NTNU
Norwegian University of
Science and Technology

Mixnets and Verifiable Shuffling

Åshild Bryn Damsgård

Master of Science
Submission date: June 2018
Supervisor: Kristian Gjøsteen, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Summary

In this thesis we consider different mix network protocols. First, we discuss a protocol called *cMix*. We describe the concept of verifiable shuffling, that enable mixnodes to prove that they operate correctly according to the protocol. We present three protocols for verifiable shuffling a list of encrypted elements, the *Naive protocol*, the *Simple n-shuffle* and the *Permutation matrix protocol*. The first two make use of the fact that polynomials remain invariant under permutation of their roots, and the last makes use of a permutation matrix. We discuss security achieved in all of our protocols, and explain how many exponentiations they require.

Sammendrag

I denne oppgaven ser vi på ulike mix nettverk protokoller. Vi diskuterer først en protokoll kalt *cMix*. Vi gir en beskrivelse av konseptet verifiserbar stokking, som gir mulighet for en mixnode å bevise at den følger den aktuelle protokollen korrekt. Vi presenterer tre protokoller som gir verifiserbar stokking av en liste med krypterte elementer, den *Naive protokollen*, den *Simple n-shuffelen* og *Permutasjonsmatrise protokollen*. De første to benytter seg av det faktum at polynomer er identiske under permutasjon av røttene, mens den siste benytter seg av en permutasjonsmatrise. Vi diskuterer sikkerheten som oppnås i alle våre protokoller, samt hvor mange eksponensieringer de krever.

Preface

This thesis represents the work of my final semester at NTNU. I would like to give a special thanks to my supervisor Kristian Gjøsteen. He has been a great source of inspiration, and given me excellent guidance. There has always been an open door, he has answered all my silly questions, and explained things in such a way that even I could grasp them.

So many friends have filled the time as student with so much more than studying. Thanks to all my classmates at ”Matteland”, this years would not have been nearly as great without you! Thanks to Kristin Asdal and Ellen Johanne Weydahl for proofreading this thesis. Finally, I would like to thank my family for always believing in me, and giving me a huge amount of support!

Trondheim, 31.05.2018

Åshild Bryn Damsgård

Contents

Chapter 1

Introduction

The main subject of this work is verifiable shuffling in mix networks. In this chapter we will give a brief introduction to mixnet, and previous work done in this field of cryptography. We will then explain the concept of verifiable shuffling. Finally, we give an overview of the outline of this thesis.

1.1 Mixnet

Mixnets are useful for applications which require anonymity, such as electronic voting. In these networks we consider a set of users that want to a send message to a set of receivers. The messages are encrypted by the senders and then relayed by a sequence of trusted intermediaries, called *mixnodes*. The mixnodes decrypt and randomly permute a batch messages. The last mixnode in the sequence outputs a permuted version of the original input sequence. This process makes it difficult to trace an individual message through the network [6]. A security aspect of mix networks is unlinkability between sender and receiver. Unlinkability means that no one is able to relate an output message to a users input message.

It has been developed a wide range of different mixnet protocols. The first, often referred to as *decryption mixnet*, was introduced by Chaum in 1981 [7]. In 1985, Pfitzmann and Waidner introduced *hybrid mixnets* which combine asymmetric and symmetric cryptography, and allows messages of arbitrary length [27]. Park, Itoh and Kurosawa were the first to describe a *re-encryption* mixnet. In this network each mixnode re-encrypts the input ciphertexts, instead of decrypting, by taking advantage of homomorphic properties of ElGamal encryption [26]. An offline-online approach of mixnet was introduced by Adida

and Wikström in 2007 [1]. Their protocol still requires several public-key operations in the online phase.

Recently, *cMix* was introduced as another mix network that makes use of an offline-online approach. This network was constructed by Chaum, Jevani, Das, Kate, Krasnova and Ruither in 2016 [6]. In this protocol, the online phase does not require any public-key operations. The network is claimed by its authors to be secure unless all mixnodes collude. In this thesis we will give a description of how this protocol is constructed, and discuss the security it achieves.

1.2 Verifiable shuffling

A huge threat to mixnet is that the mixnodes might be active cheaters, meaning they do not follow the protocol correctly. Protocols that provide verifiable shuffling, enable the mixnodes to prove that their output is constructed correctly. This can be proven to the other mixservers in the protocol, or to any interested independent verifier. This is an important property of mixnets, hence verifiability of shuffles has received much attention.

There are two main paradigms for proving correctness of a shuffle; one paradigm is based on polynomials being identical under permutation of their roots, and the other approach makes use of permutation matrices [18]. In this thesis we will look at two protocols that belongs to the first paradigm. One of these is a protocol given by Neff [24]. For the second paradigm we will look at a protocol that was constructed by Furukawa and Sako [13].

1.3 Outline of the thesis

In this thesis we will first consider the cMix protocol. The protocol receives messages from a group of senders, the messages are relayed by a sequence of mixnodes, and finally outputted to the receivers of the messages. Second, we will in Chapter 4 and 5 describe different protocols that provide verifiable shuffling. We stress that we then only consider one mixnode, and we explain how the mixnode proves its correctness. Finally, we compare the security achieved and exponentiations required in our protocols. This thesis is outlined as follows:

Chapter 2 We describe the theoretical background and notation used in the paper. We give definitions of zero knowledge, proof of knowledge, and commitment schemes.

Chapter 3 We describe the *cMix* protocol, and analyze its security.

Chapter 4 We describe the *Naive protocol* and the *Simple n-shuffle* by Neff, that are two verifiable shuffling protocols based on polynomials being identical under permutation of their roots. We look at the security for both of the protocols.

Chapter 5 We describe the *Permutation matrix protocol* by Furukawa and Sako, and look at the security the protocol achieves. This is a verifiable shuffling protocol that makes use of a permutation matrix.

Chapter 6 We compare and summarize the results from Chapter 3, 4 and 5.

Chapter 2

Theory

In this chapter we will present the theoretical background that is necessary and relevant for our protocols. We present notation, definitions and theorems that are used. We will first give a definition of indistinguishability, and then define the concept of zero knowledge and proof of knowledge. Further we give a definition of commitment schemes.

2.1 Indistinguishability

We say that two distributions are indistinguishable if it is hard to distinguish them. We have different levels of indistinguishability, described in the following definition [11]:

Definition 1. Given two distributions X and Y, we say that:

- X and Y are *perfectly indistinguishable* if X and Y have the same probability space,
- X and Y are *statistically indistinguishable* if the statistical distance between them is negligible. This means that there is a small advantage over a random guess of which of the distributions that produced an output,
- X and Y are *computationally indistinguishable* if no algorithm exists that can distinguish them. This means that it requires a lot of computational power to decide which of the distributions that produced an output.

2.2 Zero knowledge argument

A *zero knowledge argument* is an argument where we want to convince a player without leaking any information out of the transcript. We have given two algorithms, a prover $\mathcal{P}$

and a verifier $\mathcal{V}$.

We let X be a set, and L a subset of X called the language, $L \subseteq X$. We let W be a set of witnesses w, and E a relation such that $E \subseteq X \times W$. This gives us the following definition [9]:

Definition 2. We have a two party game ($\mathcal{P}$, $\mathcal{V}$) for a set L between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. $\mathcal{P}$ and $\mathcal{V}$ are given public input x, and w is given as private input to $\mathcal{P}$. This is an *interactive proof system* if the following are satisfied:

- *completeness*: For every $x \in L$ there exist w such that $(x, w) \in E$,
- *soundness*: If $x \notin L$, then for any $\mathcal{P}^*$, the verifier accepts with probability at most ϵ after interaction with $\mathcal{P}^*$.

We will use the notation $\mathcal{V}^*$ for an honest but curious verifier, and $\mathcal{P}^*$ for a cheating prover.

Interactive proof protocols can be *zero knowledge*. This concept was first introduced by Goldwasser, Micali and Rackoff [15]. Zero knowledge proofs allows $\mathcal{P}$ to convince $\mathcal{V}$ that a given statement is true, without revealing any information about the secrets to the verifier, or anybody else [9]:

Definition 3. The protocol ($\mathcal{P}$, $\mathcal{V}$) is *zero knowledge* if for any verifier $\mathcal{V}^*$ there exists an efficient simulator $\mathcal{S}$ such that the output construced by $\mathcal{S}$ is indistinguishable from the output constructed by ($\mathcal{P}$, $\mathcal{V}^*$).

The protocol can be perfectly, statistically or computationally zero knowledge depending on whether the output produced by $\mathcal{S}$ is perfectly, statistically or computationally indistinguishable from the output produced by ($\mathcal{P}$, $\mathcal{V}^*$).

If we only require the simulator to exist for the honest verifier $\mathcal{V}$, we obtain a significantly weaker property called honest verifier zero knowledge:

Definition 4. The protocol ($\mathcal{P}$, $\mathcal{V}$) is *honest verifier zero knowledge* (HVZK) if there exists an efficient simulator $\mathcal{S}$ such that the output constructed by $\mathcal{S}$ is indistinguishable form the output constructed by ($\mathcal{P}$, $\mathcal{V}$).

2.2.1 Σ-protocols

Σ-protocols are a particular type of three round zero knowledge proofs [10]. First, the prover sends a commitment α to the verifier. Second, the verifier answers with a random challenge β. Finally, the prover computes γ and sends the calculated value to the verifier. Σ-protocols fulfill the following three properties, with public input x [17]:

- *Completeness*: w is given as private input to $\mathcal{P}$. For every $x \in L$ there exist a w such that $(x, w) \in E$.
- *Special soundness*: From any x and any pair of accepting transcripts (α, β, γ) and (α, β', γ') where $\beta \neq \beta'$, one can efficiently compute w such that (x, w) $\in E$.
- *Special honest verifier zero knowledge* (SHVZK): There exists an efficient simulator $\mathcal{S}$, which on input x and challenge β outputs an accepted transcript (α, β, γ) indistinguishable from a real transcript constcuted by ($\mathcal{P}$, $\mathcal{V}$).

We note that SHVZK implies HVZK, and the special soundness property implies soundness.

2.3 Proof of knowledge

There are two kinds of interactive proofs: a proof of a mathematical statement, and a proof of knowledge. A proof of knowledge proves that the prover knows a secret that satisfies a certain predicate [23]. We let X, L, W and E be defined as above.

An interactive protocol ($\mathcal{P}$, $\mathcal{V}$) is a *proof of knowledge* if there exists an efficient extractor Ex with the following property: for every $\mathcal{P}^*$ with non-negligible probability of making $\mathcal{V}$ accept, Ex can use $\mathcal{P}^*$ and output with overwhelming probability w such that (x, w) $\in E$ [20].

Remark that proof of knowledge implies soundness if there exists a witness, since a proof of knowledge of a witness implies its existence. Special soundness implies proof of knowledge of the witness.

2.3.1 Rewinding

The concept of rewinding is a common proof technique in cryptography, that for instance can be used to prove that a protocol is sound. When rewinding is performed an extractor Ex can make use of $\mathcal{P}^*$ to obtain multiple accepted transcripts. Rewinding in a protocol that outputs an accepted transcript (α, β, γ) is performed as follows:

1. The protocol starts running as usual, and $\mathcal{P}^*$ outputs α. This state is saved.
2. Ex gives β to $\mathcal{P}^*$.
3. $\mathcal{P}^*$ sends γ as response to Ex. We have obtained an accepted transcript (α, β, γ).
4. Rewind back to after step 1 and before step 2.

5. Ex gives a new challenge $\beta' \neq \beta$ to $\mathcal{P}^*$.

6. $\mathcal{P}^*$ sends γ' as response to Ex. We have obtained an accepted transcript $(\alpha, \beta', \gamma')$.

We stress that the rewinding can be done multiple times. Rewinding requires that we are able to save states in the protocol, such that we can rewind back to the previously saved states [28].

2.4 Commitments

Commitments are at the heart of almost any construction of modern cryptography protocols. Making a commitment allows a player in a protocol to choose a value from a set, and make a commitment to his choice. Once the commitment is made, the player can no longer change his mind. The player can, but is not queried to, reveal his choice at a later stage [9].

We let M be a set of messages, and R a set of random numbers. To make a commitment c to message $m \in M$ with randomness $r \in R$, we write:

$$c = \text{commit}(m, r)$$

To open the commitment, the tuple (m, r) is revealed, and the verifier can check if $c = \text{commit}(m, r)$ [18].

There are two essential properties to any commitment scheme:

- *Binding property*: It should be hard for $\mathcal{P}$ to change the chosen value at a later stage.
- *Hiding property*: It must be hard for $\mathcal{V}$ to gain any additional information about the commitment.

Both of these properties can be either *unconditional* or *computational* [9]:

- *Unconditional binding*: Even with infinite computing power $\mathcal{P}$ cannot change his mind after committing. If $\mathcal{P}$ is committed to m using r, there is no pair (m',r') such that $\text{commit}(m, r) = \text{commit}(m', r')$.
- *Computational binding*: $\mathcal{P}^*$ is an algorithm that outputs two tuples (m,r) and (m', r'), where $\Pr[\text{commit}(m, r) = \text{commit}(m', r')] \leq \epsilon$.
- *Unconditional hiding*: A commitment to m reveals no information about m, even to an infinitely powerful $\mathcal{V}$. The distribution of commitments to m is perfectly indistinguishable from the distribution of commitments to m'.

- *Computational hiding*: A bounded $\mathcal{V}$ will have a hard time guessing what is inside a commitment. This means that the distribution of commitments to m is computationally indistinguishable from the distribution of commitments to m'.

Remark that at at the most one of the two properties can be unconditional at any time.

In this thesis we use homomorphic commitment as an essential part of of our schemes. We give the following definition [18]:

Definition 5. A *commitment scheme* is homomorphic if the following property holds $\forall$ $(m_0, m_1) \in M$ and $(r_0, r_1) \in R$:

$$\text{commit}(m_0 + m_1, r_0 + r_1) = \text{commit}(m_0, r_0)\text{commit}(m_1, r_1)$$

A similar definition can be given for *homomorphic encryption* [18]: An encryption scheme $\mathcal{E}$ is homomorphic if the following property holds for messages $\forall$ $(m_0, m_1) \in M$ and randomizers $(r_0, r_1) \in R$: $\mathcal{E}(m_0 m_1, r_0 + r_1) = \mathcal{E}(m_0, r_0)\mathcal{E}(m_1, r_1)$.

Most of the commitments made in the protocols of this thesis will require two exponentiations. To make a commitment to m with randomness r, we write commit$(m, r) = X^m Y^r$. For simplicity, we therefore assume that it requires two exponentiations to make a commitment, unless we want to make a commitment to zero, or the random value is zero.

2.5 Assumptions

The *Discrete logarithm assumption* and *Decisional Diffie-Hellman assumpion* are well known assumptions in cryptography that is often used as a basis to prove different security aspects. This thesis is not an exception. We will therefore define the assumptions here:

We let $\mathbb{G}$ be a cyclic group of order q, where g is a generator. The discrete logarithm assumption means that given g and $y \in \mathbb{G}$, it should be hard to find $X \in \mathbb{Z}_q$ such that $g^X = y$.

In the Decisional Diffe-Hellman (DDH) assumption r, t and z is chosen at random from $\mathbb{Z}_q$. If the DDH assumption is satisfied it should be hard to distinguish tuples on form (g^r, g^t, g^z) from (g^r, g^r, g^{rt}) [29].

Chapter 3

cMix

cMix is a mix network that was constructed by Chaum et al. in 2016 [6]. A huge advantage of this network is that all the computations that require exponentiation are done in an offline phase, called the *precomputation phase*. The senders participate only in the online *real-time phase*, that is carried out by use of fast multiplication. This lower the cryptographic latency of the network [6]. Wikström and Adida have also considered an online-offline approach to mixnets earlier, but their protocol still require several public-key operations in the online phase [1].

First, we will describe how the protocol is constructed. Our description will closely follow the description given by Chaum et al. [6]. Further we will analyze the security of the scheme. It is stated that the protocol achieves unlinkability between sender and receiver. But as we will see, a security proof of cMix is not written yet.

We stress that we in this chapter consider a mix network that enables a set of senders to send a message to a set of receivers, through a sequence of multiple mixnodes. In Chapter 4 and Chapter 5 on the other hand, we will consider protocols for verifiable shuffling, hence these protocols only examine *one* mixnode.

3.1 Construction

In the cMix protocol we assume we have a set of e senders, $(A_1, ..., A_e)$ that want to send a message through the network to a set of e' receivers $(B_1, ..., B_{e'})$. The network includes a sequence of h mixnodes $(N_1, ..., N_h)$. The network also include an additional entity called the *network handler* that performs non-sensitive computations such as computing values the mixnodes outputs. A brief overview of the network is given in Figure 3.1.

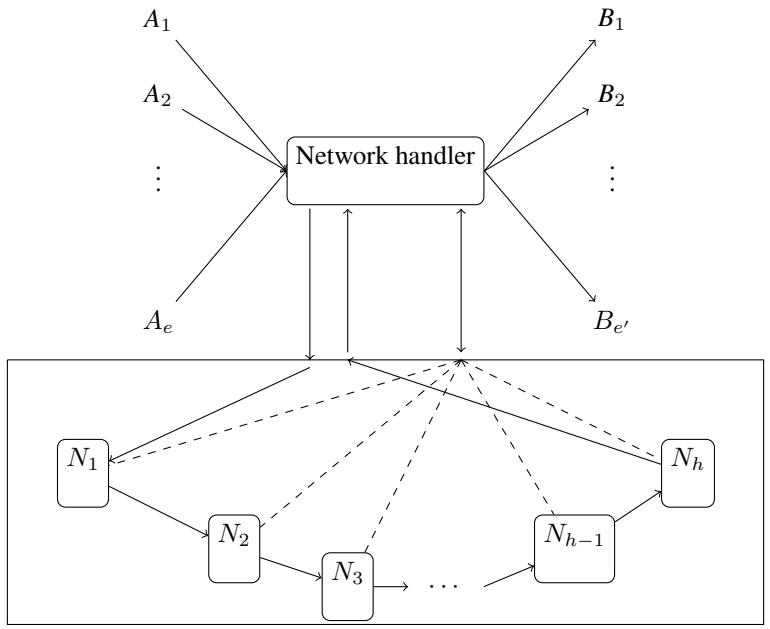

Figure 3.1: A cMix network with senders (A_j), receivers (B_j) and mixnodes (N_i).

