERO}}([\![\tilde{r}_k^A]\!])$
    run $(\mathtt{tr}_{5,k}, (\tilde{r}_k^A, s_{1,k}^A, t_k^A)) \leftarrow \mathcal{W}_{\text{PRV-MLT}}([\![\tilde{\gamma}_k]\!], [\![\tilde{u}_k]\!], [\![\tilde{r}_k^A]\!])$
  end for
  run $(\mathtt{tr}_6, (\tilde{r}_A, \{s_{2,k}^A\}_{k=0}^{m-1})) \leftarrow \mathcal{W}_{\text{ROT}}(\{[\![\tilde{u}_k]\!]\}_{k=0}^{m-1}, \{[\![\tilde{v}_k^A]\!]\}_{k=0}^{m-1})$
  print $\mathtt{tr}_1, \mathtt{tr}_2, \{\{\mathtt{tr}_{j,k}\}_{k=0}^{m-1}\}_{j=3}^{5}, \mathtt{tr}_6$

Algorithm 6.2 (continued)

for $k = 0$ to $m-1$ **do**
  pick $r_k^{B(0)} \in_R \mathcal{M}\backslash\{0\}$
  find $r_k^{B(1)}$ s.t. $r_k^{B(0)}\gamma_k^{(0)}\tilde{r}_k^A = r_{k+\theta}^{B(1)}\gamma_{k+\theta}^{(1)}\tilde{r}_{k+\theta}^A$
  for $j = 0, 1$ **do**
    for $k = 0$ to $m-1$ **do**
      $v_k^{B(j)} = r_k^{B(j)}\gamma_k^{(j)}\tilde{r}_k^A$
    end for
  end for
  $[\![\tilde{r}_k^B]\!] = [\![(1-b)r_k^{B(0)} + br_k^{B(1)}]\!]$
  $[\![\tilde{v}_k^B]\!] = [\![(1-b)v_k^{B(0)} + bv_k^{B(1)}]\!]$
  print $[\![\tilde{r}_k^B]\!], [\![\tilde{v}_k^B]\!], \mathcal{S}_{\text{NO-ZERO}}([\![\tilde{r}_k^B]\!]), \mathcal{S}_{\text{PRV-MLT}}([\![\tilde{v}_k^A]\!], [\![\tilde{v}_k^B]\!], [\![\tilde{r}_k^B]\!])$
  pick $r_B^{(0)} \in_R \{0, \ldots, m-1\}$
  $r_B^{(1)} = r_B^{(0)} - \theta$
  $\tilde{w}_{k+r_B^{(0)}} = v_k^{B(0)}$
end for
print $\{[\![\tilde{w}_k]\!]\}_{k=0}^{m-1}, \mathcal{S}_{\text{ROT}}(\{[\![\tilde{v}_k^B]\!]\}_{k=0}^{m-1}, \{[\![\tilde{w}_k]\!]\}_{k=0}^{m-1})$
for $k = 0$ to $m-1$ **do**
  print $\mathcal{S}_{\text{DECR}}([\![\tilde{w}_k]\!], \tilde{w}_k)$
end for

input x and party B has input y. The circuit for f is "encrypted" by one of the parties, say B, in which the input y is hard-coded. The encrypted circuit is sent to party A who will "decrypt" it after getting the keys corresponding to the input x. These keys are obtained via a secure 1-out-of-2 oblivious transfer. This way, party A is only able to get the value $f(x, y)$ and nothing else since the evaluation of the internal encrypted gates does not reveal any information. This enables constant round secure function evaluation independent of the circuit size. This contrast with the approach using threshold homomorphic cryptosystems where the parties must interact in a gate-by-gate fashion.

The performance of Yao's protocol is determined by the number of oblivious transfer and the encryption and decryption of the circuit. If the circuit to be evaluated has small fan-in and the number of gates is relatively small, Yao's protocol becomes efficient. However, if the circuit fan-in is large (which means many oblivious transfers) or the circuit is significantly large (which means large computations encrypting and decrypting the circuit), Yao's may not be as attractive as using an arithmetic circuit, or any other approach.

In order to withstand malicious parties, additional commitments, proof of knowledge and a stronger oblivious transfer protocol are needed. Oblivious transfer protocols should work on committed values to prevent malicious parties from cheating. An oblivious transfer working on *committed* values is thus needed. Examples are protocols for *committed oblivious transfer* [Cré90, CvdGT95, CD97, GMY04, KSV07] or variations of it such as verifiable oblivious transfer [CC00, JS07] and committing OT [KS06a, KS08]. The protocols for these variants of oblivious transfer using commitments of [KSV07] need a similar set up as the protocols presented in this thesis. That is, parties set up a threshold

	Computation	Communication	Rounds
Linear depth circuit [ST04]	$168m$	$84m$	$2m$
Log-depth circuit	$252m - 84\log m$	$126m - 42\log m$	$2\log m$
Protocol 6.1	$192m$	$84m$	6
Protocol 6.3	$124m$	$48m$	4

Table 6.2: Performance of protocols for greater-than comparison assuming that two parties use threshold homomorphic ElGamal. Computations are measured as the number of exponentiations and communication is estimated with the number of group elements exchanged.

homomorphic cryptosystem and run Yao's protocol based on it.

Integer comparison of m-bit integers is thus solved by applying Yao's protocol on a boolean circuit for comparison. The depth of those circuits does not affect the computation of the protocol. Thus, the (boolean version of the) linear depth circuit of [ST04] is enough. The computational complexity is dominated by that of the computation of m committed oblivious transfer, since the circuit is relatively small in size. In fact, especially in the semi-honest case, Yao's protocol is efficient. For example, Yao's protocol is used to perform secure integer comparison in a protocol for face recognition by Sadeghi *et al.* [SSW09]. Note that Yao's protocol requires that each party knows the integer to be compared, and hence it is less general than the approach considered in the thesis.

6.3 Problems Related to Integer Comparison

In this section, we overview various related problems to integer comparison. We also discuss different variations of integer comparison.

6.3.1 Signum

The signum function is the three-valued function $\operatorname{sgn} : \mathbb{Z} \to \{-1, 0, 1\}$ defined by $\operatorname{sgn}(z) = -1$ if $z < 0$, $\operatorname{sgn}(z) = 0$ if $z = 0$, and $\operatorname{sgn}(z) = 1$ if $z > 0$. Given two non-negative integers x and y, the following relation holds.

$$\operatorname{sgn}(x - y) = 2[x > y] - 1 + [x = y].$$

This suggests that having greater-than and equality comparisons of x and y, one obtains the signum of $x - y$ for free.

