NTNU – Trondheim
Norwegian University of
Science and Technology

Verifiable Shuffled Decryption

Martin Strand

Master of Science in Mathematics
Submission date: June 2013
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

We describe the *Verifiable Shuffled Decryption* problem, and present five solutions based on adapting several existing verifiable shuffles. All but one may have potential for implementation, the choice of which would depend on the required level of security and computational restrictions given by the available hardware.

Sammendrag

Vi beskriver problemet *verifiserbar stokket dekryptering* og demonstrer til sammen fem løsninger, ved åtilpasse eksisterende beviser for korrekt stokking. Alle unntatt den første har en praktisk gjennomførbar beregningskostnad, og valget mellom vil avhenge av ønsket sikkerhetsnivå og hvilken begrensninger som blir satt av den tilgjengelige maskinvaren.

List of Figures

Contents

1 Introduction

This chapter introduces the problem, why we would like to solve it, and gives a brief overview of our contribution.

1.1 The Problem and Motivation

The theory of *mixnets* is well developed, and is based on a setup of several computers that take a set of ciphertexts as input, and output a shuffled list of new ciphertexts, having the same plaintexts as the input. For example, with an ElGamal scheme with k ciphertexts $\{(g^{r_i}, h^{r_i} m_i)\}$, the output would be

$$\{(g^{r_{\pi(i)}} g^{t_i}, h^{r_{\pi(i)}} h^{t_i} m_{\pi(i)})\}$$

where π is a permutation and t_i are random numbers. The randomisers makes it hard to extract the permutation. When composing several such computers, a single honest player will remove the correlation between the original input and the final output.

A severe danger of this setup is that one of the shufflers might be an active cheater, for example by discarding some of the ciphertexts and injecting new ones. This is commonly countered by requiring the shuffler to prove that everything has been done correctly, called *verifiable shuffle*.

This has a variation, *shuffle-decryption*, where each player has a part of the decryption key. As long as sufficiently many players are honest, the decryption will be successful, and there will be no correlation between ciphertexts and plaintexts. Furukawa [11] has demonstrated an efficient protocol for this, although not *zero knowledge*.

In this work, the setting is reduced from a mixnet to a single computer, which is supposed to both shuffle and decrypt verifiably. It can of course be done in two separate operations, first a shuffle and a proof of its validity, and then a decryption with a separate proof. However, this may require too much computation to be practical, and feels unsatisfactory. We would like to find an algorithm that draws advantage of the fact that we have full knowledge of some of the data. Hopefully, this will reduce the number of exponentiations needed to compute the proof, and hence improve the runtime.

In 2001, Neff [27] made a shuffling protocol for the special case where all exponents were known, and used it as a building block to produce a more general shuffle. Even though we know the decryption key, the original randomisation exponents will be hard to find. In the choice between specialising a more general protocol and to generalising a simple one, we choose the former.

Problem 1 (Verifiable Shuffled Decryption). Let $\mathcal{P}$ be the prover and $\mathcal{V}$ an honest verifier. The public input is lists $\{(x_i, y_i)\} = \{(g^{r_i} m_i)\}$ and $\{z_i\}$. The private input to $\mathcal{P}$ is a such that $h = g^a$ and π such that $z_i = m_{\pi(i)}$. $\mathcal{P}$ is required to give a zero knowledge proof of knowledge of π.

In addition, the solution should require fewer exponentiations than separate shuffling and decryption.

Let p and q be primes, and assume that $q|p-1$. The cryptography will take place in an order q subgroup G of $\mathbb{Z}_p$. Let g be a generator for G. The number of ciphertexts will be k.

A main application of this is electronic voting. In the setting of the Norwegian system [13], a ballot box will collect all votes along with a name tag, so that a voter can change her mind. Both the auditor and the ballot box sees these name tags, and could be able to remember the relation between encrypted votes and identities. After the election, the name tags are stripped, and the list of ciphertexts is submitted to the tallier, which in turn outputs the individual votes.

In order to avoid the ballot box and the auditor from being able to relate votes and identities, the tallier should permute the decrypted votes before outputting. However, the auditor needs to check that the tallier have decrypted correctly. It's therefore imperative that the proof of correct decryption is zero knowledge. If we are able to prove that a set of votes is a permutation of a decryption of a set of ciphertext, then we have also proved that the decryption must have been done correctly.