The users share a secret key with each mixnode. The keys are multiplied with the users messages, and become the input to the protocol. The output of the protocol is a permuted version of the original input sequence. The messages are processed in large batches by the network handler, and we require that all the messages in a batch to have the same size. We let n be the size of the batch, where $n \leq e$. The goal of the protocol is to achieve unlinkability between senders and receivers, even though all senders and receivers in a batch are known.

We divide this section in two; first we describe the notation used in the protocol. Then we give a detailed description of how the protocol is constructed. We will in our construction only consider the forward path when messages are sent from senders to receivers, and not the return path.

3.1.1 Notation

Our computations are performed in a cyclic group $\mathbb{G}$ of prime order q, where g is a generator for this group. We assume $\mathbb{G}$ satisfies the DDH assumption. $\mathbb{G}^*$ is defined as the set of non-identity elements in $\mathbb{G}$. The protocol make use of a group homomorphic encryption scheme written by Balaloh [3], based on ElGamal.

Each mixnode N_i has a decryption share $X_i \in \mathbb{Z}_q^*$ of the secret key. The public key for encryption can be computed by use of all the secret shares: $y = \prod_{i=1}^{h} g^{X_i}$. Encryption and decryption of a message m goes as follows, where C is the ciphertext obtained and r is a random number from $\mathbb{Z}_q$:

- $C = (C_1, C_2) = \mathcal{E}(m) = (g^r, my^r)$. C_1 is called the *random component* of the ciphertext, and C_2 is called the *message component*.

- To decrypt (C_1, C_2) N_i computes a decryption share of C_1:

  $$\mathcal{D}_i(g^r) = g^{-rX_i}$$

  All the decryption shares are then multiplied with C_2 to receive the message:

  $$\prod_{i=1}^{h} g^{-rX_i} my^r = y^{-r} my^r = m$$

The encryption scheme is homomorphic. This gives us that if we encrypt or decrypt a vector of values, each value in the vector is encrypted or decrypted individually, and we achieve a new vector with ciphertexts.

As mentioned above, the messages are divided into batches of size n by the network handler. We let a $slot$ denote one of the messages in the batch, hence we have n slots in the batch. The following notation and values are used in our construction of the protocol:

- π_i is a random permutation used by N_i on the n slots in the batch.
- $\Pi_i(a)$ is the composition of all the permutation performed on a value a through i mixnodes.
$$\Pi_i = \begin{cases} \pi_1(a) & i = 1 \\ \pi_i(\Pi_{i-1}(a)) & 1 < i \leq h \end{cases}$$
- $k_{i,j} \in \mathbb{G}^*$ is the secret key shared between N_i and sender user of slot j.
- $\vec{k_i} = (k_{i,1}, ..., k_{i,n})$ is a vector with all the shared keys between N_i and senders for the n slots in the batch.
- $K_j \in \mathbb{G}^*$ is the product of all the shared keys for the user that sends slot j, $K_j = \prod_{i=1}^{h} k_{i,j}$
- $\vec{K} = (K_1, ..., K_n)$ is a vector of the computed products K_j from the users sending the n slots in the batch. $\vec{K}^{-1}$ is the inverse vector; $\vec{K}^{-1} = (K_1^{-1}, ..., K_n^{-1})$
- $m_j \in \mathbb{G}^*$ is the message sent by user A_j.
- $r_{i,a}$ and $t_{i,a}$ are random values used on slot a by N_i, fleshly generated for each round: $r_{i,a}, t_{i,a} \xleftarrow{r} \mathbb{G}^*$. This gives N_i two vectors of random values for the n slots: $\vec{r_i} = (r_{i,1}, ..., r_{i,n})$ and $\vec{t_i} = (t_{i,1}, ..., t_{i,n})$.
- $\vec{R_i}$ is the product of all the i first local random r values: $\vec{R_i} = \prod_{j=1}^{i} \vec{r_j}$.
- T_i is the product and permutation of all the i first local random t values:
$$\vec{T_i} = \begin{cases} \vec{t_i} & i = 1 \\ \pi_i(\vec{T_{i-1}}) \times \vec{t_i} & 1 < i \leq h \end{cases}$$

3.1.2 Protocol description

We will now give a detailed description of how the protocol is constructed. As mentioned, the protocol is divided in two phases, the precomputation phase and the real-time phase. We also descripe an initial setup phase. The goal in the precomputation phase is to calculate values that later can be used in the real-time phase. Through a number of calculations,

the association between the sender and receiver is hidden, before the message finally is delivered to the receiver.

Our protocol involves a network handler. We stress that this entity only performs calculations that later will become public so he does not learn any secret information. The computations done by the network handler could be replaced by an additional pass through the mixnet, but his will reduce the networks latency significantly.

Setup

The mixnodes establish their decryption share X_i, and the public key y is computed. Each user A_j will individually establish a symmetric key $k_{i,j}$ with each mixnode N_i in the network. This can be done using any (offline) key distribution method, e.g. by use of Diffie-Hellman. The mixnodes draw their random values $\vec{r}_i$ and $\vec{t}_i$ for the n slots.

Precomputation phase

The goal in this phase is to perform the public-key operations that is needed in the real-time phase. The calculations are done once for each real-time phase. We further divide this phase in three distinct steps: preprocessing, mixing and postprocessing. An illustration of the precomputation phase is given in Figure 3.2.

Step 1 - Preprocessing: Mixnode N_i computes $\mathcal{E}(\vec{r_i}^{-1})$, and send their calculated vector to the network handler. The network handler then computes $\mathcal{E}(\vec{R}_h^{-1}) = \prod_{i=1}^{h} \mathcal{E}(\vec{r_i}^{-1})$.

Step 2 - Mixing: $N_i (i = 1, ..., h-1)$ computes and sends the following to N_{i+1}:

$$\mathcal{E}(\Pi_i(\vec{R}_h^{-1}) \times \vec{T}_i^{-1}) = \begin{cases} \pi_1(\mathcal{E}(\vec{R}_h^{-1})) \times \mathcal{E}(\vec{t}_1^{-1}) & i = 1 \\ \pi_i(\mathcal{E}(\Pi_{i-1}(\vec{R}_h^{-1}) \times \vec{T}_{i-1}^{-1})) \times \mathcal{E}(\vec{t_i}^{-1}) & 1 < i < h \end{cases}$$

We see that $N_i (i = 1, ..., h-1)$ use their random permutation π_i to permute the vector they receive, and encrypt their local random values $\vec{t}_i^{-1}$. N_h finally computes: $(\vec{C}_1, \vec{C}_2) = \mathcal{E}((\Pi_h(\vec{R}_h) \times \vec{T}_h)^{-1}) = \pi_h(\mathcal{E}(\Pi_{h-1}(\vec{R}_h^{-1}) \times \vec{T}_{h-1}^{-1})) \times \mathcal{E}(\vec{t}_h^{-1})$. He sends $\vec{C}_1$ to the other mixnodes, and store $\vec{C}_2$ locally for use in the real-time phase.

Step 3 - Postprocessing: Mixnode N_i use their decryption share X_i to decrypt the vector of random components they received in the previous step; $\mathcal{D}_i(\vec{C}_1) = \vec{C}_1^{-X_i}$. They publish a commitment to their calculated decryption share.

Real-time phase

In this phase the senders are involved. A_j constructs a blinded message $m_j \times K_j^{-1}$. The blinded messages are the input to the protocol, and they are combinded by the network

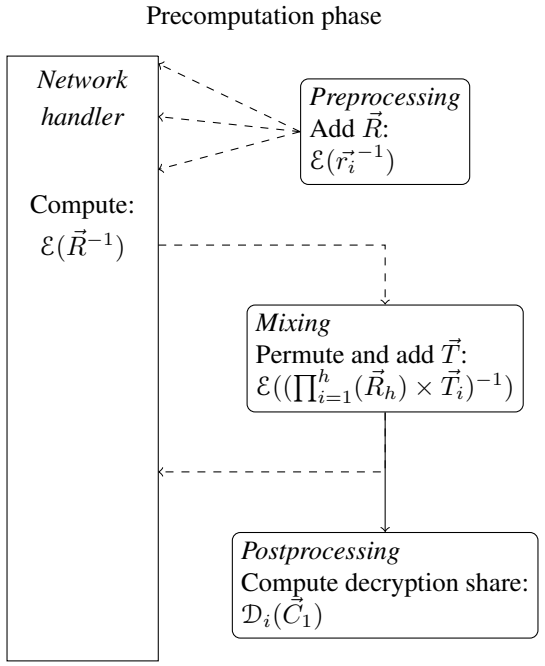

Figure 3.2: An illustration of the precomputation phase in cMix.

handler to yield the vector $\vec{m} \times \vec{K}^{-1}$ Analogously to the precomputation phase, we separate the real-time phase in three distinct steps: preprocessing, mixing and postprocessing. An illustration of the real-time phase is given in Figure 3.3.

Step 1 - Preprocessing: Every mixnode N_i calculates $\vec{k_i} \times \vec{r_i}$, and sends the resulting vector to the network handler. The network handler can then compute $\vec{m} \times \vec{R}_h = \vec{m} \times \vec{K}^{-1} \times \prod_{i=1}^{h} \vec{k}_i \times \vec{r}_i$, hence the $\vec{K}^{-1}$ vector is replaced with the random r values of each mixnode.

Step 2 - Mixing: The goal in this step is to hide the association between senders and receivers. $N_i (i = 1, ..., h-1)$ computes and sends the following to N_{i+1}:

$$(\Pi_i(\vec{m} \times \vec{R}_h) \times \vec{T_i}) = \begin{cases} \pi_1(\vec{m} \times \vec{R}_h) \times \vec{t_1} & i = 1 \\ \pi_i(\Pi_{i-1}(\vec{m} \times \vec{R}_h) \times \vec{T}_{i-1}) \times \vec{t_i} & 1 < i < h \end{cases}$$

We see that $N_i (i = 1, ..., h-1)$ use their random permutation π_i to permute the value they receive, and add their local vector of random values $\vec{t_i}$. Mixnode N_h computes $\Pi_h(\vec{m} \times \vec{R}_h) \times \vec{T_h} = \pi_h(\Pi_{h-1}(\vec{m} \times \vec{R}_h) \times \vec{T}_{h-1}) \times \vec{t}_h$. He commits to this vector, and sends his commitment to the remaining mixnodes.

Step 3 - Postprocessing: When mixnodes $N_i (i = 1, ..., h-1)$ receive the commitment from N_h, they send their decryption share $\mathcal{D}_i(\vec{C}_1)$ computed in the precomputation phase to the network handler. The last mixnode computes and send the following to the network handler: $\Pi_h(\vec{m} \times \vec{R}_h) \times \vec{T_h} \times \vec{C}_2 \times \mathcal{D}_h(\vec{C}_1)$. Due to the properties of the encryption scheme, the network handler can compute:

$$\begin{aligned} \Pi_h(\vec{m} \times \vec{R}_h) \times \vec{T_h} \times \vec{C}_2 \times \prod_{i=1}^{h} \mathcal{D}_i(\vec{C}_1) &= \Pi_h(\vec{m} \times \vec{R}_h) \times \vec{T_h} \times (\Pi_h(\vec{R}_n) \times \vec{T_h})^{-1} \\ &= \Pi_h(\vec{m}) \end{aligned}$$

The network handler outputs $\Pi_h(\vec{m})$, that is a permutation of the input messages.

3.2 Security

Chaum et al. have written a security analysis of the protocol in their paper. They state that the protocol provides unlinkability for all the n messages in the batch, even if the attacker compromises all but two users and all but one mixnode. This means that an attacker cannot map any input message to the corresponding output message [5]. For this reason, it is reasonable to believe that a possible attack of cMix does not exist. However, a formal security proof of the protocol is not written. It is possible that the authors of the cMix

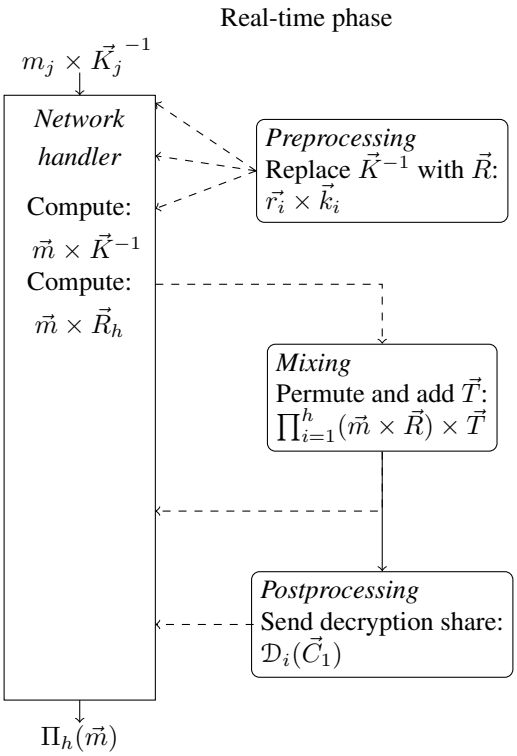

Figure 3.3: An illustration of the real-time phase in cMix.

paper have missed something in their security analysis. We can therefore not assure that the protocol is secure.

We will describe two possible attacks on the protocol that breaks the unlinkability property of cMix, originally written by Galteland, Mjølsnes and Olimid [14]; an *insider attack* and a *tagging attack*. We explain how these attacks can be prevented. Security mechanisms that hamper both the attacks are implemented in an updated publication of the cMix protocol [5].

Chaum et al. introduce an attacker $\mathcal{A}$ that can eavesdrop, forward and delete messages at any point in the protocol, but not modify, replay or inject new messages. They assume authenticated communication channels among the network handler and the mixnodes, and among all mixnodes. This means that the communication channels are resistant to tampering, but not necessarily resistant to overhearing.

3.2.1 Insider attack

We let $\mathcal{A}$ be an attacker that compromise mixnode N_h and the network handler. The attacker is then able to cancel all permutations added by the previous mixnodes, and perform the overall mixing himself. The output of the protocol will be a batch of messages permuted by an permutation known by $\mathcal{A}$, and he can easily break the unlinkability of the protocol.

Construction of the attack

The attack is constructed as follows:

1. In step 2 of the precomputation phase N_h ignores the input he receives from N_{h-1}, and chooses its own output. He draws a vector with random values $\theta_i \xleftarrow{r} \mathbb{G}_q$, $(i = 1, ..., n)$, and let $\vec{\theta}_h = (\theta_1, ..., \theta_n)$. The mixnode computes the ciphertext $(\vec{C}_1, \vec{C}_2) = \pi_h(\mathcal{E}(\vec{R}_h)^{-1} \times \mathcal{E}(\vec{\theta}_h^{-1}))$. He sends $\vec{C}_1$ to the other mixnodes.

2. In step 2 of the real-time phase N_h ignores the input he receives from N_{h-1}. He chooses $\pi_h(\vec{m} \times \vec{R}_h \times \vec{\theta}_h)$ to be his output, instead of $\pi_h(\Pi_{h-1}(\vec{m} \times \vec{R}_h) \times \vec{T}_{h-1}) \times \vec{t}_h$. The mixnodes commits to this, and sends the following to the network handler: $\pi_h(\vec{m} \times \vec{R}_h \times \vec{\theta}_h) \times \vec{C}_2 \times \mathcal{D}_h(\vec{C}_1)$.

3. The network handler computes the output of the protocol:

$$\pi_h(\vec{m} \times \vec{R}_h \times \vec{\theta}_h) \times \vec{C}_2 \times \prod_{i=1}^{h} \mathcal{D}_i(\vec{C}_1) = \pi_h(\vec{m} \times \vec{R}_h \times \vec{\theta}_h) \times \pi_h(\vec{R}_h^{-1} \times \vec{\theta}^{-1})$$
$$= \pi_h(\vec{m})$$

The output is only permuted with π_h, which is a permutation known by the attacker. The attacker can then easily apply the inverse permutation π_h^{-1}, and receive the corresponding input.

How the attack can be prevented

The insider attack can easily be prevented if *randomized partial checking* (RPC) is included in the protocol. RPC was first introduced by Jakobsson, Juels and Rivest [21]. RPC forces every mixnode to publicly show parts of its permutation. The mixnodes have to commit to its input and output, and then release a large fraction of its input-output pairs, e.g. half of them. Anyone can then verify if the mixnodes have operated correctly according to the protocol, and used the right input.

If RPC is implemented in the protocol, it will be detected if the last mixnodes ignores the input he receives in step 2 of the percomutation- and real-time phases, and chooses his own output.

3.2.2 Tagging attack

The main idea in this attack is for the attacker to put an identifier tag on an input message to the protocol, such that it is recognizable in the output. This breaks the protocols unlinkability between sender and receiver. We let $\mathcal{A}$ be an attacker that compromises mixnode N_h. The attacker wants to learn the receiver of message m_j. We assume that it is possible to determine whether an output message is valid.

Construction of the attack

1. N_h creates $\vec{s}$ that consist of $(n-1)$ 1's, and one tag $s \in \mathbb{G}^*$ in slot j. He computes $\vec{k}_h \times \vec{r}_h \times \vec{s}$.

2. In step 3 of the real-time phase, N_h receives all the decryption shares from the other

mixnodes. The last mixnode computes:

$$\Pi_h(\vec{m} \times \vec{s}) = \Pi_h(\vec{m} \times \vec{R}_h \times \vec{s}) \times \vec{T}_h \times \vec{C}_2 \times \prod_{i=1}^{h} \mathcal{D}_i(\vec{C}_1)$$

The attacker will then recognize the tagged message in slot j'.

3. N_h creates the inverse vector $\vec{s}^{-1}$, that consists of $(n-1)1$'s, and one tag s^{-1} in slot j'. N_h computes $\vec{C}_2' = \vec{C}_2 \times \vec{s}^{-1}$, and sends $\Pi_h(\vec{m} \times \vec{R}_h) \times \vec{T}_h \times \mathcal{D}_h(\vec{C}_1) \times \vec{C}_2'$ to the network handler.

4. The network handler computes:

$$\Pi_h(\vec{m} \times \vec{R}_h \times \vec{s}) \times \vec{T}_h \times \vec{C}_2' \times \prod_{i=1}^{h} \mathcal{D}_i(\vec{C}_1) =$$
$$\Pi_h(\vec{m} \times \vec{R}_h \times \vec{s}) \times \vec{T}_h \times (\Pi_h(\vec{R}_n) \times \vec{T}_h)^{-1} \times \vec{s}^{-1} = \Pi_h(\vec{m})$$

The network handler retrieves the permuted batch as normal, and he will not be able to determine whether the attack took place or not.