Using the log-depth arithmetic circuit one can compute greater-than and equality at once and hence the computation of the signum of $x - y$ comes at virtually no extra cost. A more direct approach can be obtained by observing the following recursive formula. Splitting the inputs x and y in two chunks such that $x = X_1X_0$ and $y = Y_1Y_0$ of respectively the same size, we get that

$$\operatorname{sgn}(x - y) = \operatorname{sgn}(X_1 - Y_1) + [X_1 = Y_1] \operatorname{sgn}(X_0 - Y_0).$$

The computation of the signum is then possible through a recurrence formula for both signum and equality. Splitting the strings x and y in about equally long chunks one obtains a logarithmic depth circuit.

	Computation	Communication	Rounds
Linear depth circuit [ST04]	$2mc$	$2mb$	rm
Log-depth circuit	$3mc$	$3mb$	$r \log m$
Protocol 6.1	$54mn + mc$	$21mn + mb$	$2 + r + n$
Protocol 6.3	$62mn$	$24mn$	$2n$

Table 6.3: Performance of protocols for greater-than comparison assuming n parties using a threshold cryptosystem. Every multiplication induces a cost of c computations, b units of communication, and r rounds of interaction.

Analogously to the circuit for greater-than comparison, similar trade-offs between size and depth appear here. If the recurrence structure is set up in such a way that the length the most significant part is always a bit, then the equality term requires a total of $m - 1$ multiplication gates only. Additionally, $m - 1$ multiplications are needed for computing the recurrence, yielding a circuit of size $2m - 2$ with depth m. On the other hand, if the least significant bits in the recurrence have always length 1, we need $2m - 2$ multiplications for the equality computation and $m - 1$ multiplications for the recursive merging, yielding $3m - 3$ multiplication in depth m. Note that in any case, the multiplication $x_0 y_0$ is not needed at all. These particular cases are the solutions presented in [ST04]. In the case of equally long chunks at each recursive step, the depth of the circuit is $\log m$ while the size of the circuit is $3m - \log m - 2$.

6.3.2 Addition Circuits

Given two m-bit non-negative integers x and y, the bits z_k of the sum $z = x + y$ are determined by the carries c_k and the bits x_k and y_k. More concretely, bit z_k is computed as $z_k = x_k + y_k + c_{k-1} - 2c_k$, where c_k is the k-th carry bit. A carry c_k is set to 1 if either it is generated by the bits x_k and y_k, or the previous carry c_{k-1} is propagated. This is written as follows:

$$c_k = g_k + c_{k-1} p_k,$$

where g_k indicates if a carry is generated while p_k tells if a carry is propagated. A carry generation is given by $g_k = x_k y_k$, while a propagation is $p_k = x_k + y_k - 2x_k y_k$. Note that once $x_k y_k$ is computed, g_k, p_k, and z_k are obtained using linear operations.

Similarly as with the circuit for comparison, the following is noted. If $x = X_1 X_0$ and $y = Y_1 Y_0$ where $|X_0| = |Y_0|$ and $|X_1| = |Y_1|$ the sum $x + y$ produces a carry c if either $X_1 + Y_1$ gives a carry C_1, or $X_0 + Y_0$ gives a carry C_0 and it propagates through $X_1 + Y_1$. Note that $x + y$ propagates a carry only if both $X_0 + Y_0$ and $X_1 + Y_1$ propagate a carry. This induces the following recurrence formula

$$\begin{aligned} c &= C_1 + P_1 C_0 \\ p &= P_1 P_0, \end{aligned} \tag{6.4}$$

which yields an arithmetic circuit with similar features as that of greater-than comparison.

Note that a carry is propagated through $x + y$ exactly when $x = \overline{y}$, where $\overline{y}$ denotes the bit-complement of y, i.e., $\overline{y} = 2^m - 1 - y$. Otherwise, a propagated carry is "absorbed" by $x + y$.

Protocol 6.4 Greater-than comparison

Input: $[\![x]\!]$, $[\![y]\!]$
Output: $[\![[x = y]]\!]$
Generate $[\![b]\!]$ such that $b \in_R \{0, 1\}$
Compute:
$[\![\tilde{x}]\!] = [\![(1 - b)x + by]\!]$
$[\![\tilde{y}]\!] = [\![(1 - b)y + bx]\!]$
run $c \leftarrow \text{GT}([\![\tilde{x}]\!], [\![\tilde{y}]\!])$
return $[\![(1 - b)c + b(1 - c)]\!]$

Interestingly, we can show an equivalence between computing the carry of $x + y$ and greater-than comparison. In fact, we can check that $x + y$ gives a carry exactly when $x > \overline{y}$.

Proposition 6.4 *Let x and y be m-bit integers. $x > \overline{y}$ if and only if $x + y \geq 2^m$.*

Proof. $x > \overline{y} \Leftrightarrow x + 2^m - 1 - y \geq 2^m \Leftrightarrow x - 1 \geq y \Leftrightarrow x > y$ ■

The proposition states that the computation of a greater-than comparison of two m-bit integers immediately gives the carry of the sum of two m-bit integers. In fact, the recurrence given in Eq. (6.4) represents the same formula of Eq. (6.3) by replacing $C_i = [X_i > \overline{Y_i}]$ and $P_i = [X_i = \overline{Y_i}]$.

The consequence is that it suffices to solve one of the problems to have a solution to the other problem. This equivalence may be useful to use intrinsic properties of one of the problems to solve the other, say, more efficiently.

6.3.3 Comparisons with Public Output

Consider a protocol for any of the integer comparison on encrypted inputs that gives the result in encrypted form. The comparison with public output is easily achieved by adding a threshold decryption step. The other way around, computing the result of the comparison in encrypted form given a protocol for computing comparison in the clear is much trickier. In this section, we consider some issues appearing when comparison with public output is used to provide a solution with encrypted output.

Greater-Than Comparison

In the case of the greater-than comparison going from public output to encrypted output requires little overhead by the following observation made by Toft [Tof09b].

The construction described in Protocol 6.4 allows the computation of greater-than with encrypted output using any protocol for greater-than comparison with public outputs. The intuition is that inputs can be swapped according to an unknown and random bit b, given in encrypted form. This swap is performed by using a multiplication gate $[\![(x - y)b]\!] \leftarrow \text{MULT}([\![x - y]\!], [\![b]\!])$. Then, using linear operations on the encryptions we get $[\![\tilde{x}]\!] = [\![x - (x - y)b]\!]$ and $[\![\tilde{y}]\!] = [\![y + (x - y)b]\!]$.