We have aimed for unconditional soundness when possible. First of all, it makes it easier to communicate the security to the general public, e.g. "it can't be done" vs. "we believe nobody are able to do it". Also, it may allow somewhat looser bounds in other parts of a system. However, it may result in weaker performance, since fewer computational techniques typically will be available. We have not attempted to achieve stronger privacy than computational zero knowledge, since the weakest link nonetheless will be the ElGamal encryption.

1.2 Our contribution

We present five ways to solve the above problem. The first is mostly an illustration that it can be done by applying the original idea while achieving unconditional

soundness. The next protocol is a direct adaptation of a shuffle-decryption for mixnets, and is modified to achieve unconditional soundness.

Next we have two variants of the same idea, with computational and unconditionally soundness respectively, the latter suffering from somewhat worse efficiency. The final successful protocol only achieves computational soundness, but may have good efficiency, depending on the implementation of the underlying primitive.

Proofs of all properties are included, although some of the protocols rely heavily on other primitives. In those cases, we use the results from the original articles without proof.

1.3 Outline of the thesis

The thesis consists of three main parts. In the first, we study the general topic. Next, we study and adapt specific protocols. Finally, we discuss our results.

Chapter 2 This is an introduction to some of the notions and small results that will be needed later on, the computational model, zero knowledge proofs and arguments, and variants thereof. We also prove the forking lemma.

Chapter 3 This chapter discusses the general idea of proving a shuffle using roots of polynomials, and adapts the work of Neff and Groth in order to solve the problem in four ways.

Chapter 4 Here we present shuffles using permutation matrices, and discuss some of the work of Furukawa, Peng-Dawson-Bao and Terelius-Wikström. We note that Furukawa's shuffle-decryption is readily available, and succeed in using a vital tool from [33] to create a working protocol.

Chapter 5 This chapter summarises and compares the results from the previous chapters, and gives an informal discussion of possible limitations.

1.4 Acknowledgements

I am very grateful to my supervisor Kristian Gjøsteen for his excellent guidance and mild encouragement. Thank you!

Next, my fellow students. By doing such an incredible job with your projects, you have made me put in a little more effort myself – apart from during the card breaks, naturally[1].

[1] There has been a lot of shuffling involved, at least.

For my family, thanks for making me able to even take on such a work, in particular thanks to Håvard for helping out with the proofreading. Finally, I should really thank Ragnhild for the support and for being so patient with me.

2 Theory

This chapter gives an introduction to the background necessary and relevant for the problem. Much of the theory on commitments, indistinguishability and zero knowledge is based on Damgård and Nielsens instructive text [9].

We assume that the reader is familiar with algorithms. Whenever we use the word "player", it will mean a computer performing an algorithm and possibly submitting data to another computer. The computational model is described more formally later.

2.1 Indistinguishability

Most players and algorithms in this work will be probabilistic, so the output will be drawn from a probability distribution. Much of the security analysis is based on comparisons of distributions, and so we will need precise notions for such comparisons.

Let μ_1 and μ_2 be discrete probability distributions over the same set S, and denote the distributions with X and Y. We then define the *statistical distance* as

$$\mathrm{d}(X,Y) = \frac{1}{2}\sum_{s\in S}\left|\mu_1(s) - \mu_2(s)\right|.$$

It is clear that d is a metric.

Algorithms and players will typically be challenged to distinguish two distributions. That makes it sensible to have notions of *success* probability in distinguishing the distribution X_0 from X_1. An algorithm recieves input x selected at random from X_b where $b \in \{0,1\}$, and must output the correct b.

$$\mathrm{Succ}_P(\mathcal{A}) = \Pr[\mathcal{A}(x) = b \mid b \xleftarrow{r} \{0,1\}, x \leftarrow X_b]$$

A player with a low success probability can easily be converted to one with high success probability by simply switching the reply. Therefore, we want to know how much a player's success rate deviates from simply flipping a coin, the *advantage*.

$$\mathrm{Adv}_P(\mathcal{A}) = \left|\mathrm{Succ}_P(\mathcal{A}) - \frac{1}{2}\right|$$

$\mathcal{A}$ is subject to a time and resource bound t. If $\mathrm{Adv}_P(\mathcal{A})$ is greater than some small ϵ, then we say that $\mathcal{A}$ is an ϵ-*distinguisher* for the problem P.

We say that two distributions are indistinguishable if it is hard to distinguish them. We have three levels of indistingishability, describing how hard the problem is. Let U and V be distributions.

Perfectly indistinguishable, $U \sim^p V$**.** U and V is the same probability space.