How the attack can be prevented

To detect this attack the last mixnode has to make a commitment to $\vec{C}_2$ in step 3 of the precomputation phase. This causes that N_h is not able to remove the tag, by replacing $\vec{C}_2$ with $\vec{C}_2'$. Hence, the tag will not be removed. Due to the assumption that it is possible to determine whether an output message is valid the users will be aware that a tagging attack took place. The attack can also be prevented by RPC. It is worth mentioning that the messages are blinded with random values at all locations in the path, except at the last mixnode. If a tagging attack is detected N_h should therefore be removed from the cascade.

3.3 Number of exponentiations

We will in this section compute how many exponentiations that are required in the cMix protocol. The establishment of the symmetric keys $k_{i,j}$ between user A_j and mixnode N_i is done in the setup phase. As mentioned, this key establishment can be done with any key distribution method, so it is unknown how many exponentiations this requires. This will therefore only include the exponentiations required in the precomputation- and real-time phases. We recall that encryption of a value requires 2 exponentations. Table 3.1 summarize the data we need.

Where	What N_i computes	Number of exponentiations performed by N_i
Precomutation step 1	$\mathcal{E}(\vec{r_i}^{-1})$	$2n$
Precomputation step 2	$\mathcal{E}(\Pi_i(\vec{R}_h^{-1}) \times \vec{T}_i^{-1})$	$2n$
Precomputation step 3	$\mathcal{D}_i(\vec{C}_1) = \vec{C}_1^{-X_i}$	n

Table 3.1: Exponentiations required in cMix.

This gives us that N_i has to perform $5n$ exponentiations to run the protocol for a batch of size n, and the total number of exponentiations required in the network of h mixnodes is $5nh$.

Chapter 4

Roots of polynomials

We will in this chapter and in Chapter 5 look at protocols for verifiable shuffling, and will only consider *one* of the mixnodes in the mix network. All our protocols consider that the mixnodes shuffles an input batch of size n. The idea of verifiable shuffling is that the mixnodes can prove that the output is computed correctly. We stress that both the input and the output sequences will be encrypted messages, therefore it is not trivial to show that the output is a permutation of the input.

The fundamental idea in this chapter is that two polynomials are identical under permutation of their roots. This means that if we have a polynomial $f(X) = \prod_{i=1}^{n}(X - i)$ and a permutation π, the following equality holds:

$$f(X) = \prod_{i=1}^{n}(X - i) = \prod_{i=1}^{n}(X - \pi(i))$$

The following theorem gives us that if two polynomials have the same value at a random point, the polynomials are with high probability equal [25]:

Theorem 1. *Let $f(X)$, $\tilde{f}(X) \in \mathbb{Z}_q$ be monic polynomials of degree at most d, with $f(X) \neq \tilde{f}(X)$. If t is selected at random from $\mathbb{Z}_q$ then,*

$$Pr[f(t) = \tilde{f}(t) | t \xleftarrow{r} \mathbb{Z}_q] \leq \frac{d-1}{q}$$

The idea of making use of roots of polynomials to verifiable shuffling a list of encrypted elements was first introduced by Neff [24]. Neff later published an improved version of his protocol [25]. Groth generalizes Neff's approach to an abstract protocol for any homomorphic cryptosystem [16, 18]. We will in this chapter first look at a simpler

protocol than the protocol of Neff, to understand the basic ideas. We have chosen to call this the *Naive protocol*. We will then look at the *Simple n-shuffle*, constructed by Neff. The security will be discussed for both of them. As we will see, the Naive protocol might be easier to understand, but it is very inefficient and requires a lot of work.

Notation

We assume p and q are two publicly known primes such that $p = kq + 1$. $\mathbb{G}_q$ is cyclic group of order q; $\mathbb{G}_q \subseteq \mathbb{Z}_p^*$. Let g be a generator for $\mathbb{G}$. The arithmetic operations are performed in the modular ring $\mathbb{Z}_p$, and n is a positive integer.

4.1 The Naive protocol

First, we construct the *Multiplication protocol* that is used as a building block in the Naive protocol. We assume we have a homomorphic commitment scheme that is unconditionally binding and computationally hiding.

4.1.1 The Multiplication protocol

This protocol allows $\mathcal{P}$ to convince $\mathcal{V}$ that a commitment c is a commitment to a product, without revealing any extra information. $\mathcal{P}$ is given private input a, b, u, r_a, r_b and r_u, where $u = ab$. Commitments c_a, c_b and c_u are made public. The Multiplication protocol is used so $\mathcal{P}$ can convince $\mathcal{V}$ that c_u is such that $c_u = \text{commit}(ab, r_u)$. This protocol is inspired by a similar construction written by Cramer, Damgård, Dziembowski, Hirt and Rabin [8].

Construction

The Multiplication protocol is a three move protocol, illustrated in Figure 4.1. The output of the protocol is (α, β, γ), where $\alpha = (c_\theta, c_{b\theta})$, $\beta = \beta$, and $\gamma = (d, s_1, s_2)$. If $\mathcal{V}$ accepts the protocol, $\mathcal{P}$ has accomplished to convince $\mathcal{V}$ that c_u is a commitment to the product of a and b.

Public input: p, q, g, c_a, c_b, c_u
Private input to $\mathcal{P}$: a, b, u, r_a, r_b, r_u

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\theta \xleftarrow{r} \mathbb{Z}_q$		
$r_\theta, r_{b\theta} \xleftarrow{r} \mathbb{Z}_q$		
$c_\theta \leftarrow \text{commit}(\theta, r_\theta)$		
$c_{b\theta} \leftarrow \text{commit}(b\theta, r_{b\theta})$	$\xrightarrow{c_\theta, c_{b\theta}}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{r} \mathbb{Z}_q$
Compute:		
$d \leftarrow \beta a + \theta$		
$s_1 \leftarrow \beta r_a + r_\theta$		
$s_2 \leftarrow d r_b - r_{b\theta} - \beta r_u$	$\xrightarrow{d, s_1, s_2}$	
		Verify :
		$c_a^\beta c_\theta = \text{commit}(d, s_1)$
		$c_b^d (c_{b\theta} c_u^\beta)^{-1} = \text{commit}(0, s_2)$

Figure 4.1: The Multiplication protocol.

Security

Theorem 2. *The Multiplication protocol is complete.*

Proof. The protocol is complete if $\mathcal{V}$ always accept when $\mathcal{P}$ follows the protocol correctly. $\mathcal{V}$ accepts if the following equations hold:

$$c_a^{\beta} c_\theta = \text{commit}(d, s_1)$$
$$c_b^{d}(c_{b\theta} c_u^{\beta})^{-1} = \text{commit}(0, s_2)$$

We assume that $\mathcal{P}$ follows the protocol correctly and that $u = ab$. Due to the fact that the commitment scheme is homomorphic, we have:

$$\begin{aligned} c_a^{\beta} c_\theta &= \text{commit}(\beta a + \theta, \beta r_a + r_\theta) = \text{commit}(d, s_1) \\ c_b^{d}(c_{b\theta} c_u^{\beta})^{-1} &= \text{commit}(db - b\theta - \beta u, dr_b - r_{b\theta} - \beta r_{ab}) \\ &= \text{commit}(\beta ab + b\theta - b\theta - \beta u, dr_b - r_{b\theta} - \beta r_u) = \text{commit}(0, s_2) \end{aligned}$$

The commitment scheme is unconditionally binding, so these equations will not hold for any other input. The equations are satisfied, and $\mathcal{V}$ will accept. We have proven that the Multiplication protocol is complete. □

Theorem 3. *The Multiplication protocol has special soundness.*

Proof. We assume we have two accepted transcripts (c_θ, $c_{b\theta}$, β, d, s_1, s_2) and (c_θ, $c_{b\theta}$, β', d', s_1', s_2'), $\beta \neq \beta'$. We need to prove that we are able to extract a, b and u. First, we prove that a easily can be computed:

$$\begin{aligned} d - d' &= \beta a + \theta - \beta' a - \theta = (\beta - \beta')a \\ a &= \frac{(d - d')}{(\beta - \beta')} \end{aligned}$$

Second, we use the fact that $db - b\theta - \beta u = 0$ and $d'b - b\theta - \beta' u = 0$, to extract b and u:

$$\begin{aligned} db - b\theta - \beta u &= d'b - b\theta - \beta' u \\ (d - d')b &= (\beta - \beta')u \end{aligned}$$

This gives us that u and b easily can be computed, and we can conclude that $ab = u$. Hence, the protocol has special soundness. □

We note that this implies that the protocol is a proof of knowledge of a, b and u such that $u = ab$.

Theorem 4. *The Multiplication protocol is SHVZK.*

Proof. We will construct a simulator $\mathcal{S}$ that gets $(p, q, g, c_a, c_b, c_u, \beta)$ as its input, and outputs an accepted transcript (α, β, γ). The protocol is SHVZK if transcripts constructed by $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}$ have the same distribution, hence it is impossible to distinguish if (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ or $\mathcal{S}$. Figure 4.4 shows how $\mathcal{S}$ is constructed.

Input: $p, q, g, c_a, c_b, c_u, \beta$

Simulator, $\mathcal{S}$
$d, s_1, s_2 \xleftarrow{r} \mathbb{Z}_q$
Compute:
$c_\theta \leftarrow c_a^{-\beta}\text{commit}(d, s_1)$
$c_{b\theta} \leftarrow c_u^{-\beta} c_b^d (\text{commit}(0, s_2))^{-1}$

Figure 4.2: Construction of a simulator for the Multiplication protocol.

The simulator outputs (α, β, γ), where $\alpha = (c_\theta, c_{b\theta})$, $\beta = \beta$ and $\gamma = (d, s_1, s_2)$. Table 4.1 summarize how $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}$ constructs their transcripts.

$(\mathcal{P}, \mathcal{V})$	$\mathcal{S}$
$\theta \xleftarrow{r} \mathbb{Z}_q$ $r_\theta, r_{b\theta} \xleftarrow{r} \mathbb{Z}_q$ Compute c_θ and $c_{b\theta}$ Compute d, s_1 and s_2	$d, s_1, s_2 \xleftarrow{r} \mathbb{Z}_q$ Compute c_θ and $c_{b\theta}$
Transcript: $(c_\theta, c_{b\theta}, \beta, d, s_1, s_2)$	Transcript: $(c_\theta, c_{b\theta}, \beta, d, s_1, s_2)$

Table 4.1: Construction of transcripts by $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}$.

Let $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$ be an accepted transcript. We define two following two probabilities p_1 and p_2:

$$p_1 = \Pr[(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = (\alpha, \beta, \gamma) | (\alpha, \beta, \gamma) \leftarrow (\mathcal{P}, \mathcal{V})]$$
$$p_2 = \Pr[(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = (\alpha, \beta, \gamma) | (\alpha, \beta, \gamma) \leftarrow \mathcal{S}]$$

We note that there is only one possible output for $(\mathcal{P}, \mathcal{V})$ when $(\theta, r_\theta, r_{b\theta})$ is chosen at random from $\mathbb{Z}_q$, and one possible output for $\mathcal{S}$ when (d, s_1, s_2) is chosen at random from

$\mathbb{Z}_q$, due to the fact that the commitment scheme is unconditionally binding. We can use this to compute p_1:

$$\begin{aligned} p_1 &= \Pr[(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = (\alpha, \beta, \gamma) | (\theta, r_\theta, r_{b\theta}) \leftarrow \mathbb{Z}_q] \\ &= \frac{1}{q} \cdot \frac{1}{q} \cdot \frac{1}{q} = \frac{1}{q^3} \end{aligned}$$

and p_2:

$$\begin{aligned} p_2 &= \Pr[(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) = (\alpha, \beta, \gamma) | (d, s_1, s_2) \leftarrow \mathbb{Z}_q] \\ &= \frac{1}{q} \cdot \frac{1}{q} \cdot \frac{1}{q} = \frac{1}{q^3} \end{aligned}$$

Due to the fact that $p_1 = p_2$, we are not able to distinguish if $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$ was constructed by ($\mathcal{P}$, $\mathcal{V}$) or $\mathcal{S}$, and the protocol is SHVZK. □

We note that the protocol is perfect SHVZK. The Multiplication protocol satisfies completeness, has special soundness and is SHVZK, hence the protocol is a Σ-protocol according to the definition given in Section 2.2.1.

Theorem 5. *The number of exponentiations required in the Multiplication protocol is 11.*

Proof. Exponentiations required are summarized in Table 4.2.

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$	
Computation	Exponentiations	Computation	Exponentiations
c_θ	2	$c_a^\beta c_\theta$	1
$c_{b\theta}$	2	$c_b^d (c_{b\theta} c_u^\beta)^{-1}$	3
		commit(d, s_1)	2
		commit(0,s_2)	1
Total: 4		Total: 7	

Table 4.2: Exponentiations required in the Multiplication protocol.

□

4.1.2 The Naive protocol

The Naive protocol is a protocol that provides verifiable shuffling. A permutation π and two sequences $(X_1, ..., X_n)$ and $(r_1, ..., r_n)$ are given as private input to $\mathcal{P}$. Commitments (c_i) such that $c_i = \text{commit}(X_i, r_i)$ are made public. In addition a sequence $(y_1, ..., y_n)$

is made public. $\mathcal{P}$ is required to convince $\mathcal{V}$ that there exists a permutation π with the property that:

$$y_i = X_{\pi(i)} \tag{4.1}$$

without revealing any extra information about (X_i).

Construction

The protocol is a four move protocol, illustrated in Figure 4.3. We assume we have two polynomials $f(X)$ and $\tilde{f}(X)$:

$$f(X) = \prod_{i=1}^{n}(X - X_i)$$
$$\tilde{f}(X) = \prod_{i=1}^{n}(X - y_i)$$

We let $t \xleftarrow{r} \mathbb{Z}_q$, and define a_i and b_i as follows:

$$b_i = t - X_i$$
$$a_i = \prod_{j=1}^{i}(t - X_j)$$

In the protocol $\mathcal{V}$ makes a commitent c_0 to the random value t with randomness 0. $\mathcal{P}$ wants to make a commitent to $f(t)$. This is done by first making commitments to every factor in $f(t)$: $\tilde{c}_i = c_0/c_i = \text{commit}(t - X_i, r_i)$, due to the fact that the commitment scheme is homomorphic. Further, $\mathcal{P}$ computes commitments $\hat{c}_i = \text{commit}(a_i, \hat{r}_i) = \text{commit}(\prod_{j=1}^{i}(t - X_j), \hat{r}_i)$, that are commitments to the i-th first factors in $f(t)$. For $\mathcal{P}$ to prove that his computations are performed correctly, he execute the Multiplication protocol from Section 4.1.1. He can then prove that for $(i = 2, ..., n)$:

$$\hat{c}_i = \text{commit}(a_{i-1}b_i, \hat{r}_i) = \text{commit}(\prod_{j=1}^{i-1}(t - X_j)(t - X_i), \hat{r}_i).$$

If the computations are done correctly, the final commitment $\hat{c}_i = \text{commit}(a_n, \hat{r}_n)$ is a commitment to $f(t)$. Hence $a_n = f(t)$. $\mathcal{V}$ receives a_n, and can easily verify if $a_n = \tilde{f}(t)$ when (y_i) are given as public input. This gives us a four move protocol with accepted transcript $(t, c_0, a_n, (c_{\theta_i}), (c_{b\theta_i}), (\beta_i), (d_i), (s_{1,i}), (s_{2,i}))$.

Public input: $p, q, g, n, (y_i), (c_i)\ (i = 1, ..., n)$
Private input to $\mathcal{P}$: $(X_i), (r_i)\ (i = 1, ..., n), \pi$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
		$t \xleftarrow{r} \mathbb{Z}_q$
		$c_0 \leftarrow \text{commit}(t, 0)$
	$\xleftarrow{t, c_0}$	
$\tilde{c}_i \leftarrow c_0/c_i, (i = 1, ..., n)$		$\tilde{c}_i \leftarrow c_0/c_i, (i = 1, ..., n)$
$\hat{r}_i \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n)$		
Let $\hat{c}_1 = \tilde{c}_1$		
Compute :		
$\hat{c}_2 \leftarrow \text{commit}(a_2, \hat{r}_2)$		
$\vdots$		
$\hat{c}_n \leftarrow \text{commit}(a_n, \hat{r}_n)$		
$\theta_i \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n)$		
$r_{\theta_i}, r_{b\theta_i} \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n)$		
Compute$(i = 2, ..., n)$:		
$c_{\theta_i} \leftarrow \text{commit}(\theta_i, r_{\theta_i})$		
$c_{b\theta_i} \leftarrow \text{commit}(b_i\theta_i, r_{b\theta_i})$		
	$\xrightarrow{a_n, (c_{\theta_i})_{i=2}^n, (c_{b\theta_i})_{i=2}^n}$	
		$\beta_i \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n)$
	$\xleftarrow{(\beta_i)_{i=2}^n}$	
Compute$(i = 2, ..., n)$:		
$d_i \leftarrow \beta_i a_{i-1} + \theta_i$		
$s_{1,i} \leftarrow \beta_i \hat{r}_{i-i} + r_{\theta_i}$		
$s_{2,i} \leftarrow d_i r_i + r_{b\theta_i} + \beta_i \hat{r}_i$		
	$\xrightarrow{(d_i)_{i=2}^n, (s_{1,i})_{i=2}^n, (s_{2,i})_{i=2}^n}$	
		Verify $\forall (i = 2, ..., n)$:
		$\hat{c}_{i-1}^{\beta_i} c_{\theta_i} = \text{commit}(d_i, s_{1,i})$
		$\tilde{c}_i^{d_i} (c_{b\theta_i} \hat{c}_i^{\beta_i})^{-1} = \text{commit}(0, s_{2,i})$
		$a_n = \tilde{f}(t)$

Figure 4.3: The Naive protocol.

Security

Theorem 6. *The Naive protocol is complete.*

Completeness can easily be proved with simple calculations. We stress that the commitment scheme is unconditionally binding by our assumptions, so the equations verified by $\mathcal{V}$ will only hold for the attempted values.

Theorem 7. *The Naive protocol is sound.*

Proof. We need to prove that if $\mathcal{V}$ accepts the Naive protocol, there exists a permutation π such that (4.1) is satisfied.