Clearly, if $b = 0$, then $\tilde{x} = x$ and $\tilde{y} = y$ or, vice versa, if $b = 1$ we have $\tilde{x} = y$ and $\tilde{y} = x$. Since the bit b is unknown and random, the result of the comparison on $\tilde{x}$ and $\tilde{y}$ gives no clue about the comparison between x and y. The inputs are flipped with 50% probability.

Protocol 6.5 Equality test with encrypted inputs and public output

Input: $[\![x]\!]$, $[\![y]\!]$
Output: $[x = y]$
Generate $[\![r]\!]$ such that $r \in_R \mathcal{M}$
run $[\![z]\!] \leftarrow \text{MULT}([\![x - y]\!], [\![r]\!])$
Decrypt $[\![z]\!]$
return $[z = 0]$

At the end, the actual comparison is computed by flipping the disclosed comparison according to the encrypted bit b. In fact, this step requires only linear properties, thus it needs no interaction for homomorphic encryptions.

Therefore, we get greater-than comparison with encrypted output from greater-than comparison with public output at the cost of the generation of an encrypted unknown and random bit plus one multiplication gate. This says that greater-than comparison with encrypted output has almost the same complexity as a greater-than comparison on public outputs.

Note that for a greater-than comparison with bit-wise encrypted inputs, we can achieve a similar equivalence. The difference is that all encrypted inputs must be swapped at the same time. Thus, a multiple swap must be performed on the inputs, costing n extra multiplication gates at the beginning.

For the sake of completeness, we mention that a protocol for generating an encryption of a random bit can be obtained following the approach given in [DFK+06, Section 4]. However, this requires a cryptosystem or a linear secret sharing with full decryption and message space of prime order. This may also be achieved using some multiplication gates as well as explained in [ST06, Section 2.3]. Observe, however, that the generation of a random and unknown encrypted bit can be moved completely to an off-line phase, since it does not depend on the inputs at all.

Equality Test

In the case of an equality test, solutions providing encrypted output require a lot more work compared to those that deliver public output. In order to illustrate this fact, we consider the equality test given by Protocol 6.5.

Given $[\![x]\!]$ and $[\![y]\!]$, Protocol 6.5 depicts a simple, yet effective way to decide whether $x = y$ publicly. Parties, simply compute an encryption $[\![r]\!]$ of a uniformly random r unknown to everyone. Then, using a multiplication gate compute $[\![(x - y)r]\!]$ which is later decrypted. If the decrypted value equals 0, then it means that $x = y$. Otherwise, $x \neq y$, and the decrypted value is random disclosing no information of x or y.

In the case of bit-wise inputs $\{[\![x_k]\!]\}_{k=0}^{m-1}$ and $\{[\![y_k]\!]\}_{k=0}^{m-1}$ getting public result of an equality comparison is achieved by simply computing $[\![x]\!]$ and $[\![y]\!]$ using homomorphic properties and apply Protocol 6.5

Obtaining the encryption $[\![r]\!]$ where r in an unknown random element is simple. We let the i-th party give an encryption $[\![r_i]\!]$ of a private value r_i selected at random. The encryption $[\![r]\!] = [\![\sum_i r_i]\!]$ can be computed with no further interaction. Note that r may equal 0 in which case, Protocol 6.5 may give a false response (i.e., when $x \neq y$). The protocol is invoked as many independent times in parallel as to make the probability that all picked values r are 0 as small as needed.

Protocol 6.6 Bit-decomposition

Input: $[\![x]\!]$, $0 \leq x < 2^m$.
Output: $\{[\![x_k]\!]\}_{k=0}^{m-1}$ such that $x = \sum_{i=0}^{m-1} 2^i x_i$.
  for $k = m-1$ down-to 0 **do**
    $[\![x_k]\!] = [\![[x > 2^k - 1]]\!]$ $\rightsquigarrow$ Note that $[x > 2^k - 1] = [x \geq 2^k]$
    $[\![x]\!] \leftarrow [\![x - 2^k x_k]\!]$
  end for
  return $\{[\![x_k]\!]\}_{k=0}^{m-1}$

If the bit of information telling if $x = y$ is required to be in an encryption, we need to apply a more elaborated protocol, as for instance shown in [DFK+06] and later improved in [NO07]. This protocol is the result of the evaluation of an arithmetic circuit that requires the execution of many multiplication gates.

The same holds for bit-wise inputs. In fact, the circuits in Section 6.1 require at least $2m$ multiplications. Damgård *et al.* [DFK+06] present an arithmetic constant depth circuit via the computation of an unbounded fan-in AND. The equality comparison is obtained by computing bits $f_k = x_k y_k$ for all $0 \leq k < n$. Then $[x = y] = \prod_{k=0}^{m-1} f_k$.

6.3.4 Bit-decomposition Problems

Bit-decomposition protocols take as input an encrypted value $[\![x]\!]$ with $x < 2^m$ and produce encryptions $\{[\![x_k]\!]\}_{k=0}^{m-1}$ where $\{x_k\}_{k=0}^{m-1}$ is the bit representation of a non-negative integer x. See [ACS02, DFK+06, ST06, NO07, Tof09a] for examples of protocols solving this kind of problems. Bit-decomposition is an important primitive in secure arithmetic since it bridges the gap between problems that are inherently bit-oriented but the inputs are available in integer form.

A typical example is greater-than comparison. Suppose $[\![x]\!]$ and $[\![y]\!]$ be given as inputs with x and y being non-negative integers. A natural approach for circuits and protocols is to compute $[x > y]$ (or eventually $[\![[x > y]]\!]$) relying on a bit-decomposition protocol. Namely, inputs $[\![x]\!]$ and $[\![y]\!]$ are converted into $\{[\![x_k]\!]\}_{k=0}^{m-1}$ and $\{[\![y_k]\!]\}_{k=0}^{m-1}$. Then, a bit-wise greater-than comparison is performed, like the ones presented in this thesis. This is the overall approach that is used in [DFK+06] where they get constant depth circuit for greater-than comparison with encrypted integer input. This is acquired by using special tricks to get both bit-decomposition and bit-wise greater-than in constant depth. However, the size of the resulting circuit is very large and hence unsatisfactory for practical applications.