Statistically ϵ**-indistinguishable,** $U \sim^s V$**.** $\mathrm{d}(U, V) < \epsilon$.

Computationally (t, ϵ)**-indistinguishable,** $U \sim^c V$**.** No (t, ϵ)-distinguisher has an advantage greater than ϵ over the problem in time t.

At one point, this needs to be connected to the real world to have any practical use. This is often done by defining what we mean by a negligible probability, in terms of some security parameter. We will instead state how good a distinguisher is, and leave it to the application to decide if the probability is negligible. Equivalently, we can state how hard we believe a problem to be.

If we had defined negligibility and thus indistinguishability as described above, then we would have been able to say that indistinguishability is transitive. Gjøsteen, Petrides and Steine gives a sufficient although somewhat weaker transitivity theorem in [14]. The formulation below is adapted to this setting.

Theorem 2.1. *If* U *and* V *are* ϵ*-indistinguishable, and* V *and* W *are* ϵ'*-indistinguishable, then* U *and* W *are* $\epsilon + \epsilon'$*-indistinguishable.*

The proof is a telescope and triangle inequality argument. The result will be used to do honest verifier zero knowledge arguments in steps.

Example 2.2 (Decisional Diffie-Hellman)**.** We will often use the Decisional Diffie-Hellman problem (DDH) in the security analysis. Let G be a group with generator g. We then have two distributions

$$\mathcal{D} = \left\{(g^a, g^b, g^{ab})\right\}$$
$$\mathcal{D}' = \left\{(g^a, g^b, g^c) \mid c \text{ random}\right\},$$

and we can then construct the distinguishing problem $(\mathcal{D}, \mathcal{D}', G \times G \times G)$. This problem is believed to be (t, ϵ)-hard in certain groups.

2.2 Interactive Turing Machines

We use probabilistic and interactive Turing machines as the computational framework in this thesis. *Probabilistic* refers to the property that the next state and action is selected from available transitions subject to a probability distribution. This formal definition is due to [16]:

Definition 2.3 (Interactive Turing Machine)**.** An *interactive Turing machine* (ITM) is a Turing machine equipped with a read-only input tape, a work tape, a random tape, one read-only communication tape, and one write-only communication tape. The random tape contains an infinite sequence of random bits, and can be scanned only from left to right. We say that an interactive Turing machine *flips a coin*, meaning that it reads the next bit in its own random tape.

2.3 Zero Knowledge

The main tool of this thesis is the concept of *zero knowledge proofs*. It allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of knowledge of a value or correctness of a computation, but without revealing any secrets to the verifier, or anybody else. Applications include authentication, proving knowledge of the contents of a ciphertext, correctness of decryption, and so on. This will soon be made more precise.

Zero knowledge proofs were first introduced by Goldwasser, Micali and Rackoff [15] in 1985. During just a few years, several authors looked into how the concept could be used in cryptography [10, 32], and made non-interactive [10, 3, 5].

2.3.1 Commitments

The notion of *commitments* is a central building block for cryptography in general and zero knowledge proofs in particular. A commitment has two properties:

- it is *binding*, meaning that it should be hard for $\mathcal{P}$ to change the value at a later stage, and
- it is *hiding*, meaning that it must be hard for $\mathcal{V}$ to extract any additional information about the commitment.

A secure commitment scheme is always dependent on randomness, since a powerful opponent could simply compute any commitment, and check for matches. In the following discussion, we consider commitments to bitstrings of length m. Assume that the random number is a bit string of length n, then we can define

$$\text{commit} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^k$$

An opening to commitment c is a tuple (m, r) such that $c = \text{commit}(m, r)$.

These properties can be either *unconditional* or *computational*, but at most one property can be unconditional at any time.

Unconditional binding and computational hiding This means that not even a computationally unbounded prover will be able to change his mind, i.e. if $\mathcal{P}^*$ committed to m using r, then there is no (m', s) such that $\text{commit}(m, r) = \text{commit}(m', s)$.

For it to be computationally hidden, we need that the distribution of commitments to m is computationally indistinguishable from the distribution of commitments to m'.

Computational binding and unconditional hiding Let $\mathcal{A}$ be any probabilistic algorithm with resource bound t that outputs a two tuples (m, r) and (m', s). Then we demand that

$$\Pr\left[\text{commit}(m, r) = \text{commit}(m', s)\right] < \epsilon$$

Unconditional hiding means that the distribution of commitments to m is perfectly indistinguishable from the distribution of commitments to m'. Note that this means that there for any valid opening (b, r) exists another valid opening (m', s).