Assume $\tilde{f}(t) = f(t)$ such that $\mathcal{V}$ accepts. Theorem 1 gives us that $f(X) = \tilde{f}(X)$, when t is chosen at random, except with negligible probability. We know that polynomials are identical under permutation of their roots, hence there exists a permutation π such that $y_i = X_{\pi(i)}$ $(i = 1, ..., n)$. □

Theorem 8. *The Naive protocol is a proof of knowledge of (4.1).*

Proof. We will with rewinding prove that we are able to extract the permutation π that satisfies (4.1). We assume that we have an extractor Ex that can use $\mathcal{P}^*$:

1. t and c_0 is given as input to $\mathcal{P}^*$.
2. $\mathcal{P}^*$ outputs a_n, (c_{θ_i}) and $(c_{b\theta_i})$ $(i = 2, ..., n)$.
3. $\mathcal{P}^*$ is given random challenges (β_i) $(i = 2, ..., n)$.
4. Let $\mathcal{P}^*$ compute and outputs (d_i), $(s_{1,i})$ and $(s_{2,i})$ $(i = 2, ..., n)$.
5. Rewind to after step 2 and before step 3.
6. Give $\mathcal{P}^*$ new random challenges (β'_i), $(\beta_i) \neq (\beta'_i)$ $(i = 2, ..., n)$.
7. Let $\mathcal{P}^*$ compute and output (d'_i), $(s'_{1,i})$ and $(s'_{2,i})$ $(i = 2, ..., n)$.

We then have two accepted transcripts $(t, c_0, a_n, (c_{\theta_i}), (c_{b\theta_i}), (\beta_i), (d_i), (s_{1,i}), (s_{2,i}))$ and $(t, c_0, a_n, (c_{\theta_i}), (c_{b\theta_i}), (\beta'_i), (d'_i), (s'_{1,i}), (s'_{2,i}))$. If $\mathcal{V}$ accepts, we know that the following equations hold:

$$
\begin{aligned}
\hat{c}_1 &= \text{commit}(a_1, \hat{r}_1) = \text{commit}(t - X_1, \hat{r}_1) \\
\hat{c}_2 &= \text{commit}(a_2, \hat{r}_2) = \text{commit}(a_1 b_2, \hat{r}_2) = \text{commit}(a_1(t - X_2), \hat{r}_2) \\
&\vdots \\
\hat{c}_n &= \text{commit}(a_n, \hat{r}_n) = \text{commit}(a_{n-1} b_n, \hat{r}_n) = \text{commit}(a_{n-1}(t - X_n), \hat{r}_n)
\end{aligned}
$$

In the proof of Theorem 3 we proved that we are able to extract $(a_i), (a_{i-1})$ and (b_i), from the two accepted transcripts. This gives us that we are able to compute (X_i):

$$\begin{aligned} X_1 &= t - a_1 \\ X_2 &= t - a_2(a_1)^{-1} \\ &\vdots \\ X_n &= t - a_n(a_{n-1})^{-1} \end{aligned}$$

When (X_i) $(i = 1, ..., n)$ are computed, the permutation that satisfies $y_i = X_{\pi(i)}$ can easily be extracted. This proves that the protocol is a proof of knowledge of (4.1). □

We note that proof of knowledge implies that the protocol is sound, but soundness can also be proven directly (similarly to the proof of Theorem 7). Soundness means that the witness π exists when $\mathcal{V}$ accepts, and proof of knowledge means that the $\mathcal{P}^*$ knows the witness π when $\mathcal{V}$ accepts.

Theorem 9. *Let $\mathcal{A}$ be an attacker that can distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ in the Naive protocol, or by a simulator $\mathcal{S}$ for the Naive protocol. Then Adv$\mathcal{A}$ is negligible.*

Proof. In Figure 4.4 we construct a simulator $\mathcal{S}$ for the Naive protocol that outputs an accepted transcript $(t, c_0, a_n, (c_{\theta_i}), (c_{b\theta_i}), (\beta_i), (d_i), (s_{1,i}), (s_{2,i}))$.

We note that commitments $\hat{c}_i (i = 2, ..., n-1)$ are commitments to random values (a_i) and $\hat{c}_n$ is a commitment to $\tilde{f}(t)$, such that the transcript is accepted. Due to the fact that the commitment scheme is computationally binding it requires a lot of computational power for $\mathcal{A}$ to distinguish if an accepted transcript was constructed by $\mathcal{S}$ or by $(\mathcal{P}, \mathcal{V})$. Hence, the advantage of the attacker is negligible. □

We note that this implies that the protocol is computationally HVZK.

Theorem 10. *The number of exponentiations required in the Naive protocol is* $(13n-12)$.

Proof. Exponentiations required are summarized Table 4.3. This gives a total of $(13n - 12)$ exponentiations to run the protocol. □

4.2 Neff's shuffle

The protocol of Neff was one of the first efficient proof of shuffles [30]. First, we construct *ILMPP* and then the *Simple n-shuffle protocol*. ILMPP is not a protocol that shuffle the

Input: $p, q, g, n, (y_i), (c_i)\ (i = 1, ..., n)$

Simulator, $\mathcal{S}$

$t, \beta_i \xleftarrow{r} \mathbb{Z}_q (i = 1, ..., n)$
$\hat{r}_i, d_i, s_{1,i}, s_{2,i} \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n)$
$a_i \xleftarrow{r} \mathbb{Z}_q (i = 2, ..., n-1)$
$a_n = \tilde{f}(t)$
Compute:
$c_0 \leftarrow \text{commit}(t, 0)$
$\tilde{c}_i \leftarrow c_0 / c_i, (i = 1, ..., n)$
$\hat{c}_2 \leftarrow \text{commit}(a_2, \hat{r}_2)$
$\vdots$
$\hat{c}_n \leftarrow \text{commit}(a_n, \hat{r}_n)$
$c_{\theta_i} \leftarrow \hat{c}_{i-1}^{-\beta_i} \text{commit}(d_i, s_{1,i}) (i = 2, ..., n)$
$c_{b\theta_i} \leftarrow \tilde{c}_i^{d_i} \hat{c}_i^{-\beta_i} (\text{commit}(0, s_{2,i}))^{-1} (i = 2, ..., n)$

Figure 4.4: Construction of a simulator for the Naive protocol

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$	
Computation	Exponentiations	Computation	Exponentiations
$\hat{c}_i, (i = 2, ..., n)$	$2(n-1)$	c_0	1
Multiplication protocol	$4(n-1)$	Multiplication protocol	$7(n-1)$
$(n-1)$ times		$(n-1)$ times	
Total: $6n - 6$		Total: $7n - 6$	

Table 4.3: Exponentiations required in the Multiplication protocol.

input elements, but is used to convince a verifier that given two sets of group elements (x_i) and (y_i), then $\prod_{i=1}^{n} log_g x_i = \prod_{i=1}^{n} log_g y_i$. The Simple n-shuffle makes use of the ILMPP and is useful when we want to commit to a permutation. We will follow Neff [24, 25] in our description.

4.2.1 Iterated logarithmic multiplication proof protocol (ILMPP)

In ILMPP two sequences of n elements are given as public input: $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$, where $(x_i) \neq 1$ and $(y_i) \neq 1$. The sequences $(X_i, ..., X_n)$ and $(Y_i, ..., Y_n)$ are given as private input to $\mathcal{P}$, where $X_i = log_g x_i$ and $Y_i = log_g y_i$. The prover wants to convince the verifier of the relation:

$$\prod_{i=1}^{n} X_i = \prod_{i=1}^{n} Y_i \tag{4.2}$$

without revealing any extra information about the secret logarithms.

Construction

The protocol, illustrated in Figure 4.5, is a three move protocol, where $(\alpha, \beta, \gamma) = ((D_i), \beta, (s_i))$ is given as output. We define two sets of equations:

$$\begin{aligned} D_1 &\leftarrow y_1^{\theta_1} \\ D_2 &\leftarrow x_2^{\theta_1} y_2^{\theta_2} \\ &\vdots \\ D_{n-1} &\leftarrow x_{n-1}^{\theta_{n-2}} y_{n-1}^{\theta_{n-1}} \\ D_n &\leftarrow x_n^{\theta_{n-1}} \end{aligned} \tag{4.3}$$

and,

$$\begin{aligned} y_1^{s_1} &\leftarrow D_1 x_1^{-\beta} \\ x_2^{s_1} y_2^{s_2} &\leftarrow D_2 \\ &\vdots \\ x_{n-1}^{s_{n-2}} y_{n-1}^{s_{n-1}} &\leftarrow D_{n-1} \\ x_n^{s_{n-1}} &\leftarrow D_n y_n^{(-1)^{n-1}\beta} \end{aligned} \tag{4.4}$$

Public input: p, q, g, n, (x_i), (y_i) $(i = 1, ..., n)$
Private input to $\mathcal{P}$: (X_i), (Y_i) $(i = 1, ..., n)$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\theta_i \xleftarrow{r} \mathbb{Z}_q (i = 1, ..., n-1)$		
Compute:		
$D_i (i = 1, ..., n)$ that satisfy(4.3)	$\xrightarrow{(D_i)_{i=1}^{n}}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{r} \mathbb{Z}_q$
Compute:		
$s_i (i = 1, ..., n-1)$ that satisfy(4.4)	$\xrightarrow{(s_i)_{i=1}^{n-1}}$	
		Verify if (4.4) holds

Figure 4.5: ILMPP.

Security

Theorem 11. *The ILMPP protocol is complete.*

Proof. We assume (4.2) is satisfied. The protocol achieves completeness if $\mathcal{P}$ always can find $(s_1, ..., s_{n-1})$ that satisfies (4.4), given arbitrary $(\theta_1, ..., \theta_{n-1})$ and β. To show that this is the case, we first take log_g on each side of the equations in system (4.4) and obtain:

$$\begin{aligned} s_1 Y_1 &= \theta_1 Y_1 - \beta X_1 \\ s_1 X_2 + s_2 Y_2 &= \theta_1 X_2 + \theta_2 Y_2 \\ &\vdots \\ s_{n-2} X_{n-1} + s_{n-1} Y_{n-1} &= \theta_{n-2} X_{n-1} + \theta_{n-1} Y_{n-1} \\ s_{n-1} X_n &= \theta_{n-1} X_n + (-1)^{n-1} \beta Y_n \end{aligned}$$

This can be expressed in the following $n \times (n-1)$ system of linear equations:

$$\begin{pmatrix} Y_1 & 0 & 0 & \cdots & 0 & 0 \\ X_2 & Y_2 & 0 & \cdots & 0 & 0 \\ 0 & X_3 & Y_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & X_{n-1} & Y_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & X_n \end{pmatrix} \begin{pmatrix} s_1 - \theta_1 \\ s_2 - \theta_2 \\ s_3 - \theta_3 \\ \vdots \\ s_{n-2} - \theta_{n-2} \\ s_{n-1} - \theta_{n-1} \end{pmatrix} = \begin{pmatrix} -\beta X_1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ (-1)^{n-1}\beta Y_n \end{pmatrix}$$

$$P_1 \vec{S_1} = \vec{J_1} \tag{4.5}$$

We are interested in whether column vectors in this system are linearly independent or dependent. We therefore look at a $(n-1) \times (n-1)$ sub-system, where the first row of the matrix P_1 and vectors $\vec{S_1}$ and $\vec{J_1}$ is removed:

$$\begin{pmatrix} X_2 & Y_2 & 0 & \cdots & 0 & 0 \\ 0 & X_3 & Y_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & X_{n-1} & Y_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & X_n \end{pmatrix} \begin{pmatrix} s_2 - \theta_2 \\ s_3 - \theta_3 \\ \vdots \\ s_{n-2} - \theta_{n-2} \\ s_{n-1} - \theta_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ (-1)^{n-1}\beta y_n \end{pmatrix}$$

$$P_2 \vec{S_2} = \vec{J_2} \tag{4.6}$$

We observe that P_2 is an upper triangular matrix, which gives us that $\det(P_2) = \prod_{i=2}^{n} X_i$. We know that the determinant is non-zero because of the assumption that $(x_i) \neq 1$. This is equivalent with the fact that the matrix is non-singular and the $(n-1)$ column vectors in P_2 are linearly independent. Hence the column vectors in P_1 are linearly independent.

Next, we look at the following matrix:

$$Q = \begin{pmatrix} X_1 & Y_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & X_2 & Y_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & X_3 & Y_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & X_{n-1} & Y_{n-1} \\ (-1)^n Y_n & 0 & 0 & 0 & \cdots & 0 & X_n \end{pmatrix}$$

$$Q = \begin{pmatrix} \vec{V_1}, & \vec{V_2}, & \cdots, & \vec{V_{n-1}}, & \vec{V_n} \end{pmatrix}$$

We observe that $\det(Q) = \prod_{i=1}^{n} X_i - \prod_{i=1}^{n} Y_i$. By our assumption we know that $\det(Q) = 0$. This is equivalent with the fact that Q is a singular matrix and that the column vectors $(\vec{V_i})$ are dependent. We have already proved that the column vectors $(\vec{V_2}, ..., \vec{V_n})$ are linearly independent, hence $\vec{V_1}$ has to be a linear combination of $(\vec{V_2}, ..., \vec{V_n})$. This implies that it has to exist a vector $\vec{S_1}$ that satisfies (4.5), and the protocol is complete. □

Theorem 12. *The ILMPP protocol is a proof of knowledge of (4.2).*

Proof. The theorem is constructed similarly to the proof of Theorem 8, but we will outline the main differences. We rewind and obtain three accepted transcripts: $((D_i), \beta, (s_i))$ and $((D_i), \beta', (s'_i))$ and $((D_i), \tilde{\beta}, (\tilde{s}_i))$. We need to prove that $(X_1, ..., X_n)$ and $(Y_1, ..., Y_n)$ can be extracted.

We know that (4.4) holds for each of the accepted transcripts. This gives us the following two sets of equations:

$$\begin{aligned}
(s'_1 - s_1)Y_1 &= (\beta - \beta')X_1 \\
(s'_2 - s_2)Y_2 &= (s_1 - s'_1)X_2 \\
&\vdots \\
(s'_{n-1} - s_{n-1})Y_{n-1} &= (s_{n-2} - s'_{n-2})X_{n-1} \\
(-1)^n(\beta - \beta')Y_n &= (s'_{n-1} - s_{n-1})X_n
\end{aligned}$$

and,

$$\begin{aligned}
(\tilde{s}_1 - s_1)Y_1 &= (\beta - \tilde{\beta})X_1 \\
(\tilde{s}_2 - s_2)Y_2 &= (s_1 - \tilde{s}_1)X_2 \\
&\vdots \\
(\tilde{s}_{n-1} - s_{n-1})Y_{n-1} &= (s_{n-2} - \tilde{s}_{n-2})X_{n-1} \\
(-1)^n(\beta - \tilde{\beta})Y_n &= (\tilde{s}_{n-1} - s_{n-1})X_n
\end{aligned}$$

We have $2n$ equations, with $2n$ unknowns: $(X_i)_{i=1}^n$ and $(Y_i)_{i=1}^n$. The logarithms can easily be extracted, hence the protocol is a proof of knowledge of (4.2). □

Theorem 13. *The ILMPP protocol is SHVZK.*

Proof. We will prove that given β a simulator $\mathcal{S}$ for ILMPP can construct an accepted transcript that is indistinguishable from the real transcript constructed by $(\mathcal{P}, \mathcal{V})$. Construction of $\mathcal{S}$ is illustrated in Figure 4.6.

Input: p, q, g, n, β, (x_i), (y_i) $(i = 1, ..., n)$

Simulator, $\mathcal{S}$
$s_i \overset{r}{\leftarrow} \mathbb{Z}_q (i = 1, ..., n-1)$
Compute:
$D_1 \leftarrow x_1^{\beta} y_1^{s_1}$
$D_2 \leftarrow x_2^{s_1} y_2^{s_2}$
$\vdots$
$D_{n-1} \leftarrow x_{n-1}^{s_{n-2}} y_{n-1}^{s_{n-1}}$
$D_n \leftarrow x_n^{s_{n-1}} y_n^{(-1)^n \beta}$

Figure 4.6: Construction of a simulator for ILMPP.

The simulator outputs the accepted transcript $((D_i), \beta, (s_i))$. It is clear that a randomly generation of θ_i from $\mathbb{Z}_q$ $(i = 1, ..., n-1)$ gives the same distribution as a randomly generation of s_i from $\mathbb{Z}_q$ $(i = 1, ..., n-1)$. Hence, $\mathcal{S}$ will perfectly simulate ($\mathcal{P}$,$\mathcal{V}$) and the protocol is SHVZK. □

We note that this gives that the protocol is perfect SHVZK.

Theorem 14. *The number of exponentiations required in ILMPP is* $3n$.

Proof. We note that $(D_i)(i = 2, ..., n-1)$ can be computed as: $g^{\theta_{i-1} X_i + \theta_i Y_i}$. This gives that it requires n exponentiations for $\mathcal{P}$ compute $(D_i)(i = 1, .., n)$. It requires $2n$ exponentiations for $\mathcal{V}$ to verify if (4.4) hold. This gives that a total of $3n$ exponentiations are required in ILMPP. □

4.2.2 The Simple n-shuffle

We have two public known sequences $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$, where $(x_i, y_i) \neq 1$. The sequences $(X_i, ..., X_n)$ and $(Y_i, ..., Y_n)$ are given as private input to $\mathcal{P}$, where $X_i = log_g x_i$ and $Y_i = log_g y_i$. Constants k, $l \in \mathbb{Z}_q$ are known to $\mathcal{P}$, and commitments $K = g^k$ and $L = g^l$ are made public.

In this protocol $\mathcal{P}$ is required to convince $\mathcal{V}$ that there exists a permutation π with the property that $(i = 1, ..., n)$:

$$y_i^l = x_{\pi(i)}^k \tag{4.7}$$

without revealing any information about X_i, Y_i, k, l or π.

Construction

We define two polynomials:

$$f(X) = \prod_{i=1}^{n}(\frac{X_i}{l} - X)$$
$$\tilde{f}(X) = \prod_{i=1}^{n}(\frac{Y_i}{k} - X)$$

For simplification, we make the following two assumptions:

1. $X_i \neq X_j$ and $Y_i \neq Y_j$ for $i \neq j$.
2. $X_i \neq 1 (i = 1, ..., n)$.