The bits of an m-bit integer x can be computed using greater-than comparisons as follows. First, note that x_k, the k-th bit of x, can be extracted by $x_k = [x \geq 2^k]$. Protocol 6.6 iteratively extract all bits.

A related primitive to bit-decomposition is the computation of the least significant bit of an encrypted integer. That is, given an encryption $[\![x]\!]$ compute $[\![x_0]\!]$, where x_0 is the least significant bit of x, denoted with $\mathrm{LSB}(x)$. Protocols for this task have been investigated by Schoenmakers and Tuyls [ST06] and Nishide and Otha [NO07].

The following proposition relates greater-than comparison and the least significant bit.

Proposition 6.5 *Let q be an odd integer. $x > (q-1)/2$ if and only if $\mathrm{LSB}(2x \bmod q) = 1$.*

In fact, if $x > (q-1)/2$ then $2x \geq q$, and thus $2x \bmod q = 2x - q$ wraps around. Then, $2x \bmod q$ is odd.

This equivalence is used by Nishide and Otha [NO07] to construct a general greater-than comparison circuit based only on comparisons with the public value $(q-1)/2$, solving it via a protocol for the least significant bit. Interestingly, computing the least significant bit reduces to computing bit-wise greater-than comparisons. Nishide and Otha [NO07] thus avoid the use of a bit-decomposition protocol to compute greater-than comparison on encrypted integer inputs.

6.4 A Cost Model for Arithmetic Circuits

Circuit complexity is a branch of theoretical computer science in which a circuit is defined as a directed acyclic graph where nodes are gates, there are sources (input gates) and sinks (output gates). The complexity parameters are usually the size (number of gates) and the depth (longest path in the graph). The analysis of the size and the depth of circuits is necessary in some practical scenarios, such as designing circuits for microprocessors, where minimizing the size and/or depth of circuits is important. More fundamentally, circuit complexity provides a model in which one can prove lower bounds on the depth or size of those circuits. In fact, a number of lower bounds for circuits are known nowadays, and there is a still-growing range of techniques to prove these lower bounds. We refer the reader to [Vol99, Chapter 3] and references therein for an overview of these techniques.

We would like to answer similar questions about the arithmetic circuits we consider in this chapter. In particular, since we are interested in computational efficiency, one could be interested to know a lower bound on the size of arithmetic circuits solving integer comparison. Remember that the size in our context just takes into account multiplication gates only. Therefore one should define an appropriate cost model to bound the complexity. In most of cases, however, the lower bound is computed in a model where every gate is equally expensive, so one must count all gates.

In the following, we sketch a model that considers arithmetic circuits with an appropriate cost model. Let $\mathcal{M}$ be a ring, and let $\{x_1, \ldots, x_n\}$ be a set of input variables. An arithmetic circuit with inputs $x_1, \ldots, x_n$ is a directed acyclic graph where input gates are the nodes with in-degree zero and output gates are the nodes with out-degree zero. Input gates are labelled with the input variables, and we assume a number of gates labelled with constants in $\mathcal{M}$. Non-input gates have in-degree two and are labelled with either '$+$', '$\cdot$' or '$\times$' (in the first case the node is an addition gate, in the second case a multiplication by a constant, and in the third case a multiplication gate). Addition gates may be reached by any gate, while multiplication by a constant is restricted to be reached by exactly one constant input gate, and multiplication gates are not allowed to be reached by constant input gates. Non-output gates may have out-degree larger than one.

Every node computes a polynomial in the ring $\mathcal{M}[x_1, \ldots, x_n]$ in the following way. An input node just computes the input variable, or the constant it is labelled with. An addition gate computes the sum of the polynomials of all gates that reach it, while a multiplication by a contant and a multiplication gate compute the product of the two polynomial of the gates that reach it. The polynomial $g \in \mathcal{M}[x_1, \ldots, x_n]$ is said to be computed by the arithmetic circuit if one of the output gates of the circuit computes g.

We define the size of the circuit as the number of multiplication gates while the depth

is the largest number of multiplication gates in any directed path going from any input gate to any output gate. This definition of size and depth is motivated by the fact that the secure evaluation of an arithmetic circuit only multiplications require interaction. Additions and multiplication by a constant, however, are considered costless since they are computed directly using homomorphic properties, and thus they do not affect the computation of the protocol. This model provides a framework where we can answer questions like what is the minimal size of an arithmetic circuit for certain functionality.

Several lower bounds of circuits with different bases, cost models and restrictions can be found across the literature. It is possible to prove the minimal size of a circuit for a given (fixed) depth, or assuming a specific set of gates that circuits are allowed to use. As in our case, some gates may have assigned some cost which makes the problem of calculating lower bounds more challenging. In some cases, the most natural solution is proved to be the best. An example is the boolean circuit for the addition of two m-bit integers. Red'kin [Red81] has shown that for this problem in particular, the folklore circuit is the smallest one. More concretely, he has shown that $5m - 3$ boolean gates is a lower bound for a circuit computing that problem.

We turn the attention to integer comparison circuits. We have analyzed the complexity of various circuits in Table 6.1. We have seen that the most efficient circuit is that in [ST04] which requires $2m - 1$ multiplication gates for m-bit inputs. One may wonder if it is possible to get any better, and thus obtaining a more efficient solution. We believe that cost model for directed acyclic graphs representing arithmetic circuits as the one described above will help to find concrete answers to this kind of questions. In fact, we state the following conjecture.

Conjecture 1 *Any arithmetic circuit (using the model described above) for integer comparison on m-bit inputs has size at least $2m - 1$.*

In other words, we believe that the solution of [ST04] uses the minimal number of multiplication gates. We do note, however, that we do not claim that this is the most efficient solution. As seen in Table 6.2 one can get more efficient solutions using different approaches. This conjecture suggests that no other protocol that limits itself to the evaluation of an arithmetic circuit for integer comparison (using additions and multiplications only) can be better in a computational sense.

Chapter 7

Conclusions

In this thesis, we have presented solutions to two important primitives of secure multiparty computation: special verifiable shuffles and integer comparison. We have considered three types of special shuffles, namely rotations, affine transformations, and Möbius transformations. The protocols presented in this thesis for these verifiable special shuffles are the most efficient to date. We have also studied efficient protocols for integer comparison in the setting where multiple parties do secure computations using a threshold homomorphic cryptosystem. Our solutions are competitive compared to the existing solutions in the literature, especially regarding computational performance.