From the last remark, we get that it is impossible for a commitment scheme to be both unconditionally binding and hiding, since unconditional binding implies that no collision should exists, while unconditional hiding means that any pair should have a collision that opens to a different value.

Example 2.4 (Pedersen commitments, [29])**.** Let g and h be generators of G. To commit to m, select a random r, and compute

$$\text{commit}(m, r) = g^m h^r.$$

The commitment is opened by revealing m and r. This scheme is perfectly hiding and computationally binding. The commitment is uniformly distributed in G as long as r is chosen uniformly in $\mathbb{Z}_q$. In order to prove that it is computationally binding, assume that there exist two openings (m, r) and (m', s). We then compute the logarithm of h as

$$\log_g h = \frac{r - s}{m' - m},$$

which is assumed to be hard.

Example 2.5 (ElGamal commitments)**.** As above, let g and $h = g^a$ be generators of G, and assume that $\log_g h$ is unknown. One commits by computing $(g^r, g^m h^r)$, and opens by revealing m and r. This scheme is computationally hiding and perfectly binding. It is easy to show the binding property. Assume that $(x, y) = (g^r, g^m h^r)$, and let (m', s) be a different opening. Then $g^r = g^s$, so $r = s$, and so $m + ar = m' + as$, so $m = m'$.

The hiding property is more complicated to prove. The strategy is to make a reduction to DDH, and show that distinguishing commitments to two different messages is at least as hard as the DDH problem.

2.3.2 Proofs and arguments

We need to formalise the notion of a conversation between two ITMs.

Definition 2.6. An *interactive proof system* $(\mathcal{P}, \mathcal{V})$ for a set S is a two party game between a probabilistic polynomial time verifier $\mathcal{V}$ and an unbounded prover $\mathcal{P}$, satisfying

Completeness For every $x \in S$, the verifier accepts after interacting with P on public input x and potentially auxiliary, private input y to $\mathcal{P}$.

Soundness If $x \notin S$, then for any $\mathcal{P}^*$, the verifier accepts with probability at most ϵ after interacting with $\mathcal{P}^*$ on public input x.

We use the term *argument* when the prover is only assumed to be probabilistic polynomial time, and we will use the word protocol to denote either argument or proof system. Note that an argument without auxiliary input to the prover is somewhat meaningless, since the verifier then could do everything the prover could. The auxiliary input is usually called a *witness*.

In our case, $x \in S$ will mean that a claim is true, for instance that the shuffle and decryption was done correctly. We also want to prevent $\mathcal{V}$ from collecting any information from the protocol.

Definition 2.7. The protocol $(\mathcal{P}, \mathcal{V})$ is *zero knowledge* if for any probabilistic polynomial time $\mathcal{V}^*$ there is a simulator $\mathcal{M}$ running in expected probabilistic polynomial time such that $\mathcal{M} \sim^c (\mathcal{P}, \mathcal{V}^*)$ on public input x.

Sometimes we can trust the verifier to send honest challenges, which gives us a weaker definition that turns out to be very useful.

Definition 2.8. The protocol $(\mathcal{P}, \mathcal{V})$ is *honest verifier zero knowledge* if there exists a probabilistic polynomial time simulator $\mathcal{M}$ running on public input such that $M \sim^c (\mathcal{P}, \mathcal{V})$.

The definition can be strengthened by requiring the indistinguishability to be either statistical or perfect.

2.3.3 Σ-protocols

There is a special flavour of zero knowledge arguments called Σ-protocols.

Definition 2.9. Let R be a relation, and let $(x, w) \in R$. Then $(\mathcal{P}, \mathcal{V})$ is a Σ-protocol if it is of a three round form with commitment α, challenge e and response z, where x is the public input and w is the private input to $\mathcal{P}$.

Completeness Whenever $(x, w) \in R$, then $\mathcal{V}$ must accept.

Special soundness For any two accepting conversations (α, e, z) and (α, e', z') with $e \neq e'$, then one can efficiently compute w such that $(x, w) \in R$.

Special HVZK There exists a polynomial-time simulator M, which on input x and e outputs an accepting conversation of the form (α, e, z), indistinguishable from a real conversation (P, V).

The special soundness condition is a special case of *proof of knowledge*, a notion that will be introduced shortly. A particularly nice property of Σ-protocols is that a such can be transformed into a true zero knowledge protocol by using a suitable commitment scheme and adding one more round.