The Simple n-shuffle is a five move protocol, that goes as follows:

1. $\mathcal{V}$ generates $t \xleftarrow{r} \mathbb{Z}_q$, and sends this challenge to $\mathcal{P}$.
2. $\mathcal{P}$ and $\mathcal{V}$ publicly compute:
   - $F \leftarrow L^t = g^{lt}$
   - $H \leftarrow K^t = g^{kt}$
   - $\hat{x}_i \leftarrow x_i/F, (i = 1, ..., n)$
   - $\hat{y}_i \leftarrow y_i/H, (i = 1, .., n)$
3. $\mathcal{P}$ and $\mathcal{V}$ execute the ILMPP protocol for the two sequences of length $2n$:

   $$\phi \leftarrow (\hat{x}_1, ..., \hat{x}_n, K, ..., K), \text{with } n\text{-K's}$$

   $$\psi \leftarrow (\hat{y}_1, ..., \hat{y}_n, L, ..., L), \text{with } n\text{-L's}$$

4. $\mathcal{V}$ accepts if it accepts ILMPP for ϕ and ψ.

We stress that the verifier will accept if:

$$(log_g K)^n \prod_{i=1}^{n} log_g \hat{x}_i = (log_g L)^n \prod_{i=1}^{n} log_g \hat{y}_i$$
$$\prod_{i=1}^{n}(X_i - lt)k = \prod_{i=1}^{n}(Y_i - kt)l$$
$$\prod_{i=1}^{n}(\frac{X_i}{l} - t) = \prod_{i=1}^{n}(\frac{Y_i}{k} - t)$$

Hence, $\mathcal{V}$ will accept if $f(t) = \tilde{f}(t)$

Security

Theorem 15. *The Simple n-shuffle is complete, HVZK and is a proof of knowledge of (4.7).*

This follows immediately from proof of Theorems 11, 12 and 13. We proved that ILMPP is perfect SHVZK. The SHVZK property is only defined for Σ-protocols, hence the Simple n-protocol will be perfect HVZK. Soundness can also be proved directly, similarly to proof of Theorem 7.

Theorem 16. *The number of exponentiations required to in the Simple n-shuffle is* $(6n + 2)$.

Proof. It requires 2 exponentiations to compute F and H. The input sequences ϕ and ψ to ILMPP is of length $2n$. Theorem 14 gives us that it requires $2n$ exponentiations to construct ILMPP with this input, and $4n$ exponentiations to verify. This gives that a total of $(6n + 2)$ exponentitations are required in the protocol. □

Chapter 5

Permutation matrices

The other paradigm for verifiable shuffling uses permutation matrices. It has been written a wide range of different protocols that belongs to this paradigm. Each protocol is based on a theorem stating that a permutation matrix A_{ij} is a permutation matrix if and only if certain conditions holds. The differences between the approaches are the conditions that state that the matrix actually is a permutation matrix [22].

This research line was first introduced by Furukawa and Sako [13]. In their approach you first make a commitment to a permutation matrix. Next you prove that the matrix actually is a permutation matrix, and prove that the input sequence is shuffled correctly according to this permutation [16].

We will in this chapter give a description of the original protocol of Furukawa and Sako from 2001 [13], and look at its security. This approach has later been optimized and generalized by Furukawa [12].

Notation

We assume p and q are two publicly known primes such that $p = kq + 1$. $\mathbb{G}_q$ is cyclic group of order q, $\mathbb{G}_q \subseteq \mathbb{Z}_p^*$. Let g be a generator for $\mathbb{G}_q$. Arithmetic operations are performed in the modular ring $\mathbb{Z}_p$, and n is a positive integer. We will specify when we calculate exponents, hence the computations are performed $\mathrm{mod} q$. The protocol makes use of a ElGamal cryptosystem, with public keys (p, q, g, y) and secret key $X \in \mathbb{Z}_q$ such that $y = g^X \bmod p$.

5.1 Basic ideas

A ciphertext C_i is defined as $C_i = (g_i, m_i)$, where g_i and m_i both have order q. A re-encryption $C'_i = (g'_i, m'_i)$ of this ciphertext can be computes as:

$$g'_i = g^{r_i} g_{\pi^{-1}(i)}$$
$$m'_i = y^{r_i} m_{\pi^{-1}(i)}$$

where r_i is chosen at random from $\mathbb{Z}_q$.

First in this section, we give a definition of permutation matrices. Second, we describe how the shuffling can be performed by use of a permutation matrix, and describe the two properties that needs to be shown in order to prove correctness of a shuffle. Finally, we will give an outline of how the protocol is constructed.

5.1.1 Permutation matrix

A permutation matrix A_{ij} is defined as follows:

Definition 6. Let q be a prime and π a permutation. A matrix A_{ij} is a permutation matrix over $\mathbb{Z}_q$ if the following holds $(i, j = 1, ..., n)$:

$$A_{ij} = \begin{cases} 1 \bmod q & \text{if } \pi(i) = j \\ 0 \bmod q & \text{otherwise} \end{cases}$$

The following theorem will be very useful in our construction of the protocol:

Theorem 17. *A matrix A_{ij} is a permutation matrix if and only if for all i,j, and k, both of the following equations holds:*

$$\sum_{h=1}^{n} A_{hi} A_{hj} = \begin{cases} 1 \bmod q & \textit{if } i = j \\ 0 \bmod q & \textit{otherwise} \end{cases} \tag{5.1}$$

$$\sum_{h=1}^{n} A_{hi} A_{hj} A_{hk} = \begin{cases} 1 \bmod q & \textit{if } i = j = k \\ 0 \bmod q & \textit{otherwise} \end{cases} \tag{5.2}$$

We will give a proof of Theorem 17. For convenience we introduce the following definition:

Definition 7. δ_{ij} and δ_{ijk} is defined as follows:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \delta_{ijk} = \begin{cases} 1 & \text{if } i = j = k \\ 0 & \text{otherwise} \end{cases}$$

Proof. We assume (5.1) and (5.2) hold, and want to prove that A_{ij} is a permutation matrix. We will first prove that it is exactly one non-zero element e in each row vector and each column vector of A_{ij}, and then argue that $e = 1$.

Let $\vec{V_i}$ be the i-th column vector of the matrix A_{ij}, $\vec{V_i} = (A_{1i}, ..., A_{ni})$. Let $(\vec{V_i}, \vec{V_j})$ be the inner product of $\vec{V_i}$ and $\vec{V_j}$, $(\vec{V_i}, \vec{V_j}) = \delta_{ij}$. This implies that we have n orthogonal vectors, and rank$(A_{ij}) = n$. Hence, there are least one non-zero element e in each row vector and each column vector. We observe that $\vec{V_i} \neq \vec{0} \; \forall \; i$, from the fact that $(\vec{V_i}, \vec{V_i}) = 1$.

Next, we consider a vector $\vec{V_j} \odot \vec{V_k} = (A_{1j}A_{1k}, ..., A_{nj}A_{nk})$, and observe that $(\vec{V_i}, \vec{V_j} \odot \vec{V_k}) = \delta_{ijk}$. We assume $j \neq k$, which gives us $(\vec{V_i}, \vec{V_j} \odot \vec{V_k}) = 0 \forall i$. This implies that $\vec{V_j} \odot \vec{V_k} = \vec{0}$, for $j \neq k$. This means for any h, j, k such that $j \neq k$ either $A_{hj} = 0$ or $A_{hk} = 0$. Hence, we know that there are a maximum one non-zero element in each row vector. Combining this with the result above, we know that there is exactly one non-zero element e in each row vector of A_{ij}, and a total number of n non-zero elements in the matrix. Since $\vec{V_i} \neq \vec{0} \; \forall \; i$, the number for non-zero elements in each column vector also equals one.

We let e_i be the non-zero element in the i-th row. From (5.1) we get that $e_i^2 = 1 \bmod q$, and (5.2) gives us that $e_i^3 = 1 \bmod q$. Hence $e_i = 1 \bmod q$, and we have shown that the matrix A_{ij} is a permutation matrix over $\mathbb{Z}_q$.

To prove that (5.1) and (5.2) holds when A_{ij} is a permutation matrix is clear from inspection of Definition 6. □

5.1.2 Correctness of shuffle

We note that a permutation matrix A_{ij} can be used to compute a re-encryption pair $C_i' = (g_i', m_i')$ of a ciphertext $C_i = (g_i, m_i)$:

$$g_i' = g^{r_i} \prod_{j=1}^{n} g_j^{A_{ji}} \tag{5.3}$$

$$m_i' = y^{r_i} \prod_{j=1}^{n} m_j^{A_{ji}} \tag{5.4}$$

For a mixnode to show correctness of a shuffle, the following two properties have to be shown:

1. For each pair $\{(g_i', m_i')\}$ the same r_i and A_{ij} has been used.

2. A_{ij} is a permutation matrix.

To prove the first property, we will use a standard technique from Brands [4]. To prove the second property, we prove given (g_i) and (g'_i) it there exits a matrix A_{ij} satisfying (5.1) and (5.2) and (r_i) such that (5.3) is satisfied.

5.1.3 Outline of the Permutation matrix protocol

The goal in this chapter is to construct a protocol for verifiable shuffling a list of encrypted elements. This is done by first constructing protocols I, II, III and IV, and then run them in parallel in the *Permutation matrix protocol*.

Protocol I Prove that given (g_i) and (g'_i), (g'_i) can be expressed as (5.3), by use of (r_i) and a matrix that satisfy (5.1).

Protocol II Prove that given (g_i) and (g'_i), (g'_i) can be expressed as (5.3), by use of (r_i) and a matrix that satisfy (5.2).

Protocol III Prove that (r_i) and the matrix used in Protocol I and Protocol II are identical.

Protocol IV For each pair (g'_i, m'_i) the same r_i and (A_{ij}) has been used.

5.1.4 Security of the protocols

The following definition and theorems are later used in the security analysis of our protocols:

Definition 8. Define R_n^m to be the set of tuples of $n \times m$ elements in $\mathbb{G}_q$: $I = (x_1^{(1)}, ..., x_1^{(m)}, x_2^{(1)}, ..., x_2^{(m)}, ..., x_n^{(1)}, ..., x_n^{(1)})$. E_n^m is defined to be the subset of R_n^m satisfying:

$$log_{x_1^{(1)}} x_1^{(i)} = log_{x_j^{(1)}} x_j^{(i)},$$

for all $i (i = 1, ..., n)$, and j $(j = 2, ..., n)$.

Theorem 18. *For any n $\geq$ 2 and m $\geq$ 2, let $\mathcal{F}_1$ be an attacker that can distinguish uniform instances from E_n^2 and R_n^2, and $\mathcal{F}_2$ an attacker that can distinguish uniform instances from E_n^m and R_n^m. Then:*

$$Adv\mathcal{F}_2 \leq Adv\mathcal{F}_1$$

.

Theorem 19. *Let $\mathcal{F}_1$ be an attacker that can distinguish uniform instances from E_n^2 and R_n^2, for any $n \geq 2$, and $\mathcal{A}$ an attacker that can solve DDH. Then:*

$$Adv\mathcal{F}_1 \leq Adv\mathcal{A}$$

.

We will not provide the proves of these theorems here, but a sketch can be found in the paper by Furukawa and Sako for the interested reader [13].

5.2 Protocol I

We will in this section first construct Protocol I, and then look at the security of the protocol. Sequences (g_i) and (g'_i) are made public, and a sequence (r_i) and A_{ij} is given as private input to $\mathcal{P}$. The protocol proves that both (5.1) and (5.3) hold.

Construction

The main idea is for $\mathcal{P}$ to calculate $s_0 = \sum_{j=1}^{n} r_j\beta_j$ and $s_i = \sum_{j=1}^{n} A_{ij}\beta_j (i = 1, ..., n)$, and give $\gamma = (s_0, (s_i))$ as response to a challenge (β_i). $\mathcal{V}$ can then verify if:

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = \prod_{j=1}^{n} {g'_j}^{\beta_j} \tag{5.5}$$

$$\sum_{i=1}^{n} s_i^2 = \sum_{j=1}^{n} \beta_j^2 \bmod q \tag{5.6}$$

Apparently, this will leak information about A_{ij}. To prevent this, randomizers ψ, (ψ_i), and σ are added, and (5.5) and (5.6) are modified.

First, we add randomizers ψ and (ψ_i). $\mathcal{P}$ sends $\alpha = (g', (F_i), H)$ as commitments in advance, where $F_i = \sum_{j=1}^{n} 2\psi_j A_{ji}$, $H = \sum_{j=1}^{n} \psi_j^2$, and $g' = g^{\psi} \prod_{i=1}^{n} g_i^{\psi_i}$.

Given a challenge (β_i), $\mathcal{P}$ computes $s_0 = \sum_{j=1}^{n} r_j\beta_j + \psi$ and $s_i = \sum_{j=1}^{n} A_{ij}\beta_j + \psi_i (i = 1, ..., n)$, and sends $\gamma = (s_0, (s_i))$ to $\mathcal{V}$. The verifier will accept if the following equations hold:

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = g' \prod_{j=1}^{n} {g'_j}^{\beta_j}$$

$$\sum_{i=1}^{n} s_i^2 = \sum_{j=1}^{n} \beta_j^2 + \sum_{j=1}^{n} F_j\beta_j + H \bmod q$$

Further, we add randomizer σ and (5.6) is modified to be:

$$\sum_{i=1}^{n} s_i^2 + \sigma s_0 = \sum_{j=1}^{n} \beta_j^2 + \sum_{j=1}^{n} (F_j + \sigma r_j)\beta_j + H + \sigma\psi \bmod q$$

This verification is computed over exponents in order the hide the values of $(F_j + \sigma r_j)$, $(H + \sigma\psi)$ and σ. This gives us the two verification equations:

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = g' \prod_{j=1}^{n} g_j'^{\beta_j} \tag{5.7}$$

$$w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} = \hat{w} \prod_{j=1}^{n} \hat{w}_j^{\beta_j} \tag{5.8}$$

The protocol is three move, illustrated in Figure 5.1. The output of the protocol is (α, β, γ), where $\alpha = (w, g', (\hat{w}_i), \hat{w})$, $\beta = (\beta_i)$ and $\gamma = (s_0, (s_i))$. The prover computes g'. We enumerate the following equation for convenience:

$$g' = g^{\psi} \prod_{j=1}^{n} g_j^{\psi_j} \tag{5.9}$$

Security

Theorem 20. *Protocol I is complete.*

Completeness of the protocol can be proved with simple calculations.

Theorem 21. *If $\mathcal{V}$ accepts Protocol I with non-negligible probability, then $\mathcal{P}^*$ either knows both (r_i) and A_{ij} that satisfy (5.1), or can generate non-trivial integers (a_i) and a satisfying $g^a \prod_{i=1}^{n} g_i^{a_i} = 1$ with overwhelming probability.*

Proof. First, we prove that if $\mathcal{V}$ accepts with non-negligible probability, then $\mathcal{P}^*$ knows r_i, A_{ij}, ψ and (ψ_i) that satisfy (5.3) and (5.9). Recall that (5.3) gives that $g_i' = g^{r_i} \prod_{j=1}^{n} g_j^{A_{ji}}$.

This can be proved with rewinding. The technique is described in detail in proof of Theorem 8. First, we start running the protocol and $\mathcal{P}^*$ outputs $\alpha = (w, g', (\hat{w}_i), \hat{w})$. $\mathcal{P}$ is given challenge $(\beta_{i,1})$, and outputs $\gamma_1 = (s_{0,1}, s_{i,1})$. This gives us one accepted transcript $(\alpha, \beta_{i,1}, \gamma_1)$.

We perform rewinding n times. Each time is $\mathcal{P}^*$ is given challenge $(\beta_{i,b})$, $(\beta_{i,1}) \neq (\beta_{i,b})$, and $\mathcal{P}^*$ outputs the corresponding output $\gamma_b = (s_{0,b}, (s_{i,b}))(b = 2, ..., n+1)$,

Public input: p, q, g, n, (g_i), (g_i') $(i = 1, ..., n)$
Private input to $\mathcal{P}$: (r_i), A_{ij} $(i, j = 1, ..., n)$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\psi, \psi_i, \sigma \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$		
Compute:		
$w \leftarrow g^{\sigma}$		
$g' \leftarrow g^{\psi} \prod_{j=1}^{n} g_j^{\psi_j}$		
$\hat{w}_i \leftarrow g^{\sum_{j=1}^{n} 2\psi_j A_{ji} + \sigma r_i} (i = 1, ..., n)$		
$\hat{w} \leftarrow g^{\sum_{j=1}^{n} \psi_j^2 + \sigma\psi}$		
	$\xrightarrow{w, g', (\hat{w}_i)_{i=1}^{n}, \hat{w}}$	
		$\beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
	$\xleftarrow{(\beta_i)_{i=1}^{n}}$	
Compute :		
$s_0 \leftarrow \sum_{j=1}^{n} r_j \beta_j + \psi \bmod q$		
$s_i \leftarrow \sum_{j=1}^{n} A_{ij} \beta_j + \psi_i \bmod q, (i = 1, ..., n)$		
	$\xrightarrow{s_0, (s_i)_{i=1}^{n}}$	
		Verify if (5.7) and (5.8) hold

Figure 5.1: Protocol I.

$(i = 1, ..., n)$. This gives us a total of $(n + 1)$ accepted transcripts:

$$\begin{aligned}(\alpha, \beta_1, \gamma_1) &= (w, g', (\hat{w}_i), \hat{w}, (\beta_{i,1}), s_{0,1}, (s_{i,1}))\\ (\alpha, \beta_2, \gamma_2) &= (w, g', \{\hat{w}_i\}, \hat{w}, (\beta_{i,2}), s_{0,2}, (s_{i,2}))\\ \vdots \quad &= \vdots\\ (\alpha, \beta_{n+1}, \gamma_{n+1}) &= (w, g', (\hat{w}_i), \hat{w}, (\beta_{i,n+1}), s_{0,n+1}, (s_{i,n+1}))\end{aligned}$$

First, we extract (r_i) by making use of the fact that $s_{0,b} = \sum_{j=1}^{n} r_j \beta_{j,b} + \psi$. This gives us the following system of equations:

$$\begin{pmatrix} \beta_{1,1} & \beta_{2,1} & \cdots & \beta_{n,1} & 1 \\ \beta_{1,2} & \beta_{2,2} & \cdots & \beta_{n,2} & 1 \\ \vdots & \vdots & \cdots & \vdots & \\ \beta_{1,n+1} & \beta_{2,n+1} & \cdots & \beta_{n,n+1} & 1 \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \\ \psi \end{pmatrix} = \begin{pmatrix} s_{0,1} \\ s_{0,2} \\ \vdots \\ s_{0,n+1} \end{pmatrix} \tag{5.10}$$

The matrix to the left will be invertible except with negligible probability, when (β_i) is chosen at random from $\mathbb{Z}_q$. Hence $(r_1, r_2, ..., r_n, \psi)$ can be extracted.