We have seen that verifiable rotation can be used in a variety of problems, like fragile-mixes [RW04], specific electronic voting protocols [RS06], and integer comparison protocols [BK04, ABFL06, GSV07, DGK07, RT09]. Moreover, rotations are fundamental in some settings, for example in protocols for secure function evaluation based on mix-and-match [JJ00] or tallying protocols for preferential elections [WB09]. We believe that there are many other cryptographic scenarios where we can apply rotations and multiple fragile permutations.

We have presented zero-knowledge proofs of knowledge for rotations for which we can prove witness-extended emulation. This property implies computationally convincing proof of knowledge [DF02], a widely accepted notion that shows that a possibly cheating prover who knows some trapdoor information of the public key of the surrounding cryptosystem does not gain any advantage to cheat in the protocol. Even though this property is less general than general knowledge soundness, witness-extended emulation has the advantage of providing a simple and compact analysis. As an extra advantage, witness extended-emulation simplifies the construction of a simulator for zero-knowledge proofs used in higher-level protocols.

For integer comparison we have studied two types of solutions. On the one hand, we have presented a protocol that is the result of the secure evaluation of an arithmetic circuit. On the other hand, we have used a more direct approach that combines basic cryptographic building blocks to construct the solutions. These two types of solutions have pros and cons which we briefly summarize in the following.

The circuit approach abstracts away all details of the implementation of the gates. It suffices to construct an arithmetic circuit for the desired functionality. The arithmetic circuit is compiled into a secure protocol following the guidelines of the underlying framework (e.g., [CDN01, ST04]). The performance of the resulting protocol is determined by the size and the depth of the arithmetic circuit. A circuit for the desired functionality may be chosen trading off performance needs: a circuit with few multiplication gates yields

an efficient protocol, whereas with a shallow circuit one gets low round complexity.

The direct approach, in contrast, lacks the simplicity of the circuit approach. Instead, one has to craft the protocols almost from scratch and prove their security formally. However, since one has more freedom to look at the intrinsic properties of the problem, we may be able to optimize the performance of the resulting protocols. In fact, this is the lesson learned for integer comparison protocols: the use of a direct approach has given rise to protocols with improved performance.

We have worked with a security model that assumes the execution of our protocols in isolation (i.e., in a stand-alone model). More general security frameworks are considered today, in which several instances of a protocol may be running simultaneously (e.g., the universal composability (UC) model of Canetti [Can01, Can05]). We believe that our protocols can be adapted to stronger models using standard techniques.

Due to the recent work by Terelius and Wikström [TW10] one can prove in zero-knowledge that a shuffle uses a permutation from a restricted set, more specifically, the automorphism group of a hypergraph. The hypergraph can be set up to have as automorphism group the set of all rotations, fragile permutations, or other sets of permutations such as all permutations that keep a rooted tree intact. This opens the door to the study of shuffles using other families of permutations together with potential applications where the properties of these special families of permutations can be exploited.

One may wonder if it is possible to further improve the performance of the existing circuit-based solutions for integer comparison. As stated above, the performance of a protocol based on an arithmetic circuit is improved by optimizing the size and/or the depth. A natural question is: what is the best size and/or depth that one can achieve for any integer comparison circuit? This kind of question may be addressed using some techniques from circuit complexity.

Arithmetic circuits are directed acyclic graphs where nodes are gates and edges are wires. There are input gates (sources) and output gates (sinks) and the other gates can be either addition gates or multiplication gates (see Section 6.4). This provides a model to ask questions like what is the minimal number of multiplication gates required for any integer comparison protocol. The smallest known arithmetic circuit for integer comparison is the msb-to-lsb solution of [ST04], requiring $2m - 1$ multiplication gates. We conjecture that this is the best one can get, and leave as an open problem proving that this is indeed the case.

Index

Bibliography

[Abe99] M. Abe. Mix-networks on permutation networks. In *ASIACRYPT 1999*, volume 1716 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 1999.

[ABFL06] M. J. Atallah, M. Blanton, K. B. Frikken, and J. Li. Efficient correlated action selection. In *Financial Cryptography – FC 2006*, volume 4107 of *Lecture Notes in Computer Science*, pages 296–310. Springer-Verlag, 2006.

[ACS02] J. Alesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer-Verlag, 2002.

[BA01] M. Bhattacharya and J. Astola. Recursive structure for linear filtering using number theoretic transform. *8th IEEE International Conference on Electronics, Circuits and Systems, 2001. ICECS 2001*, 1:525–528, 2001.

[BG92] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *ACM Symposium on Theory of Computing – STOC 1988*, pages 1–10. ACM Press, 1988.

[BK04] I. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 515–529. Springer-Verlag, 2004.

[Blu70] L. Bluestein. A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Trans. Audio. Electroacust.*, AU-18:451–455, 1970.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *ACM Symposium on Theory of Computing – STOC 1990*, pages 503–513. ACM Press, 1990.

[BQ00] A. Bonisoli and P. Quattrocchi. Each invertible sharply d-transitive finite permutation set with $d \geq 4$ is a group. *Journal of Algebraic Combinatorics*, 12(3):241–250, 2000.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical. In *ACM conference on Computer and Communications Security – CCS 1993*, pages 62–73. ACM, 1993.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[Can05] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. `http://eprint.iacr.org/2000/067`.

[CC00] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer-Verlag, 2000.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *ACM Symposium on Theory of Computing – STOC 1988*, pages 11–19. ACM Press, 1988.

[CD97] R. Cramer and I. Damgård. Linear zero-knowledge – a note on efficient zero-knowledge proofs and arguments. In *ACM Symposium on Theory of Computing – STOC 1997*, pages 436–445. ACM Press, 1997.

[CD98] R. Cramer and I. Damgård. Zero-knowledge for finite field arithmetic. Or: Can zero-knowledge be for free? In *CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441. Springer-Verlag, 1998.

[CDN01] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300. Springer-Verlag, 2001.

[CDPW07] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Theory of Cryptography Conference – TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer-Verlag, 2007.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cra97] R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, Universiteit van Amsterdam, Netherlands, 1997.

[Cré90] C. Crépeau. Verifiable disclosure of secrets and applications. In *EUROCRYPT 1990*, volume 434 of *Lecture Notes in Computer Science*, pages 181–191. Springer-Verlag, 1990.

[CT65] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[CvdGT95] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computation. In *CRYPTO 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer-Verlag, 1995.

[Dam08] I. Damgård. On Σ-protocols. Lecture Notes for the course Cryptographic Protocol Theory, Å rhus University, 2008. `http://www.daimi.au.dk/~ivan/CPT.html`.