Example 2.10 (Graph isomorphism)**.** This protocol must be repeated a number of times with distinct commitments in order to be secure. It is a zero knowledge proof that the prover knows an isomorphism between two graphs, and hence that they are isomorphic.

Public input: Graphs G_0 and G_1.
Private input to $\mathcal{P}$: σ such that $\sigma : G_0 \to G_1$ is an isomorphism

Prover		Verifier
Pick random iso τ from G_1		
Compute $H = \tau(G_1)$	$\xrightarrow{H}$	
	$\xleftarrow{b}$	$b \xleftarrow{r} \{0, 1\}$
$f \leftarrow \tau \circ \sigma^{1-b}$	$\xrightarrow{f}$	$G_b \simeq H$ by f?

Figure 2.1: Isomorphism of graphs.

It is clear that this protocol is complete. In order to cheat, $\mathcal{P}^*$ needs to guess the challenge in advance, and compute H isomorphic to the correct graph. Clearly, this has probability $\frac{1}{2}$, so by increasing the number of independent rounds, we can reach any soundness bound.

This is zero knowledge since it is hard to factor f correctly when none of the factors are known. We make a simulator by reading b, and then publishing a

random isomorphism f from G_b to a graph H. This is perfect special honest verifier zero knowledge, since H is selected from the same distribution in both the real and simulated protocol, and hence f is distributed equally.

2.4 Knowledge

We begin with a definition that in some sense is stronger than the soundness requirement.

Definition 2.11 (Proof of knowledge)**.** Let $\mathcal{P}^*$ be any polynomial time interactive Turing machine, which on public input, time T and ℓ moves with $\mathcal{V}$ makes $\mathcal{V}$ accept with probability ϵ. The protocol is a proof of knowledge if there exists a simulator $\mathcal{S}$ depending on $\mathcal{P}^*$ with public input such that $\mathcal{S}$ outputs the private input to $\mathcal{P}$ in time $f(T)$ and chance of success $g(\epsilon)$, where f and g are nice functions.

Proof of knowledge implies soundness if there exists a witness, since a proof of knowledge of a witness implies its existence. However, proving that an argument is a proof of knowledge sometimes requires a weaker reduction than a soundness proof.

The notion of "nice" is somewhat undefined. We require that the work required should be feasible, and that the chance of success not should be substantially worse that ϵ.

The forking lemma is a useful tool for proving that a protocol is sound and a proof of knowledge. It was first introduced by Pointcheval and Stern in 1996 [31] and later generalised by Bellare and Neven in 2006 [2]. This formulation and the proof is by Bellare and Neven.

Fix an integer $n \geq 1$ and a set H of size $h \geq 2$. Let $\mathcal{A}$ be a randomised algorithm that on input $x, h_1, \ldots, h_n$ returns a pair, the first element of which is an integer in the range $0, \ldots, n$ and the second element of which we refer to as a *side output*. Let $\mathcal{IG}$ be a randomised algorithm that we call the input generator. The accepting probability of $\mathcal{A}$, denoted acc, is defined as the probability that $J \geq 1$ in the experiment

$$x \xleftarrow{r} \mathcal{IG} \qquad h_1, \ldots, h_n \xleftarrow{r} H \qquad (J, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_n)$$

The *forking algorithm* $\mathcal{F}_\mathcal{A}$ associated to $\mathcal{A}$ is the randomised algorithm that takes input x and proceeds as follows:

1. Pick coins ρ for $\mathcal{A}$ at random
2. $h_1, \ldots, h_n \xleftarrow{r} H$
3. $(I, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_n; \rho)$

4. if $I = 0$ then return $(0, \epsilon, \epsilon)$
5. $h'_I, \ldots, h'_n \xleftarrow{r} H$
6. $(I', \sigma') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_n)$
7. if $I = I'$ and $h_I \neq h'_I$, then return $(1, \sigma, \sigma')$
8. else, return $(0, \epsilon, \epsilon)$

Lemma 2.12 (The forking lemma, (Lemma 1, [2]))**.** *Let $\mathcal{IG}$ and $\mathcal{F}_\mathcal{A}$ be as above, and let* frk *be the probability that the above algorithm is successful. Then*

$$\text{frk} = \Pr\left[b = 1 \mid x \xleftarrow{r} \mathcal{IG}, (b, \sigma, \sigma') \xleftarrow{r} \mathcal{F}_\mathcal{A}(x)\right] \geq \text{acc} \cdot \left(\frac{\text{acc}}{n} - \frac{1}{h}\right).$$