Further, we extract (A_{ij}) by making use of the fact that $s_{i,b} = \sum_{j=1}^{n} A_{ij} \beta_{j,b} + \psi_i$. This gives us the following system of equations:

$$\begin{pmatrix} \beta_{1,1} & \beta_{2,1} & \cdots & \beta_{n,1} & 1 \\ \beta_{1,2} & \beta_{2,2} & \cdots & \beta_{n,2} & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \beta_{1,n+1} & \beta_{2,n+1} & \cdots & \beta_{n,n+1} & 1 \end{pmatrix} \begin{pmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \\ \psi_1 & \psi_2 & \cdots & \psi_n \end{pmatrix} = \begin{pmatrix} s_{1,1} & s_{2,1} & \cdots & s_{n,1} \\ s_{1,2} & s_{2,2} & \cdots & s_{n,2} \\ \vdots & \vdots & \cdots & \vdots \\ s_{1,n+1} & s_{2,n+1} & \cdots & s_{n,n+1} \end{pmatrix} \tag{5.11}$$

The matrix to the left will be invertible except with negligible probability, hence (A_{ij}) and (ψ_i) can be extracted. If $\mathcal{V}$ accepts, the same (r_i), A_{ij}, ψ and (ψ_i) are used to calculate g' and (g'_i) by (5.3) and (5.9) respectively. This proves $\mathcal{P}^*$'s knowledge of (r_i), A_{ij}, ψ and (ψ_i) such that (5.3) and (5.9) are satisfied. Next, we will prove that this knowledge implies that either (5.1) is satisfied, or $\mathcal{P}^*$ is able to find a non-trivial representation of 1

with overwhelming probability.

First, we look at (5.7), the first verification equation. Assume that $\mathcal{P}^*$ knows $(r_i), A_{ij}, \psi$ and (ψ_i) satisfying (5.3) and (5.9), and (a_i) and a satisfying (5.7). We note that the following gives a non-trivial representation of 1:

$$g^{s_0-\sum_{j=1}^n r_j\beta_j+\psi}\prod_{i=1}^n g_i^{s_i-\sum_{j=1}^n A_{ij}\beta_j+\psi_i}=1$$

If either $s_0 \neq \sum_{j=1}^n r_j\beta_j + \psi$ or $s_i \neq A_{ij}\beta_j + \psi_i$ for some i, then $\mathcal{P}^*$ is able to find a non-trivial representation of 1 with overwhelming probability if (5.7) is satisfied.

Next, we look at (5.8) that is the other equation verified by $\mathcal{V}$. Assume that $\mathcal{P}^*$ knows $(r_i), A_{ij}, \psi$ and (ψ_i) satisfying (5.3) and (5.9). If (5.8) is satisfied, then the following holds $(\mathrm{mod} q)$:

$$s_0+\sum_{j=1}^n(s_j^2-\beta_j^2)=\sum_{j=1}^n(\psi_j^2+\sigma\psi)+\sum_{i=1}^n(\sum_{j=1}^n 2\psi_jA_{ji}+\sigma r_i)\beta_i$$

and $(\mathrm{mod} q)$,

$$\begin{aligned}&\sigma(\sum_{j=1}^n(r_j\beta_j+\psi))+\sum_{i=1}^n(\sum_{j=1}^n(A_{ij}\beta_j+\psi_i)^2-\beta_i^2)\\&=\sum_{j=1}^n(\psi_j^2+\sigma\psi)+\sum_{i=1}^n(\sum_{j=1}^n 2\psi_jA_{ji}+\sigma r_i)\beta_i\end{aligned}$$

We can use the fact that $\sum_{i=1}^n \beta_j = \sum_{i=1}^n \sum_{j=1}^n \delta_{ij}\beta_i\beta_j$. This gives us $(\mathrm{mod} q)$:

$$\begin{aligned}0=&(\sum_{j=1}^n\psi_j^2+\sigma\psi)-(\sum_{j=1}^n\psi_j^2+\sigma\psi)+\sum_{i=1}^n(\sum_{j=1}^n 2\psi_jA_{ji}+\sigma r_i)\beta_i\\&-\sum_{i=1}^n(\sum_{j=1}^n 2\psi_jA_{ji}+\sigma r_i)\beta_i+\sum_{i=1}^n\sum_{j=1}^n(\sum_{h=1}^n A_{hi}A_{hj}-\delta_{ij})\beta_i\beta_j\end{aligned}$$

It is clear that if (5.1) does not hold, the probability that (5.8) holds is negligible. This gives us that if $\mathcal{V}$ accepts either (5.1) holds, or $\mathcal{P}^*$ is able to find a non-trivial representation of 1. This completes our proof. □

Remark that since $(g_1, ..., g_n)$ originally are chosen by those who encrypt the messages, we cannot assure that the prover does not know the relation among (g_i). Hence, we can not assure that $\mathcal{P}^*$ is not able to construct (a_i) and a and such that $g^a\prod_{i=1}^n g_i^{a_i}=1$. We

note that the protocol does not satisfy the ordinary soundness property.

Theorem 22. *Let $\mathcal{B}$ be an attacker that can distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ in Protocol I or by a simulator $\mathcal{S}$ for Protocol I. We then have an attacker $\mathcal{A}$ against DDH such that $Adv\mathcal{B} \leq Advd\mathcal{A}$.*

Proof. The proof is constructed as follows:

1. Construct a simulator $\mathcal{S}$ for Protocol I.
2. Construct an attacker $\mathcal{F}_1$ that can distinguish between uniform instances from E^2_{n+1} and R^2_{n+1}.
3. Describe how the attacker can act as a simulator $\mathcal{S}'$.
4. Prove that $\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ when $I \in E^2_{n+1}$ and that $\mathcal{S}'$ perfectly simulates $\mathcal{S}$ when $I \in R^2_{n+1}$.

Construction of $\mathcal{S}$: A simulator $\mathcal{S}$ for Protocol I is constructed in Figure 5.2. The simulator outputs $(w, g', (\hat{w}_i), \hat{w}, (\beta_i), s_0, (s_i))$.

Input: $p, q, g, n, (g_i), (g'_i)$ $(i = 1, ..., n)$

Simulator, $\mathcal{S}$

$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$

$w, \hat{w}_i \xleftarrow{r} \mathbb{G}_q, (i = 1, ..., n)$

Compute:

$g' \leftarrow g^{s_0} \prod_{j=1}^{n} g_j^{s_j} g_j'^{-\beta_j}$

$\hat{w} \leftarrow w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} \prod_{j=1}^{n} \hat{w}_j^{-\beta_j}$

Figure 5.2: Construction of a simulator for Protocol I.

Construction of $\mathcal{F}_1$: We will construct an attacker $\mathcal{F}_1$ that can distinguish uniform instances from E^2_{n+1} and R^2_{n+1} if $\mathcal{S}$ cannot simulate Protocol I. Assume $I = (x_1^{(1)}, x_1^{(2)}, ..., x_{n+1}^{(1)}, x_{n+1}^{(2)})$, was chosen uniformly from either E^2_{n+1} or R^2_{n+1}.

We let $g = x_1^{(1)}$. First, $\mathcal{F}_1$ generates (g_i) as the constants used in Proof 1, and a permutation matrix A_{ij}. He computes $(i = 1, ..., n)$:

$$g'_i = x_{i+1}^{(1)} \prod_{j=1}^{n} g_j^{A_{ji}}$$

The attacker will now act as a simulator $\mathcal{S}'$, based on the values and the permutation matrix he has obtained. $\mathcal{S}'$ is constructed in Figure 5.3.

Input: p, q, g, n, (g_i), (g'_i) $(i = 1, ..., n)$

Simulator, $\mathcal{S}'$

$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$

$w \leftarrow x_1^{(2)}$

Compute:

$g' \leftarrow g^{s_0} \prod_{j=1}^{n} g_j^{s_j} g_j'^{-\beta_j}$

$\psi_j \leftarrow s_j - \sum_{k=1}^{n} A_{jk}\beta_k \bmod q, (j = 1, ..., n)$

$\hat{w}_i \leftarrow x_{i+1}^{(2)} \prod_{j=1}^{n} g^{2\psi_j A_{ji}}, (i = 1, ..., n)$

$\hat{w} \leftarrow w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} \prod_{j=1}^{n} \hat{w}_j^{-\beta_j}$

Figure 5.3: Construction of simulator $\mathcal{S}'$.

$\mathcal{S}'$ outputs an accepted transcript $(w, g', (\hat{w}_i), \hat{w}, (\beta_i), s_0, (s_i))$. We will now prove that $\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ when $I \in E_{n+1}^2$ and that $\mathcal{S}'$ perfectly simulates $\mathcal{S}$ when $I \in R_{n+1}^2$.

$\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ when $I \in E_{n+1}^2$: We assume $I \in E_{n+1}^2$. $\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ if transcripts constructed by $\mathcal{S}'$ and $(\mathcal{P}, \mathcal{V})$ have the same distribution. The proof is similar to the proof of Theorem 4, but we will outline the main differences.

In protocol I $\mathcal{P}$ chooses $\sigma \xleftarrow{r} \mathbb{Z}_q$. We now let: $\sigma = log_{x_1^{(1)}} x_1^{(2)}$. Remark that this will tamper the protocol, because σ will still be a random value. $\mathcal{P}$ is given private input (r_i) defined as follows: $r_i = log_{x_1^{(1)}} x_{i+1}^{(1)}$.

This gives us that $(\mathcal{P}, \mathcal{V})$ chooses ψ, (ψ_i) and (β_i) at random from $\mathbb{Z}_q$, and $\mathcal{S}'$ chooses s_0, (s_i) and (β_i) at random from $\mathbb{Z}_q$. It is clear that $\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ when $I \in E_{n+1}^2$.

$\mathcal{S}'$ perfectly simulates $\mathcal{S}$ when $I \in R_{n+1}^2$: We assume $I \in R_{n+1}^2$. In this case $(x_1^{(2)}, ..., x_{n+1}^{(2)})$ is chosen at random from $\mathbb{G}_q$ by $\mathcal{S}'$. This gives us Table 5.1. It is clear that transcripts constructed by $\mathcal{S}$ and $\mathcal{S}'$ gives the same distribution, hence $\mathcal{S}'$ perfectly simulate $\mathcal{S}$ when $I \in R_{n+1}^2$.

Figure 5.4 illustrates what we have proven so far. Transitivity gives us that if we have an attacker $\mathcal{B}$ that is able to distinguish transcripts from $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}$, then we have an attacker $\mathcal{F}_1$ that is able do distinguish uniform instances from E_{n+1}^2 and R_{n+1}^2. From Theorem 19 we know that if it is easy for $\mathcal{F}_1$ to distinguish these instances, then it is easy

$\mathcal{S}$	$\mathcal{S}'$
$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q (i = 1, ..., n)$	$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
$w, \hat{w}_i \xleftarrow{r} \mathbb{G}_q (i = 1, ..., n)$	$x_i^{(2)} \xleftarrow{r} \mathbb{G}_q \ (i = 1, ..., n + 1)$

Table 5.1: Outline of the randomness in transcripts constructed by $\mathcal{S}$ and $\mathcal{S}'$.

for $\mathcal{A}$ to solve the DDH problem. Hence, if it is easy for $\mathcal{B}$ to distinguish the transcripts, it is easy for $\mathcal{A}$ to solve DDH. This completes the proof of the theorem. □

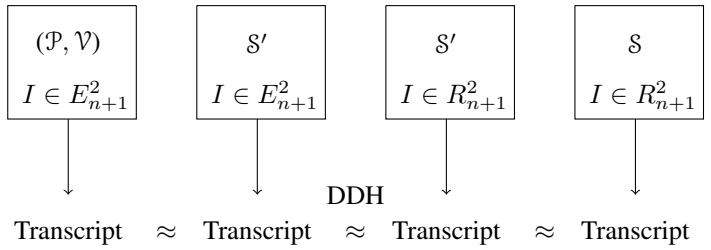

Figure 5.4: An illustration of how the proof is constructed.

We note that as long as it is difficult to solve DDH, $\mathcal{B}$ will not be able to distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ or by $\mathcal{S}$. The protocol is computationally HVZK.

5.3 Proof II

We will in this section first construct Protocol II, and then look at the security of the protocol. Sequences (g_i) and (g'_i) are made public, and a sequence (r_i) and A_{ij} is given as private input to $\mathcal{P}$. The protocol proves that A_{ij} satisfies the second condition for permutation matrices given in (5.2) and that (g'_i) are computed correctly according to (5.3).

Construction

Similarly to protocol I, we need to add randomizers to prevent that the protocol leaks information about A_{ij}. $\mathcal{P}$ draws randomizers $\tau, \theta, \psi, (\psi_i), \lambda, (\lambda_i)$ and calculate commitments. The commitments are sent to $\mathcal{V}$, and $\mathcal{V}$ gives a challenge (β_i) as a response. Finally $\mathcal{P}$

computes s_0, (s_i) and d, send this to $\mathcal{V}$, and $\mathcal{V}$ verifies if the following equations hold:

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = g' \prod_{j=1}^{n} {g'_j}^{\beta_j} \tag{5.12}$$

$$g^d = u \prod_{j=1}^{n} u_j^{\beta_j^2} \tag{5.13}$$

$$t^d v^{s_0} g^{\sum_{j=1}^{n}(s_j^3 - \beta_j^3)} = \hat{v} \prod_{j=1}^{n} \hat{v}_j^{\beta_j} \hat{t}_j^{\beta_j^2} \tag{5.14}$$

This gives us as is a three move protocol with output (α, β, γ), where $\alpha = (t, v, u, (u_i), g', (\hat{t}_i), (\hat{v}_i), \hat{v})$, $\beta = (\beta_i)$, $\gamma = (s_0, (s_i), d)$. The protocol is illustrated in Figure 5.5.

Public input: p, q, g, n, (g_i), (g'_i) $(i = 1, ..., n)$
Private input to $\mathcal{P}$: (r_i), A_{ij} $(i, j = 1, ..., n)$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\tau, \theta, \psi, \psi_i, \lambda, \lambda_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$		
Compute:		
$t \leftarrow g^\tau, v \leftarrow g^\theta, u \leftarrow g^\lambda$		
$u_i \leftarrow g^{\lambda_i}, (i = 1, ..., n)$		
$g' \leftarrow g^\psi \prod_{j=1}^{n} g_j^{\psi_j}$		
$\hat{t}_i \leftarrow g^{\sum_{j=1}^{n} 3\psi_j A_{ji} + \tau\lambda_i}, (i = 1, ..., n)$		
$\hat{v}_i \leftarrow g^{\sum_{j=1}^{n} 3\psi_j^2 A_{ji} + \theta r_i}, (i = 1, ..., n)$		
$\hat{v} \leftarrow g^{\sum_{j=1}^{n} \psi_j^3 + \tau\lambda + \theta\psi}$		
	$\xrightarrow{t, v, u, (u_i)_{i=1}^n, g', (\hat{t}_i)_{i=1}^n, (\hat{v}_i)_{i=1}^n, \hat{v}}$	
		$\beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
	$\xleftarrow{(\beta_i)_{i=1}^n}$	
Compute :		
$s_0 \leftarrow \sum_{j=1}^{n} r_j \beta_j + \psi \bmod q$		
$s_i \leftarrow \sum_{j=1}^{n} A_{ij} \beta_j + \psi_i \bmod q, (i = 1, ..., n)$		
$d \leftarrow \sum_{j=1}^{n} \lambda_j \beta_j^2 + \lambda \bmod q$		
	$\xrightarrow{s_0, (s_i)_{i=1}^n, d}$	
		Verify if $(5.12), (5.13)$ and (5.14) hold

Figure 5.5: Protocol II.

Security

Theorem 23. *Protocol II is complete.*

Completeness can be proved with simple calculations.

Theorem 24. *If $\mathcal{V}$ accepts Protocol II with non-negligible probability, then $\mathcal{P}^*$ either knows both (r_i) and A_{ij} satisfying (5.2), or can generate integers (a_i) and a satisfying $g^a \prod_{i=1}^n g_i^{a_i} = 1$ with overwhelming probability.*

Recall that (5.2) is given by: $\sum_{h=1}^n A_{hi}A_{hj}A_{hk} = 1$ if $i = j = k$ and 0 otherwise, (5.3) is given by: $g_i' = g^{r_i} \prod_{j=1}^n g_j^{A_{ji}}$ and (5.9) is given by: $g' = g^{\psi} \prod_{j=1}^n g_j^{\psi_j}$.

Proof. The theorem can be proved analogously to proof of Theorem 21. We can with rewinding prove that $\mathcal{P}^*$ knows (r_i), A_{ij}, ψ and (ψ_i) such that (5.3) and (5.9) are satisfied. This knowledge implies that if $\mathcal{V}$ accepts then either (5.2) holds or $\mathcal{P}^*$ is able to find a non-trivial representation of 1. For the last statement, we can perform the exact same argument as described in proof of Theorem 21. This shows that (5.12) will hold if $\mathcal{P}^*$ is able to find (a_i) and a such that $g^a \prod_{i=1}^n g_i^{a_i} = 1$.

For the first statement we see that if (5.14) holds, then the following is satisfied:

$$
\begin{aligned}
0 = {} & \tau(\sum_{i=1}^n \lambda_i \beta_j^2 + \lambda) - \tau(\sum_{i=1}^n \lambda_i \beta_j^2 + \lambda) + \theta(\sum_{i=1}^n r_i \beta_i + \psi) - \theta(\sum_{i=1}^n r_i \beta_i + \psi) \\
& + 3\left(\sum_{i=1}^n \sum_{j=1}^n (\psi_j^2 A_{ji} \beta_i + \psi_j A_{ji} \beta_i^2)\right) - 3\left(\sum_{i=1}^n \sum_{j=1}^n (\psi_j^2 A_{ji} \beta_i + \psi_j A_{ji} \beta_i^2)\right) \\
& + \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n (\sum_{h=1}^n A_{hi}A_{hj}A_{hk} - \delta_{ijk}) \beta_i \beta_j \beta_k
\end{aligned}
$$

where we have used the fact that $\sum_{j=1}^n \beta_j^3 = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \delta_{ijk} \beta_i \beta_j \beta_k$. It is clear that if (5.2) does not hold, then the probability that $\mathcal{V}$ accept is negligible. This completes the proof of the theorem. □

Remark that the protocol does not satisfy the ordinary soundness proopery, because we can not assure that $\mathcal{P}^*$ does not know the relation between (g_i).