[DF02] I. Damgård and E. Fujisaki. Efficient concurrent zero-knowledge in the auxiliary string model. In *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer-Verlag, 2002.

[DFK+06] I. Damgård, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference – TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer-Verlag, 2006.

[DGK07] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference Information Security and Privacy – ACISP 2007*, volume 4586 of *Lecture Notes in Computer Science*, pages 416–430. Springer-Verlag, 2007.

[dHSŠV09] S. de Hoogh, B. Schoenmakers, B. Škorić, and J. Villegas. Verifiable rotators of homomorphic encryptions. In *Public Key Cryptography–PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 393–410. Springer-Verlag, 2009.

[dHSV10] S. de Hoogh, B. Schoenmakers, and J. Villegas. Verifiable multiply fragile shuffles, 2010. Unpublished Manuscript.

[DJ01] I. Damgård and M. Jurik. A generalization, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography–PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2001.

[DM96] J. Dixon and B. Mortimer. *Permutation Groups*, volume 163 of *Graduate Texts in Mathematics*. Springer, 1996.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *IEEE Transactions on Information Theory IT-31*, pages 469–472, 1985.

[FMM+03] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *Financial Cryptography – FC 2002*, volume 2357 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 2003.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solution to identification and signature problems. In *CRYPTO 1986*, volume 223 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[FS01] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, 2001.

[Fur05] J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Trans Fundamentals*, E88-A:172–188, 2005.

[GI08] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 379–396. Springer-Verlag, 2008.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 1999.

[GL07] J. Groth and S. Lu. Verifiable shuffle of large size ciphertexts. In *Public Key Cryptography–PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 377–392. Springer-Verlag, 2007.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Special issue of Journal of Computer and Systems Sciences*, 28(2):270–299, 1984.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing – STOC 1987*, pages 218–229. ACM Press, 1987.

[GMY04] J. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *Theory of Cryptography Conference – TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer-Verlag, 2004.

[Gro03] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography–PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer-Verlag, 2003. Full version available at `http://eprint.iacr.org/2005/246`.

[Gro04] J. Groth. *Honest Verifier Zero-Knowledge Arguments Applied*. PhD thesis, University of Århus, 2004.

[GSV07] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography–PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 330–342. Springer-Verlag, 2007.

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 2000.

[Jor72] C. Jordan. Recherches sur les substitutions. *Journal Math. Pures Appl.*, 17(2):351–363, 1872.

[JS07] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer-Verlag, 2007.

[Ker74] W. Kerby. *On infinite sharply multiply transitive groups*. Vandenhoeck & Ruprecht, Göttingen, 1974.

[Knu97] D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, MA, USA, 3rd. edition, 1997.

[KS06a] M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS06b] M. Kiraz and B. Schoenmakers. Securing Yao's garbled circuit construction against active adversaries. In *Online Proceedings of 1st Benelux Workshop on Information and System Security–WISSec 2006*, 2006.

[KS08] M. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 88–105. Springer-Verlag, 2008.

[KSV07] M. Kiraz, B. Schoenmakers, and J. Villegas. Efficient committed oblivious transfer of bit strings. In *Information Security Conference – ISC 2007*, volume 4779 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2007.

[Küs06] R. Küsters. Simulation-based security with inexhaustible interactive Turing machines. In *19th IEEE Computer Security Foundations Workshop–CSFW 2006*, pages 309–320. IEEE Computer Society, 2006.

[LA06] J. Li and M. Atallah. Secure and private collaborative linear programming. *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006*, pages 1–8, 2006.

[Lin03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[LP00] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2000.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78, 2007.

[MF06] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography–PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473, 2006.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security*, pages 287–302, 2004.

[Nef01] C. Neff. A verifiable secret shuffle and its application to e-voting. In *8th ACM conference on Computer and Communications Security – CCS 2001*, pages 116–125. ACM, 2001.

[NO07] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Public Key Cryptography–PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360. Springer-Verlag, 2007.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *1st ACM conference on Electronic Commerce–EC 1999*, pages 129–139. ACM Press, 1999.

[Oka93] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1993.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[Pas92] A. Pasini. Diagram geometries for sharply n-transitive sets of permutations or of mappings. *Design, Codes and Cryptography*, (1):275–297, 1992.

[Ped91] T. Pedersen. A threshold cryptosystem without trusted party. In *EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.

[Pin03] B. Pinkas. Fair secure twoparty computation. In *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 2003.

[PS04] M. Prabhakaran and A. Sahai. New notions of security: Achieving universal composability without trusted setup. In *ACM Symposium on Theory of Computing – STOC 2004*, pages 242–251. ACM Press, 2004.

[Qui04] J. Quistorff. A new nonexistence result for sharply multiply transitive permutation sets. *Discrete Mathematics*, 288:185–186, 2004.

[Red81] N. P. Red'kin. On the minimal realization of a binary adder. *Problemy Kibernet*, 38:181–216, 1981.

[Rob95] D. Robinson. *A Course in the Theory of Groups*. Springer-Verlag, New York, 1995.

[RS06] P. Ryan and F. Schneider. Prêt-à-Voter with re-encryption mixes. In *Computer Security – ESORICS 2006*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326, Berlin, 2006. Springer-Verlag.

[RT09] T. Reistad and T. Toft. Secret sharing comparison by transformation and rotation. In *Proceedings of the International Conference on Information Theoretic Security–ICITS 2007*, volume 4883, pages 169–180. Springer-Verlag, 2009.

[RW04] M. Reiter and X. Wang. Fragile mixing. In *ACM conference on Computer and Communications Security – CCS 2004*, pages 227–235. ACM, 2004.

[Rya06] P. Ryan. Prêt-à-Voter with Paillier encryption, 2006. Technical Report CS-TR No 965, School of Computing Science, Newcastle University. `http://www.cs.ncl.ac.uk/publications/trs/papers/965.pdf`.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sha79] A. Shamir. How to share a secret. In *Communications of the ACM, 22*, pages 612–613, 1979.

[Sid07] A. Sidorenko. *Design and Analysis of Provably Secure Pseudorandom Generators*. PhD thesis, Eindhoven University of Technology, 2007.

[SK95] K. Sako and J. Killian. Receipt-free mix-type voting scheme. In *EUROCRYPT 1995*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[SS07] B. Schoenmakers and A. Sidorenko. Distributed generation of uniformly random bounded integers, 2007. Unpublished Manuscript.