Proof. First, we introduce two new probabilities. Fix $x \leftarrow \mathcal{IG}$, and let $\text{acc}(x)$ be the probability that $J \geq 1$ in the experiment

$$h_1, \ldots, h_n \xleftarrow{r} H \qquad (J, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_n)$$

and let $\text{frk}(x)$ be the probability that $b = 1$ when $\mathcal{F}_\mathcal{A}$ is run on x. Now acc and frk are the expected values of $\text{acc}(x)$ and $\text{frk}(x)$ over all x. We claim that the inequality holds for every x:

$$\text{frk}(x) \geq \text{acc}(x) \cdot \left(\frac{\text{acc}(x)}{n} - \frac{1}{h}\right). \tag{2.1}$$

Let $\mathcal{R}$ be the set of all coins, and for each i, define $X_i : \mathcal{R} \times H^{i-1} \to [0, 1]$ as

$$X_i(\rho, h_1, \ldots, h_{i-1}) = \Pr\left[J = i | h_i, \ldots, h_n \xleftarrow{r} H, (J, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_n)\right]$$

Now X_i can be regarded as a uniform random variable over its domain. Let I and I' be the return codes as in the forking algorithm. We then have

$$\begin{aligned}
\Pr[I = I' \text{ and } I \geq 1] &= \sum_{i=1}^{n} \Pr[I' = i \mid I = i] \cdot \Pr[I = i] \\
&= \sum_{i=1}^{n} \frac{1}{|\mathcal{R}| \, |H|^{i-1}} \sum_{\rho, h_1, \ldots, h_{i-1}} X_i(\rho, h_1, \ldots h_{i-1})^2 \\
&= \sum_{i=1}^{n} \mathbf{E}\left[X_i^2\right] \geq \sum_{i=}^{n} (\mathbf{E}[X_i])^2 \\
&\geq \frac{1}{n} \left(\sum_{i=1}^{n} \mathbf{E}[X_i]\right)^2 = \frac{\text{acc}(x)^2}{n},
\end{aligned}$$

which proves the claim.

Next, we have

$$\begin{aligned}\mathrm{frk}(x) &= \Pr\left[I = I' \text{ and } I \geq 1 \text{ and } h_I \neq h'_I\right] \\ &\geq \Pr\left[I = I' \text{ and } I \geq 1\right] - \Pr\left[I \geq 1 \text{ and } h_I = h'_I\right] \\ &\geq \frac{\mathrm{acc}(x)^2}{n} - \Pr\left[I \geq 1\right] \cdot \Pr\left[h_I = h'_I\right] \\ &= \frac{\mathrm{acc}(x)^2}{n} - \frac{\mathrm{acc}(x)}{h}\end{aligned}$$

and we reach the conclusion by computing the expectation:

$$\begin{aligned}\mathrm{frk} = \mathbf{E}\left(\mathrm{frk}(x)\right) &\geq \mathbf{E}\left(\frac{\mathrm{acc}(x)^2}{n} - \frac{\mathrm{acc}(x)}{h}\right) = \frac{\mathbf{E}\left(\mathrm{acc}(x)^2\right)}{n} - \frac{\mathbf{E}\left(\mathrm{acc}(x)\right)}{h} \\ &\geq \frac{\mathbf{E}\left(\mathrm{acc}(x)\right)^2}{n} - \frac{\mathbf{E}\left(\mathrm{acc}(x)\right)}{h} = \frac{\mathrm{acc}^2}{n} - \frac{\mathrm{acc}}{h}\end{aligned}$$

□

When we use this lemma later, we will refer to it as saving a state and replaying from it. At first, the lemma does not seem to allow this use, but whenever we refer to the forking lemma, we will in reality refer to the idea of the proof.

We now introduce a new assumption, namely that all parties share a common reference string. Then we have a counterpart to proof of knowledge. In [19], this is called *witness-extended emulation*, and treats both $\mathcal{P}^*$ and $\mathcal{V}$ as blackboxes.