Theorem 25. *Let $\mathcal{B}$ be an attacker that can distinguish if an accepted transcript (α, β, γ) was constructed by ($\mathcal{P}$, V) in Protocol II, or by a simulator $\mathcal{S}$ for Protocol II. We then have an attacker $\mathcal{A}$ against DDH such that $Adv\mathcal{B} \leq Adv\mathcal{A}$.*

The theorem can be proved analogously to Theorem 22. We can construct a simulator $\mathcal{S}$ for Protocol II, and construct an attacker $\mathcal{F}_2$ that can distinguish uniform instances from

E^3_{n+1} and R^3_{n+1}. $\mathcal{F}_2$ can then act as a simulator $\mathcal{S}'$. Theorem 18 gives us that if it is easy for $\mathcal{F}_2$ to distinguish these instances, then it is easy for an attacker $\mathcal{F}_1$ to distinguish uniform instances from E^2_{n+1} and R^2_{n+1}.

This is the exact same situation we had in proof of Theorem 22, and we can conclude that if it is difficult to solve DDH, then $\mathcal{B}$ will not be able to distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ or by $\mathcal{S}$. The protocol is computationally HVZK. We will not write the proof in detail here, but how $\mathcal{S}$ and $\mathcal{S}'$ compute their values, e.g. u, g' and $\hat{v}$ are included in proof of Theorem 30 in Section 5.6.

5.4 Protocol III

We want to prove that (r_i) and A_{ij} used in Protocol I and Protocol II are equal. We will argue that this will be satisfied if we include a new basis $(\tilde{g}, \tilde{g_1}, ..., \tilde{g}_n)$ in the Permutation matrix protocol, where $\tilde{g}$ and $(\tilde{g}_i)$ are chosen at random. This basis should be independent from the input ciphertexts. We can assure that the relation among the basis is unknown, in contrast to the relation among (g_i) (See proof of Theorem 21). Under the discrete logarithm assumption it will then be computationally infeasible to obtain (a_i) and a such that $\tilde{g}^a \prod_{j=1}^n \tilde{g}_j^{a_i} = 1$ [4].

In the Permutation matrix protocol constructed in Section 5.6 the prover computes $\tilde{g}'_i = \tilde{g}^{r_i} \prod_{j=1}^n \tilde{g}_j^{A_{ji}}$. He has to perform the same permutation on $(\tilde{g}, \tilde{g_1}, ..., \tilde{g}_n)$ and $(g, g_1, ..., g_n)$. A new verification equation $\tilde{g}^{s_0} \prod_{j=1}^n \tilde{g}_j^{s_j} = \tilde{g}' \prod_{j=1}^n \tilde{g}_j'^{\beta_j}$ is added. We can use the results from Protocol I and Protocol II:

- A_{ij} satisfies (5.1). We know from Protocol I that given A_{ij}, (r_i), $(\tilde{g}_i)$ and $(\tilde{g}'_i)$, $(\tilde{g}'_i)$ can be expressed as:
$$\tilde{g}'_i = g^{r_i} \prod_{j=1}^{n} \tilde{g}_j^{A_{ji}}$$
when $\mathcal{V}$ accepts.

- A_{ij} satisfies (5.2). We know from Protocol II that given A_{ij}, (r_i), $(\tilde{g}_i)$ and $(\tilde{g}'_i)$, $(\tilde{g}'_i)$ can be expressed as:
$$\tilde{g}'_i = g^{r_i} \prod_{j=1}^{n} \tilde{g}_j^{A_{ji}}$$
when $\mathcal{V}$ accepts.

Hence, A_{ij} is proved to be a permutation according to Theorem 17. If (r_i) and A_{ij} not are identical in the two representations of $(\tilde{g}_i)$ above, then the prover knows two different representations of an element in the fixed basis, which means that he knows the relation

among the elements. This goes against our assumption. It is therefore sufficient to add the new basis in the Permutation matrix protocol to assure that (r_i) and A_{ij} used in Protocols I and II are identical.

5.5 Protocol IV

This protocol proves that the same r_i and A_{ij} are used in computations of a re-encryption pair $(g_i',\ m_i')$. We will make use of a protocol that is used for proving knowledge of a representation, written by Brands [4]. Encryption pairs $(\{g_i, m_i)\}$ and re-encryption pairs $\{(g_i', m_i')\}$ are made public, in addition to the public key of the encryption scheme. A sequence (r_i) and A_{ij} are given as private input to $\mathcal{P}$.

Recall that according to (5.3) and (5.4) the re-encryption $C_i' = (g_i', m_i')$ of a ciphertext $C_i = (g_i, m_i)$ is computes as:

$$(g_i', m_i') = (g^{r_i} \prod_{j=1}^{n} g_j^{A_{ji}}, y^{r_i} \prod_{j=1}^{n} m_j^{A_{ji}})$$

Construction

The protocol is illustrated in Figure 5.6. First, $\mathcal{P}$ computes commitments g' and m', and sends $\alpha = (g', m')$ to $\mathcal{V}$. Second, $\mathcal{P}$ is given a random challenge (β_i). Further, $\mathcal{P}$ computes s_0 and (s_i) and sends $\gamma = (s_0, (s_i))$ to $\mathcal{V}$. Finally, $\mathcal{V}$ verifies if the verification equations hold:

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = g' \prod_{j=1}^{n} {g_j'}^{\beta_j} \tag{5.15}$$

$$y^{s_0} \prod_{j=1}^{n} m_j^{s_j} = m' \prod_{j=1}^{n} {m_j'}^{\beta_j} \tag{5.16}$$

This gives us a three move protocol with output (α, β, γ). For convenience, we enumerate the following equations:

$$g' = g^{\psi} \prod_{j=1}^{n} g_j^{\psi_j} \tag{5.17}$$

$$m' = y^{\psi} \prod_{j=1}^{n} m_j^{\psi_j} \tag{5.18}$$

Public input: p, q, g, n, y, $\{(g_i, m_i)\}$, $\{(g'_i, m'_i)\}$ $(i = 1, ..., n)$
Private input to $\mathcal{P}$: (r_i), A_{ij} $(i, j = 1, ..., n)$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\psi, \psi_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$		
Compute g' and m' according to (5.17) and (5.18)	$\xrightarrow{g', m'}$	
		$\beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
	$\xleftarrow{(\beta_i)_{i=1}^n}$	
$s_0 \leftarrow \sum_{j=1}^n r_j \beta_j + \psi, \mathrm{mod} q$ $s_i \leftarrow \sum_{j=1}^n A_{ij} \beta_j + \psi_i \bmod q, (i = 1, ..., n)$		
	$\xrightarrow{s_0, (s_i)_{i=1}^n}$	
		Verify if (5.15) and (5.16) hold

Figure 5.6: Protocol IV.

Security

Theorem 26. *Protocol IV is complete.*

The completeness of the protocol is easy to see with simple calculations.

Theorem 27. *If $\mathcal{V}$ accepts Protocol IV with non-negligible probability, then $\mathcal{P}^*$ either knows both (r_i) and A_{ij} such that the relation between (5.3), (5.4), (5.17) and (5.18) are satisfied, or can generate integers (a_i) and a satisfying $g^a \prod_{i=1}^n g_i^{a_i} = 1$ with overwhelming probability.*

Proof. We perform rewinding, equally to the rewinding performed in proof of Theorem 21. This proves that $\mathcal{P}^*$ knows (r_i), A_{ij}, ψ and (ψ_i) such that (5.3), (5.4), (5.17) and (5.18) are satisfied. We will prove that this knowledge implies that the same r_i and A_{ij} is used to construct a re-encryption pair (g'_i, m'_i) when $\mathcal{V}$ accepts.

If verification equation (5.15) holds, then:

$$g^{\sum_{j=1}^{n} r_j\beta_j+\psi}\prod_{i=1}^{n} g_i^{\sum_{j=1}^{n} A_{ij}\beta_j+\psi_i} = g'\prod_{i=1}^{n} {g'_j}^{\beta_j}$$
$$g^{\psi}\prod_{i=1}^{n} g_i^{\psi_i}\prod_{j=1}^{n}\left(g^{r_j\beta_j}\prod_{i=1}^{n} g_i^{A_{ij}\beta_j}\right) = g'\prod_{i=1}^{n} {g'_j}^{\beta_j}$$
$$\frac{g^{\psi}\prod_{i=1}^{n} g_i^{\psi_i}}{g'}\prod_{j=1}^{n}\left(\frac{g^{r_j}\prod_{i=1}^{n} g_i^{A_{ij}}}{g'_j}\right)^{\beta_j} = 1$$

The probability for this to hold if (5.3) and (5.17) does not hold is negligible. Similarly, if (5.16) holds, then:

$$\frac{y^{\psi}\prod_{i=1}^{n} m_i^{\psi_i}}{m'}\prod_{j=1}^{n}\left(\frac{y^{r_j}\prod_{i=1}^{n} m_i^{A_{ij}}}{m'_j}\right)^{\beta_j} = 1$$

The probability for this to hold if (5.4) and (5.18) does not hold is negligible. Hence, (g'_i, m'_i) are computed correctly.

For the second statement of the theorem, we use the exact same argument described in proof of Theorem 21, where we saw that $\mathcal{P}^*$ can forge a proof if he is able to generate non-trivial integers (a_i) and a such that $g^a\prod_{i=1}^{n} g_i^{a_i} = 1$.

This gives us that if $\mathcal{V}$ accepts the protocol, then either the same r_i and A_{ij} are used to compute a pair (g'_i, m'_i), or $\mathcal{P}^*$ is able to find a non-trivial representation of 1. □

We note that the protocol, similarly to Protocol I, does not satisfy the ordinary soundness property, because we can not assure that $\mathcal{P}^*$ does not know the relation between $(g_1, ..., g_n)$. Remark that in this protocol A_{ij} is not proved to be a permutation matrix.

5.6 The Permutation matrix protocol

We have now constructed Protocols I, II and IV, and explained how the conditions in Protocol III can be met. These proofs can be executed in parallel, resulting in a protocol that reduces communication complexity.

Encryption, and re-encryption pairs $\{(g_i, m_i)\}$ and $\{(g'_i, m'_i)\}$ are given as public input. In addition, the new basis $\tilde{g}$ and $(\tilde{g}_i)$, and the public key for the encryption scheme are public. A sequence (r_i) and a permutation matrix A_{ij} are given as private input to $\mathcal{P}$. The protocol proves that A_{ij} is a permutation matrix, and that the same r_i and A_{ij} is used to compute a re-encryption pair. We know from Section 5.1.2 that this is sufficient to prove

correctness of the shuffle.

Construction

The Permutation matrix protocol is a three move protocol illustrated in Figure 5.7. First, $\mathcal{P}$ draws random values, computes commitments, and sends the commitments to $\mathcal{V}$. Second, $\mathcal{V}$ then sends a challenge (β_i) to $\mathcal{P}$. Third, $\mathcal{P}$ computes s_0, (s_i) and d and sends this to $\mathcal{V}$. Finally, $\mathcal{V}$ verifies if the verification equations hold:

$$\tilde{g}^{s_0} \prod_{j=1}^{n} \tilde{g}_j^{s_j} = \tilde{g}' \prod_{j=1}^{n} \tilde{g}_j'^{\beta_j} \tag{5.19}$$

$$g^{s_0} \prod_{j=1}^{n} g_j^{s_j} = g' \prod_{j=1}^{n} g_j'^{\beta_j} \tag{5.20}$$

$$y^{s_0} \prod_{j=1}^{n} m_j^{s_j} = m' \prod_{j=1}^{n} m_j'^{\beta_j} \tag{5.21}$$

$$w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} = \hat{w} \prod_{j=1}^{n} \hat{w}_j^{\beta_j} \tag{5.22}$$

$$g^d = u \prod_{j=1}^{n} u_j^{\beta_j^2} \tag{5.23}$$

$$t^d v^{s_0} g^{\sum_{j=1}^{n}(s_j^3 - \beta_j^3)} = \hat{v} \prod_{j=1}^{n} \hat{v}_j^{\beta_j} \hat{t}_j^{\beta_j^2} \tag{5.24}$$

Security

For convenience we enumerate the following equations:

$$\tilde{g}_i' = \tilde{g}^{r_i} \prod_{j=1}^{n} \tilde{g}_j^{A_{ji}} \tag{5.25}$$

$$\tilde{g}' = \tilde{g}^{\psi} \prod_{j=1}^{n} \tilde{g}_j^{\psi_j} \tag{5.26}$$

Theorem 28. *The Permutation matrix protocol is complete.*

Completeness of the protocol can be proved with simple calculations.

Theorem 29. *If $\mathcal{V}$ accepts the Permutation matrix protocol with non-negligible probability, then $\mathcal{P}^*$ either knows both (r_i) and a permutation matrix A_{ij} satisfying both (5.3) and*

Public input: p, q, g, y, n, $\tilde{g}$, $(\tilde{g}_i)$ $\{(g_i, m_i)\}$, $\{(g'_i, m'_i)\}$ $(i = 1, ..., n)$
Private input to $\mathcal{P}$: (r_i), $A_{ij} (i, j = 1, ..., n)$

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$
$\sigma, \tau, \theta, \psi, \psi_i, \lambda, \lambda_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$		
Compute:		
$t \leftarrow g^\tau, v \leftarrow g^\theta, w \leftarrow g^\sigma, u \leftarrow g^\lambda$		
$u_i \leftarrow g^{\lambda_i}, (i = 1, ..., n)$		
$\tilde{g}'_i \leftarrow \tilde{g}^{r_i} \prod_{j=1}^n \tilde{g}_j^{A_{ji}}, (i = 1, ..., n)$		
$\tilde{g}' \leftarrow \tilde{g}^\psi \prod_{j=1}^n \tilde{g}_j^{\psi_j}$		
$g' \leftarrow g^\psi \prod_{j=1}^n g_j^{\psi_j}$		
$m' \leftarrow y^\psi \prod_{j=1}^n m_j^{\psi_j}$		
$\hat{t}_i \leftarrow g^{\sum_{j=1}^n 3\psi_j A_{ji} + \tau\lambda_i}, (i = 1, ..., n)$		
$\hat{v}_i \leftarrow g^{\sum_{j=1}^n 3\psi_j^2 A_{ji} + \theta r_i}, (i = 1, ..., n)$		
$\hat{v} \leftarrow g^{\sum_{j=1}^n \psi_j^3 + \tau\lambda + \theta\psi}$		
$\hat{w}_i \leftarrow g^{\sum_{j=1}^n 2\psi_j A_{ji} + \sigma r_i}, (i = 1, ..., n)$		
$\hat{w} \leftarrow g^{\sum_{j=1}^n \psi_j^2 + \sigma\psi}$		
	$\xrightarrow{t, v, w, u, (u_i)_{i=1}^n, (\tilde{g}'_i)_{i=1}^n, \tilde{g}'}$	
	$\xrightarrow{g', m', (\hat{t}_i)_{i=1}^n, (\hat{v}_i)_{i=1}^n, \hat{v}, (\hat{w}_i)_{i=1}^n, \hat{w}}$	
		$\beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
	$\xleftarrow{(\beta_i)_{i=1}^n}$	
Compute:		
$s_0 \leftarrow \sum_{j=1}^n r_j \beta_j + \psi \bmod q$		
$s_i \leftarrow \sum_{j=1}^n A_{ij} \beta_j + \psi_i \bmod q, (i = 1, ..., n)$		
$d \leftarrow \sum_{j=1}^n \lambda_j \beta_j^2 + \lambda \bmod q$		
	$\xrightarrow{s_0, (s_i)_{i=1}^n, d}$	
		Verify if: (5.19), (5.20), (5.21) (5.22), (5.23), (5.24) hold

Figure 5.7: The Permutation matrix protocol.

(5.4), or can generate integers (a_i) *and* a *satisfying* $\tilde{g}^a \prod_{i=1}^{n} \tilde{g}_i^{a_i} = 1$ *with overwhelming probability.*

Recall if (5.3) and (5.4) are satisfied then $(g'_i, m'_i) = (g^{r_i} \prod_{j=1}^{n} g_j^{A_{ji}}, y^{r_i} \prod_{j=1}^{n} m_j^{A_{ji}})$.

Proof. This theorem can easily be proved by combining the results from Theorems 21, 24 and 26. With rewinding we can prove that $\mathcal{P}^*$ knows $(r_i), A_{ij}, \psi$ and (ψ_i) that satisfy (5.25) and (5.26) when (5.19) is satisfied. This knowledge implies that A_{ij} satisfies (5.1) and (5.2), if verification equations (5.19), (5.22), (5.23) and (5.24) hold. Hence, A_{ij} is proved to be a permutation matrix according to Definition 6.

We can use the same argument described in proof of Theorem 26 to prove that when (5.20) and (5.21) are satisfied the same (r_i) and A_{ij} have been used to compute a re-encryption pair (g'_i, m'_i) . This yields the correctness of the shuffle.

For the last statement of the theorem, we use the same approach as earlier, and prove that it is a possibility for $\mathcal{P}^*$ to fore a proof. Verification equation (5.19) is satisfied if the following holds:

$$\tilde{g}^{s_0 - \sum_{j=1}^{n} r_j \beta_j + \psi} \prod_{j=1}^{n} \tilde{g}_j^{s_j - \sum_{j=1}^{n} A_{ij} \beta_j + \psi_i} = 1$$

If $s_0 \neq \sum_{j=1}^{n} r_j \beta_j + \psi$ or $s_i \neq A_{ij} \beta_j + \psi_i$ for some i, then $\mathcal{P}^*$ is able to find (a_i) and a such that:

$$\tilde{g}^a \prod_{i=1}^{n} \tilde{g}_i^{a_i} = 1$$

Hence, if $\mathcal{V}$ accepts, then $\mathcal{P}^*$ knows both (r_i) and a permutation matrix A_{ij} and the re-encryption is performed correctly according to (5.3) and (5.4), or he is able to find a non-trivial representation of 1. □

As written in proof in Section 5.4, if $(\tilde{g}_i)$ is chosen at random, then we can make it computationally infeasible for $\mathcal{P}^*$ to obtain such (a_i) and a under the discrete logarithm assumption. Hence, the Permutation matrix protocol is a proof of knowledge if is difficult to compute discrete logarithms. This gives us that the protocol is computationally sound.