[SSW09] A. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *International Conference on Information and Communications Security–ICICS 2009*, Lecture Notes in Computer Science. Springer-Verlag, 2009. To appear.

[ST04] B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2004.

[ST06] B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encrypted values. In *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537. Springer-Verlag, 2006.

[Str64] E. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 71:806–808, 1964.

[Tof09a] T. Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 357–371. Springer-Verlag, 2009.

[Tof09b] T. Toft. Personal communication with the author, 2009.

[Tof09c] T. Toft. Solving linear programs using multiparty computation. In *Financial Cryptography – FC 2009*, volume 5628 of *Lecture Notes in Computer Science*, pages 90–107. Springer-Verlag, 2009.

[TW10] B. Terelius and D. Wikström. Proofs of restricted shuffles. In *AFRICACRYPT 2010*, Lecture Notes in Computer Science. Springer-Verlag, 2010. To appear.

[Vol99] H. Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer, 1999.

[WB09] R. Wen and R. Buckland. Minimum disclosure counting for the alternative vote. In *E-Voting and Identity – VOTE-ID 2009*, volume 5767 of *Lecture Notes in Computer Science*, pages 122–140. Springer-Verlag, 2009.

[Wik09] D. Wikström. A commitment-consistent proof of a shuffle. In *Australasian Conference Information Security and Privacy – ACISP 2009*, volume 5594 of *Lecture Notes in Computer Science*, pages 407–421. Springer-Verlag, 2009.

[Yao82] A. Yao. Protocols for secure computation. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[Yao86] A. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–168, 1986.

Summary

In modern cryptography, the problem of secure multiparty computation is about the cooperation between mutually distrusting parties computing a given function. Each party holds some private information that should remain secret as much as possible throughout the computation. A large body of research initiated in the early 1980's has shown that any computable function can be evaluated using secure multiparty computation. Though these feasibility results are general, their applicability in practical situations is rather unsatisfactory. This thesis concerns the study of two particular cryptographic primitives with focus on efficiency.

The first primitive studied is a generalization of verifiable shuffles of homomorphic encryptions, where the shuffler is only allowed to apply a permutation from a restricted set of permutations. In this thesis, we consider shuffles using permutations from a k-fragile set, meaning that any k input-output correspondences uniquely identify a permutation within the set. We provide verifiable shuffles restricted to the set of all rotations (1-fragile), affine transformations (2-fragile), and Möbius transformations (3-fragile). Applications of these special shuffles include fragile mixing, electronic elections, secure function evaluation using scrambled circuits, and secure integer comparison.

Two approaches for verifiable rotations are presented. On the one hand, we use properties of the Discrete Fourier Transform (DFT) to express in a compact way that a rotation is applied in a shuffle. The solution is efficient, but imposes some mild restrictions on the parameters to allow DFT to work. On the other hand, we present a general solution that does not impose any parameter constraint and works on any homomorphic cryptosystem. These protocols for rotations are used to build efficient shuffling protocols for affine and Möbius transformations.

The second primitive is secure integer comparison. In a general scenario, parties are given homomorphic encryptions of the bits of two integers and, after running a protocol, an encryption of a bit is produced, telling the result of the greater-than comparison of the two integers. This is a useful building block for higher-level protocols such as electronic voting, biometrics authentication or electronic auctions. A study of the relationship of other problems to integer comparison is given as well.

We present two types of solutions for integer comparison. Firstly, we consider an arithmetic circuit yielding secure protocols within the framework for multiparty computation based on threshold homomorphic cryptosystems. Our circuit achieves a good balance between round and computational complexities, when compared to the similar solutions in the literature. The second type of solutions uses a intricate approach where different building blocks are used. A full analysis is made for the two-party case where efficiency of the resulting protocols compares favorably to other solutions and approaches.

Samenvatting

Binnen de moderne cryptografie bestaat het onderzoeksgebied secure multiparty computation. In dit gebied onderzoekt men de samenwerking van elkaar wantrouwende personen om via een protocol een gegeven functie uit te rekenen. Elke persoon heeft een persoonlijke invoer voor de functie en wil deze gedurende de berekening zoveel mogelijk geheim houden. Een grote hoeveelheid onderzoek uit begin jaren '80 toont aan dat elke berekenbare functie op een dergelijke veilige manier berekend kan worden. Hoewel deze resultaten veelbelovend zijn, blijken de gebruikte technieken niet erg efficiënt te zijn. Dit proefschrift behandelt twee specifieke primitieven en concentreert daarbij op de efficiëntie.

De eerste primitieve is een generalisatie van het verifieerbaar herverdelen van homomorfe encrypties, waarbij de verdeler slechts gebruik mag maken van een beperkt aantal permutaties. Dit proefschrift gaat in op herverdelingen die gebruikmaken van k-fragiele permutaties, wat inhoudt dat de permutatie vastgelegd wordt door elk k-tupel van in- en uitvoer. We geven constructies voor het gebruik met rotaties (1-fragiel), affiene transformaties (2-fragiel) en Möbius-transformaties (3-fragiel). Deze specifieke constructies hebben toepassingen in fragiele mixnetwerken, elektronische verkiezingen en secure integer comparison.

We presenteren twee aanpakken voor verifieerbare rotaties. De eerste aanpak maakt gebruik van de eigenschappen van de discrete Fouriertransformatie (DFT) om de rotatie op een compacte manier vast te leggen. Dit levert een efficiënt protocol op, maar legt ook enkele beperkingen op aan het gebruikte scenario. De tweede aanpak geeft een algemene oplossing die geen beperkingen oplegt en met ieder homomorf cryptosysteem te gebruiken is. De op deze wijze verkregen constructies worden vervolgens gebruikt om efficiënte protocollen voor affiene en Möbius-transformaties te verkrijgen.

De tweede primitieve in dit proefschrift is het zogenaamde secure integer comparison. In dit protocol hebben de verschillende deelnemers toegang tot een encryptie van de bits van twee gehele. Ze willen graag de encryptie van een enkel bit berekenen dat aangeeft welk van de twee gehele getallen groter is. Dit is een handige bouwsteen voor uitgebreidere protocollen voor bijvoorbeeld elektronisch stemmen, elektronische veilingen en authenticatie met biometrieën. We bestuderen onder andere de relatie tussen integer comparison en andere problemen.