Definition 2.13 (Witness-extended emulation, (Definition 4, [19]))**.** We say that the public coin argument $(\mathcal{P}, \mathcal{V})$ has witness-extended emulation if for all deterministic polynomial-time $\mathcal{P}^*$ there exists a polynomial-time emulator $\mathcal{E}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ and a given common reference string σ, we have

$$\begin{aligned}\Pr\left[\mathcal{A}(\mathrm{tr}) = 1 \mid (x, s) \leftarrow \mathcal{A}(\sigma); \mathrm{tr} \leftarrow \langle \mathcal{P}^*(\sigma, x, s), V(\sigma, x)\rangle\right] \simeq \\ \Pr[\mathcal{A}(\mathrm{tr}) = 1 \text{ and tr accepting, then } (\sigma, x, w) \in R \\ \mid (x, s) \leftarrow \mathcal{A}(\sigma); (\mathrm{tr}, w) \leftarrow \mathcal{E}(\sigma, x)]\end{aligned}$$

where $\mathcal{E}$ has access to a transcript oracle $\langle \mathcal{P}^*(\sigma, x, s), \mathcal{V}(\sigma, x)\rangle$ that can be rewound to a particular round and run again with the verifier choosing fresh random coins.

The notable advantage of this is that it can be used to improve proofs of computational soundness. If the first transcript is non-accepting with probability $1 - \epsilon$, we can cancel the whole process, with probability $1 - \epsilon$. Given that we want to gather n transcripts using the same commitments, we would usually need to run the protocol expected $\frac{n}{\epsilon}$ times, and the work might be comparable to actually

computing a discrete logarithm. Instead, we first run it once, and only continue if it was successful. The expected number of runs is then

$$(1-\epsilon)\cdot 1+\epsilon\cdot\left(\frac{n-1}{\epsilon}+1\right)=n,$$

which may be feasible.

2.5 Examples

The following are examples of Σ-protocols, but the presentation is varied to highlight different aspects.

Example 2.14 (Schnorr)**.** This protocol was introduced by Schnorr [32] in 1989, and has since become the standard example for zero knowledge protocols. It allows $\mathcal{P}$ to prove that he knows a discrete logarithm without revealing any knowledge about it.

Public input: $G=(g), |G|=q$ and h.
Private input to $\mathcal{P}$: a such that $h=g^a$.

Prover		Verifier
$r \xleftarrow{r} \mathbb{Z}_q$		
$\alpha \leftarrow g^r$	$\xrightarrow{\alpha}$	
	$\xleftarrow{e}$	$e \xleftarrow{r} \mathbb{Z}_q$
$z \leftarrow ae+r$	$\xrightarrow{z}$	$g^z \stackrel{?}{=} \alpha h^e$

Figure 2.2: Schnorr identification protocol.

This is a Σ-protocol. Completeness is clear by writing out z. Special soundness holds since if two instances are run with the same initial commitment, then one can compute $a=\frac{z-z'}{e-e'}$. For special HVZK, define the simulator as such:

1. Select z at random.
2. Compute α as g^z/h^e.
3. Output (α, e, z).

It is clear that the distributions of real and simulated transcripts are indistinguishable, since it is impossible to decide whether α or z was chosen first.

Public input: $G = (g), |G| = q, h, y$ and z.
Private input to $\mathcal{P}$: x such that $h = g^x$ and $z = y^x$

Prover		Verifier
$s \xleftarrow{r} \mathbb{Z}_q$		
$(a, b) \leftarrow (g^s, y^s)$	$\xrightarrow{(a,b)}$	
	$\xleftarrow{c}$	$c \xleftarrow{r} \mathbb{Z}_q$
$r \leftarrow s + cx$	$\xrightarrow{r}$	$g^r \stackrel{?}{=} ah^c$
		$y^r \stackrel{?}{=} bz^c$

Figure 2.3: Chaum-Pedersen.

Example 2.15 (Chaum-Pedersen)**.** In [7], Chaum and Pedersen demonstrated how to prove that two discrete logarithms, $\log_g h$ and $\log_y z$, are equal without revealing the logarithm itself.

It is clear that this protocol is complete. As with Schnorr, this is also a Σ-protocol. Special soundness is similar to Schnorr, and a simulator proving SHVZK will pick a random r and compute (a, b) as $(g^r/h^c, y^r/z^c)$. Chaum-Pedersen is unconditionally sound [8], meaning that even an all-powerful cheating prover essentially can do no better than guessing the challenge.

Assume that $z = h^{x+\Delta}$ and $b = y^{s+\delta}$. Then, since $g^r = ah^c$ and $y^r = bz^c$, we have that $r = s + cx$ and $r = s + \delta + c(x + \Delta)$. Combining, we get that $\delta + c\Delta = 0$. This requires the prover to have guessed c in advance, giving a soundness bound of $\frac{1}{q}$.