Theorem 30. *Let* $\mathcal{B}$ *be an attacker that can distinguish if an accepted transcript* (α, β, γ) *was constructed by* $(\mathcal{P}, \mathcal{V})$ *in the Permutation matrix protocol or by a simulator* $\mathcal{S}$ *for the Permutation matrix protocol. We then have an attacker* $\mathcal{A}$ *against DDH such that* $Adv\mathcal{B} \leq Adv\mathcal{A}$.

Proof. We can construct a proof analogously to the proof of Theorem 22. We will construct a simulator $\mathcal{S}$ of the Permutation matrix protocol, and then construct an attacker $\mathcal{F}_2$

that can distinguish uniform instances from E^5_{n+1} and R^5_{n+1}. We will explain how $\mathcal{F}_2$ can act as a simulator $\mathcal{S}'$. Theorem 18 gives us that if it is easy for $\mathcal{F}_2$ to distinguish these instances, then it is easy for an attacker $\mathcal{F}_1$ to distinguish uniform instances from E^2_{n+1} and R^2_{n+1}. This is the exact same situation we had in proof of Theorem 22, and we can conclude that if it is difficult to solve DDH, then $\mathcal{B}$ will not be able to distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ or by $\mathcal{S}$. For completeness sake, we will include the full proof of the theorem:

We first construct a simulator $\mathcal{S}$ for the Permutation matrix protocol, then we construct an attacker $\mathcal{F}_2$, and explain how $\mathcal{F}_2$ can act as a simulator $\mathcal{S}'$.

Construction of $\mathcal{S}$: The simulator is constructed in Figure 5.8. The output is given as: $(t, v, w, u, (u_i), (\tilde{g}'_i), \tilde{g}', g', m', (\hat{t}_i), (\hat{v}_i), \hat{v}, (\hat{w}_i), \hat{w}, (\beta_i), s_0, (s_i), d)$.

Input: $p, q, g, y, n, \tilde{g}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\}$ $(i = 1, ..., n)$

Simulator, $\mathcal{S}$

$s_0, s_i, \beta_i, d, \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$

$t, v, w, u_i, \hat{t}_i, \hat{v}_i, \hat{w}_i, \tilde{g}'_i \xleftarrow{r} \mathbb{G}_q, (i = 1, ..., n)$

Compute:

$u \leftarrow g^d \prod_{j=1}^{n} u_j^{-\beta_j^2}$

$\tilde{g}' \leftarrow \tilde{g}^{s_0} \prod_{j=1}^{n} \tilde{g}_j^{s_j} \tilde{g}_j'^{-\beta_j}$

$g' \leftarrow g^{s_0} \prod_{j=1}^{n} g_j^{s_j} g_j'^{-\beta_j}$

$m' \leftarrow y^{s_0} \prod_{j=1}^{n} m_j^{s_j} m_j'^{-\beta_j}$

$\hat{v} \leftarrow t^d v^{s_0} g^{\sum_{j=1}^{n}(s_j^3 - \beta_j^3)} \prod_{j=1}^{n} \hat{t}_j^{-\beta_j^2} \hat{v}_j^{-\beta_j}$

$\hat{w} \leftarrow w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} \prod_{j=1}^{n} \hat{w}_j^{-\beta_j}$

Figure 5.8: Construction of a simulator $\mathcal{S}$ for the Permutation matrix protocol.

Construction of $\mathcal{F}_2$: We construct an attacker $\mathcal{F}_2$ that can distinguish between uniform instances from E^5_{n+1} and R^5_{n+1} if $\mathcal{S}$ cannot simulate the Permutation matrix protocol. Assume that $I = (x_1^{(1)}, x_1^{(2)}, ..., x_1^{(5)}, ..., x_{n+1}^{(1)}, x_{n+1}^{(2)}, ..., x_{n+1}^{(5)})$, was chosen uniformly from either $E^{(5)}_{n+1}$ or $R^{(5)}_{n+1}$.

We let $g = x_1^{(1)}$, $\tilde{g} = x_1^{(2)}$, $y = g^X$, $X \xleftarrow{r} \mathbb{Z}_q$. $\mathcal{F}_2$ generates $\{(g_i, m_i)\}$ and $\{(\tilde{g}_i, \tilde{m}_i)\}$ as the constants used in Permutation matrix protocol. He generates a random permutation

matrix A_{ij}, and computes:

$$(g'_i, m'_i) = (x_{i+1}^{(1)} \prod_{j=1}^{n} g_j^{A_{ji}}, (x_{i+1}^{(1)})^X \prod_{j=1}^{n} m_j^{A_{ji}}), (i = 1, ..., n)$$

$\mathcal{F}_2$ will now act as a simulator $\mathcal{S}'$, constructed in Figure 5.9. The simulator outputs $(t, v, w, u, (u_i), (\tilde{g}'_i), \tilde{g}', g', m', (\hat{t}_i), (\hat{v}_i), \hat{v}, (\hat{w}_i), \hat{w}, (\beta_i), s_0, (s_i), d)$.

Input: $p, q, g, y, n, \tilde{g}, \{(g_i, m_i)\}, \{(g'_i, m'_i)\}$ $(i = 1, ..., n)$

Simulator, $\mathcal{S}'$

$s_0, s_i, \beta_i, d, h_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
Let : $t = x_1^{(3)}, v = x_1^{(4)}, w = x_1^{(5)}$
Compute:
$u_i \leftarrow (x_{i+1}^{(1)})^{h_i} \bmod p, i = 1, ..., n$
$u \leftarrow g^d \prod_{j=1}^{n} u_j^{-\beta_j^2}$
$\tilde{g}'_i \leftarrow x_{i+1}^{(2)} \prod_{j=1}^{n} \tilde{g}_j^{A_{ji}}, (i = 1, ..., n)$
$\tilde{g}' \leftarrow \tilde{g}^{s_0} \prod_{j=1}^{n} \tilde{g}_j^{s_j} \tilde{g}_j'^{-\beta_j}$
$g' \leftarrow g^{s_0} \prod_{j=1}^{n} g_j^{s_j} g_j'^{-\beta_j}$
$m' \leftarrow y^{s_0} \prod_{j=1}^{n} m_j^{s_j} m_j'^{-\beta_j}$
$\psi_j \leftarrow s_j - \sum_{k=1}^{n} A_{jk}\beta_k \bmod q, (j = 1, ..., n)$
$\hat{t}_i \leftarrow (x_{i+1}^{(3)})^{h_i} \prod_{j=1}^{n} g^{3\psi_j A_{ji}}, (i = 1, ..., n)$
$\hat{v}_i \leftarrow x_{i+1}^{(4)} \prod_{j=1}^{n} g^{3\psi_j^2 A_{ji}}, (i = 1, ..., n)$
$\hat{w}_i \leftarrow x_{i+1}^{(5)} \prod_{j=1}^{n} g^{2\psi_j A_{ji}}, (i = 1, ..., n)$
$\hat{v} \leftarrow t^d v^{s_0} g^{\sum_{j=1}^{n}(s_j^3 - \beta_j^3)} \prod_{j=1}^{n} \hat{t}_j^{-\beta_j^2} \hat{v}_j^{-\beta_j}$
$\hat{w} \leftarrow w^{s_0} g^{\sum_{j=1}^{n}(s_j^2 - \beta_j^2)} \prod_{j=1}^{n} \hat{w}_j^{-\beta_j}$

Figure 5.9: Construction of $\mathcal{S}'$.

$\mathcal{S}'$ *perfectly simulates* $(\mathcal{P}, \mathcal{V})$ *when* $I \in E_{n+1}^5$: In the Permutation matrix protocol (r_i), is given as private input to $\mathcal{P}$. We now define r_i as follows:

$$r_i = log_{x_1^{(1)}} x_{i+1}^{(1)},$$

and σ, τ, θ, $\lambda_i (i = 1, ..., n)$ as:

$$\begin{aligned}
\lambda_i &= (log_{x_1^{(1)}} x_{i+1}^{(1)})^{h_i} \\
\tau &= log_{x_1^{(1)}} x_1^{(3)} \\
\theta &= log_{x_1^{(1)}} x_1^{(4)} \\
\sigma &= log_{x_1^{(1)}} x_1^{(5)}
\end{aligned}$$

This gives the following values used by $\mathcal{S}'(i = 1, ..., n)$:

$$\begin{aligned}
x_{i+1}^{(1)} &= g^{r_i} \\
(x_{i+1}^{(1)})^{h_i} &= g^{\lambda_i} \\
x_{i+1}^{(2)} &= \tilde{g}^{r_i} \\
(x_{i+1}^{(3)})^{h_i} &= g^{\tau\lambda_i} \\
x_{i+1}^{(4)} &= g^{\theta r_i} \\
x_{i+1}^{(5)} &= g^{\sigma r_i}
\end{aligned}$$

We summarize what is chosen at random by $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}'$ in Table 5.2. We stress that (λ_i) are random values, because (h_i) is chosen at random from $\mathbb{Z}_q$. It is clear that $\mathcal{S}'$ perfectly simulates $(\mathcal{P}, \mathcal{V})$ when $I \in E_{n+1}^5$, because their constructed transcripts give the same distribution.

$(\mathcal{P}, \mathcal{V})$	$\mathcal{S}'$
$\psi, \psi_i, \lambda, \lambda_i, \beta_i \xleftarrow{r} \mathbb{Z}_q (i = 1, ..., n)$	$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$ $d, h_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$

Table 5.2: Outline of the randomness in transcripts constructed by $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}'$.

$\mathcal{S}'$ perfectly simulates $\mathcal{S}$ when $I \in R_{n+1}^5$: In this case, $\mathcal{S}'$ randomly chooses $(x_i^{(2)}, x_i^{(3)}, x_i^{(4)}, x_i^{(5)})$ $(i = 1, ..., n + 1)$ from $\mathbb{G}_q$. Recall that $\tilde{g} = x_1^{(2)}$. Hence, $\tilde{g}$ is a random number. We summarize what is chosen at random by $\mathcal{S}$ and $\mathcal{S}'$ in Table 5.3. It is easy to see that transcripts constructed by $\mathcal{S}$ and $\mathcal{S}'$ have the same distribution, hence $\mathcal{S}'$ perfectly simulates $\mathcal{S}$ when $I \in R_{n+1}^5$.

Transitivity gives us that if we have an attacker $\mathcal{B}$ that is able to distinguish transcripts from $(\mathcal{P}, \mathcal{V})$ and $\mathcal{S}$, then we have a distinguisher $\mathcal{F}_2$ that is able do distinguish uniform instances from E_{n+1}^5 and R_{n+1}^5. Theorems 18 and 19 impliy that if it is easy for $\mathcal{B}$ to distinguish the transcripts, then $\mathcal{A}$ can easily solve the DDH problem. □

$\mathcal{S}$	$\mathcal{S}'$
$s_0, s_i, \beta_i, d \xleftarrow{r} \mathbb{Z}_q (i = 1, ..., n)$	$s_0, s_i, \beta_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
$\tilde{g}, t, v, w, u_i, \hat{t}_i \xleftarrow{r} \mathbb{G}_q \ (i = 1, ..., n)$	$d, h_i \xleftarrow{r} \mathbb{Z}_q, (i = 1, ..., n)$
$\hat{v}_i, \hat{w}_i, \tilde{g}'_i \xleftarrow{r} \mathbb{G}_q (i = 1, ..., n)$	$x_i^{(2)}, x_i^{(3)}, x_i^{(4)}, x_i^{(5)} \xleftarrow{r} \mathbb{G}_q \ (i = 1, ..., n + 1)$

Table 5.3: Outline of the randomness in transcripts constructed by $\mathcal{S}$ and $\mathcal{S}'$.

Hence, $\mathcal{B}$ will not be able to distinguish if an accepted transcript (α, β, γ) was constructed by $(\mathcal{P}, \mathcal{V})$ or by $\mathcal{S}$ if it is difficult to solve DDH. The protocol is computationally HVZK.

Theorem 31. *The number of exponentiation required in the Permutation matrix protocol is* $(18n + 18)$.

Proof. The number of exponentiations required by the prover and verifier is summarized in Table 5.4. This gives us a total of $(18n + 18)$ exponentaitons to run the protocol.

Prover, $\mathcal{P}$		Verifier, $\mathcal{V}$	
Computation	Exponentiations	Computation	Exponentiations
t	1	(5.19)	$2n + 1$
v	1	(5.20)	$2n + 1$
w	1	(5.21)	$2n + 1$
u	1	(5.22)	$n + 2$
(u_i)	n	(5.23)	$n + 1$
$\tilde{g}'_i$	n	(5.24)	$2n + 3$
$\tilde{g}'$	$n + 1$		
g'	$n + 1$		
m'	$n + 1$		
$(\hat{t}_i)$	n		
$(\hat{v}_i)$	n		
$\hat{v}$	1		
$(\hat{w}_i)$	n		
$\hat{w}$	1		
Total: $8n + 9$		Total: $10n + 9$	

Table 5.4: Exponentiations required in the Permutation matrix protocol.

□

Chapter 6

Closing remarks

We have in this thesis discussed three protocols for verifiable shuffling a sequence of encrypted elements. This means that the mixnodes proves that they operate correctly according to the protocol. The Naive protocol, and Neff's Simple n-shuffle make use of the fact that polynomials are identical under permutation of their roots. The Permutation matrix protocol of Furukawa and Sako belongs to the other domain of verifiable shuffling, and make use of a permutation matrix. We note that a mixnet that exists of verifiable shuffle protocols will achieve unlinkability, because we can assure that at least one mixnode in the network operates correctly.

We have also examined the cMix protocol, which is an offline-online approach of mixnets. The authors of the protocol claim that cMix achieves unlinkability, but a formal security proof of the protocol is not written. Hence, we do not know whether it is secure or not. It is worth mentioning that including RPC in the cMix protocol will prevent both the insider- and tagging attacks described in Section 3.2. RPC can somehow be compared with verifiable shuffling because this method is used to verify if the mixnodes follow the protocol correctly. Remark that cMix will not achieve the same security as the verifiable shuffling protocols if RPC is implemented, because the mixnodes is not *proved* to operate correctly.

In Table 6.1 we summarize the security achieved in the protocols, and number of exponentiations required for each mixnode in the network.

By comparing the results represented in Table 6.1, cMix requires the least amount of work. This is not surprising, because the protocols do not provide verifiable shuffling. We stress that all the exponentiations required in cMix are done in an offline phase. This improves the protocols latency, which means that the total time spent form the network

Protocol	Exponentiations required	Security	Verifiable shuffling?
cMix	$5n$	Nothing proved	No
The Naive protocol	$13n - 12$	Sound Computationally HVZK	Yes
The Simple n-shuffle	$6n + 2$	Sound Perfect HVZK	Yes
The Permutation matrix protocol	$18n + 18$	Computationally sound Computationally HVZK	Yes

Table 6.1: Comparison of shuffles.

receives n messages and the protocol starts running, to the receives gets their messages, will be low.

We see that the Simple n-shuffle satisfies both soundness and perfect HVZK, and require the least amound of exponentiations of the verifiable shuffle protocols. The Permutation matrix protocol is only sound and HVZK if the discrete logarithm assumption and DDH assumption hold respectively.

Finally, we want to mention that both the Simple n-shuffle and the Permutation matrix protocol have been optimized and improved. The simple n-shuffle was first improved by Neff to a *General n-shuffle* [24], and later by Bayer and Groth, Groth and Groth and Ishai [2, 18, 19]. The shuffle by Furukawa and Sako has been improved by Furukawa [12], and Wikström showed how to split a shuffle that make use of a permutation matrix in an offline and online phase [30]. As we have discussed, this will reduce the protocols latency.

Bibliography

[1] Ben Adida and Douglas Wikström. Offline/online mixing. In *International Colloquium on Automata, Languages, and Programming*, pages 484–495. Springer, 2007.

[2] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 263–280. Springer, 2012.

[3] Josh Benaloh. Simple verifiable elections. In *EVT '06, Proceedings of the first Usenix/ACCURATE Electronic voting technology workshop*, pages 5–5, 2006.

[4] Stefan A Brands. An efficient off-line electronic cash system based on the representation problem. Technical report CSR9323, CWI, 1993.

[5] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T Sherman. cMix: mixing with minimal real-time asymmetric cryptographic operations. In *International Conference on Applied Cryptography and Network Security*, pages 557–578. Springer, 2017.

[6] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, Alan T Sherman, and D Das. cMix: Anonymization by high-performance scalable mixing. *25th USENIX Security Sym-posium*, 2016.

[7] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[8] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Inter-*

national Conference on the Theory and Applications of Cryptographic Techniques, pages 311–326. Springer, 1999.

[9] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *School organized by the European Educational Forum*, pages 63–86. Springer, 1998.

[10] Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[11] Ivan Damgård and Jesper Buus Nielsen. Commitment schemes and zero-knowledge protocols, 2006.

[12] Jun Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 88(1):172–188, 2005.

[13] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Annual International Cryptology Conference*, pages 368–387. Springer, 2001.

[14] Herman Galteland, Stig F Mjølsnes, and Ruxandra F Olimid. Attacks on the basic cMix design: On the necessity of commitments and randomized partial checking. In *International Conference on Cryptology in Malaysia*, pages 463–473. Springer, 2016.

[15] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[16] Jens Groth. A verifiable secret shuffe of homomorphic encryptions. In *International Workshop on Public Key Cryptography*, pages 145–160. Springer, 2003.

[17] Jens Groth. *Honest verifier zero-knowledge arguments applied*. Dissertation Series DS-04-3, BRICS, 2004.

[18] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.

[19] Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–396. Springer, 2008.

[20] Carmit Hazay and Yehuda Lindell. A note on zero-knowledge proofs of knowledge and the zkpok ideal functionality. *IACR Cryptology ePrint Archive*, 2010:552, 2010.

[21] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX security symposium*, pages 339–353. San Francisco, USA, 2002.

[22] Helger Lipmaa and Bingsheng Zhang. A more efficient computationally sound non-interactive zero-knowledge shuffle argument. *Journal of Computer Security*, 21(5):685–719, 2013.

[23] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In *International Conference on Cryptology in Africa*, pages 272–286. Springer, 2009.

[24] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[25] C Andrew Neff. Verifiable mixing (shuffling) of elgamal pairs. Technical report, In proceedings of PET 03, LNCS series, 2003.

[26] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 248–259. Springer, 1993.

[27] Andreas Pfitzmann and Michael Waidner. Networks without user observability—design options. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 245–253. Springer, 1985.

[28] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 415–431. Springer, 1999.

[29] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–266. Springer, 1997.

[30] Douglas Wikström. A commitment-consistent proof of a shuffle. In *Australasian Conference on Information Security and Privacy*, pages 407–421. Springer, 2009.