We geven twee soorten oplossingen. In de eerste plaats bekijken we aritmetische circuits. Dit levert protocollen op die passen binnen de multiparty compution gebaseerd op threshold homomorfic encryption. De gevonden circuits geven – in vergelijking met bestaande oplossingen – een goede balans tussen de rondecomplexiteit en de computationele complexiteit. De tweede oplossing is een complexe aanpak, waarbij een samenstelling van diverse cryptografische primitieven als bouwblok worden gebruikt. We maken een uitgebreide analyse in het geval van twee deelnemers; hier blijkt de efficiëntie van de resulterende protocollen beter blijkt te zijn dan bij bestaande oplossingen.

Acknowledgements

This thesis would not have been made without the help and support of many people. First and foremost, I would like to express my most sincere appreciation to my supervisor, Berry Schoenmakers, for his advice, support and patience during these years. I am very grateful to him for inviting me to do my Ph.D. in Eindhoven. I also thank Henk van Tilborg who, apart from being my promotor, has always supported me during these four years, and has given me a lot of nice feedback in this thesis. This work would not have been the same without their invaluable feedback.

The members of the reading committee of my Ph.D. thesis are Juan A. Garay, Ronald Cramer and Andries Brouwer. I appreciate their work reading my thesis and commenting on it. External members of the Ph.D. committee, Jorge Guajardo and Ahmad-Reza Sadeghi are also thanked for taking part in the defense of this dissertation. I especially thank Juan who gave me a lot of feedback that helped to improve this work significantly. He also guided, mentored and adviced me at earlier stages of Ph.D. studies. I didn't panic and hanged in there :-)

My Ph.D. studies were part of the PASC project ("Practical Approaches to Secure Computation") that has been funded by the Dutch Technology Foundation (STW). Apart from the financial support, we held regular meetings with the "user committee", composed people coming from the different partners supporting the project. These meetings provided quite good feedback in the direction that my research should aim at.

Some results included in this thesis have been discussed at the MPC club, a weekly gathering with the protocol guys of the coding and crypto group where we brainstorm and discuss interesting stuff. Special thanks go to Boris Škorić, Sebastiaan de Hoogh, Peter van Liesdonk and Tomas Toft for the many club sessions spent on my research problems. They helped to improve this work a lot! On top of this, Sebastiaan proof-read some chapters, and Peter helped me with a translation of the summary. I thank Dan Roozemond, Çiçek Güven and Maxim Hendiks for a last minute advice on bibliographic references concerning multiply sharply transitive groups.

I am grateful to all my colleagues of the crypto and security groups in special to all members of the "big Ph.D. office", HG. 8.79, including Reza, Peter B., Christianne, Peter van L., Sebastiaan, Jing, Peter S., Michael, and all the newcomers. They all have managed to make coming to the office very enjoyable, even when the office was very crowded. Together with the guys of the other big office, Bruno, Daniel, Antonino, Mayla, Relinde and Gaetan they all created a nice atmosphere at lunch time and during the afternoon tea-brakes.

I could never forget the support of Ellen, Reza, Andrey and Mehmet at early stages of my doctorate, thanks! Anita, Benne, Wil, Tanja, Dan and all the other members of the groups coding and crypto, security, and discrete algebra and geometry complemented a nice and unforgettable working atmosphere.

I am indebted to my family who, from the distance, have given me the strength and

support to face everyday. I want to thank my parents to have shaped me as I am now, encouraging me everyday to give a step forward and being there in good and especially in bad circumstances. With their strong and endless support, I would not have been able to be where I am now. There will never be a way to return all this.

Estoy totalmente agradecido a mi familia quienes, desde la distancia, me han dado la fuerza y el apoyo para poder hacer frente a cada día. Quiero agradecer a mis padres por haberme curtido de la manera que soy, dándome el valor para dar un paso adelante y estar ahí en las buenas y especialmente en las malas. Con su inacabable e infinito soporte, no hubiera sido capaz de estar en donde me encuentro ahora. Nunca encontraré la forma de devolverles todo esto.

I do not want to forget all nice people and friends I have met since I landed in Eindhoven. If you are reading this and feel identified in this group of people, I thank you for the hospitality, the many enjoyable moments, and understanding my busy schedule in the last year. A special thank go to Cipri Cornea and his co-author for some translation done in this thesis (see below).

And last but not least, I thank my wife Iulia. You have not only given me love and unconditional support, but a lovely family in Romania and a welcoming group of Romanian friends. I appreciate very much the passion you put on everything you do in life. You fill me and give me the confidence and strength I need everyday. Without your infinite support and encouragement, I would not have been able to manage in the stressful moments of writing this thesis. I will work hard to return everything you give me during the whole life together ahead of us...

Iar în final, îi mulţumesc soţiei mele Iulia. Mi-ai oferit nu doar dragoste şi sprijin necondiţionat, ci o familie minunată în România şi un grup primitor de prieteni români. Îţi apreciez foarte mult pasiunea cu care te ocupi de orice lucru pe care îl faci. Mă faci să mă simt întreg alături de tine, îmi oferi încrederea şi puterea de care am nevoie în fiecare zi. Fără necontenitul tău sprijin şi încurajare nu aş fi reuşit să trec cu bine momentele încordate din timpul scrierii tezei. Mă voi strădui pe tot parcursul vieţii ce ne aşteaptă împreună să iţi ofer şi eu ţie aceeaşi grijă, aceeaşi pasiune, aceeaşi dragoste pe care tu mi le oferi constant...

José Villegas
Eindhoven, March 2010

Curriculum Vitae

José Villegas was born on July 17, 1981 in Salta, Argentina. He finished his pre-university education at the Technical High School "Gral. Güemes" in Salta, and started his studies at the Salta National University in March 2000.

Starting with his second year in software engineering, he followed extra-curricular courses in discrete mathematics. José's interest in cryptography has led him to carry out the implementation of a software package providing cryptographic functionalities as his final project. He received his Bachelor's degree with honors in May 2003. In November 2005, after writing his master's thesis titled "Analysis and Implementation of Oblivious Transfer Protocols", José finished his Master's degree in computer science with honors.

He did different activities next to his university studies. He worked as an ICT assistant for the migration to Linux OS in the university (2002), as a teaching assistant (2002 – 2004), and was a member of the university council (2004). He worked as a database maintainer in Damesco Supermarkets (summer of 2005). In the winter of 2005 he joined a research project to analyze the warehouse management of Termoandes, a electric generation plant. During 2005, he followed different courses of the master's in applied discrete mathematics as a complement to his background.

In January 2006, José joined the Coding and Crypto group at the Eindhoven University of Technology. Until mid 2010 he was a PhD student under the supervision of dr.ir. Berry Schoenmakers.