Example 2.16 (Proof of Diffie-Hellman triple)**.** This is a proof to demonstrate that a given triple (A, B, C) is a Diffie-Hellman triple (g^a, g^b, g^{ab}). The structure is similar to all three above. The protocol is from [26].

Protocol 1. A group $G = (g)$ of prime order q, and group elements (A, B, C). $\mathcal{P}$ also knows a and b, $(A, B, C) = (g^a, g^b, g^{ab})$.

1. $\mathcal{P}$ selects $s \xleftarrow{r} \mathbb{Z}_q$ and computes $(X, Y) \leftarrow (g^s, B^s)$. The tuple (X, Y) is then sent to $\mathcal{V}$.

2. $\mathcal{V}$ selects a random challenge $c \xleftarrow{r} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ responds by sending $r \leftarrow s + ca$ to $\mathcal{V}$, who accepts if

$$g^r = XA^c \quad \text{and} \quad B^r = YC^c$$

In essence, this is only Chaum-Pedersen to prove that $\log_g A = \log_B C$, but we note it for further reference.

If we consider g^a a commitment, then this is a proof that C is a commitment to the product of the values committed to in A and B. There exist a similar proof for Pedersen commitments, and it is not too hard to construct for other commitment schemes as well [26].

2.6 Non-Interactive Zero Knowledge

We often want to avoid the interactivity of the proofs, while maintaining the same or even better security. One clear advantage with interactive proofs is that the proof can be verified again at a later stage, for as we have seen, a defining property for zero knowledge is that a real transcript should be indistinguishable from a simulated transcript.

Another advantage is that it leaves the verifier unable to cheat actively – i.e. it is always an honest verifier. It is easier to design protocols to be HVZK rather than true ZK. The most well-known technique for making a protocol interactive is the *Fiat-Shamir heuristic*, but there has also been later development, such as one by Groth, Ostrovsky and Sahai [20].

2.6.1 The Random Oracle Model

The random oracle model was formalised by Bellare and Rogaway [3] in 1993. Previously, many practitioners had used some of the same techniques by letting hash functions represent an honest third party. Bellare and Rogaway show that some of these may be justified, but not as a rule of thumb.

A random function is a function $H : \{0,1\}^* \to \{0,1\}^n$ such that the output of $H(x)$ is selected randomly and independent of all other output.

In the Random Oracle Model, there exist a random oracle with access to such functions for all n, so that two equal queries always have the same response. All players can query the same random oracle. In the real application, the oracle is then exchanged for a cryptographic hash function. The last step is in its nature heuristic, as no function can be truly random. This implies that any security proof in the random oracle model may fail to hold in the standard model.

2.6.2 Fiat-Shamir Transformation

The idea of the Fiat-Shamir transformation was first used by Fiat and Shamir [10] in 1986, and later formalised in [3]. Informally, the idea is that since the verifier only asks random questions, all we really require is the unpredictability of the

challenges. Hence, that task can be done by a random oracle, which will give random replies to queries containing the commitment. The random oracle is then modelled by a family of hash functions.

The following formulation is due to [4].

Definition 2.17. Let $\Sigma = (\mathcal{P}_\Sigma, \mathcal{V}_\Sigma)$ be a Σ-protocol, and let H be a hash funtion. The weak Fiat-Shamir transformation of Σ is the proof system $w\mathcal{FS}(\Sigma) = (\mathcal{P}, \mathcal{V})$ defined as follows:

$\mathcal{P}(x, w)$ Run $\mathcal{P}_\Sigma(x, w)$ to obtain a commitment α. Compute $e \leftarrow H(\alpha)$, and input e to $\mathcal{P}_\Sigma$. Get a response z, and output (e, z).

$\mathcal{V}(x, e, z)$ Compute α from e and z, then run $\mathcal{V}_\Sigma(\alpha, e, z)$.

The strong Fiat-Shamir transformation $s\mathcal{FS}(\Sigma)$ of Σ is obtained as above with the difference that e is computed as $e \leftarrow H(x, \alpha)$.

This idea can be extended to other protocols than Σ-protocols. The unpredictability is maintained by including even more data in the hashes.

The Fiat-Shamir heuristic has received much criticism [4, 22], and it has been demonstrated that provably secure Σ-protocols can become insecure with any hash function used in the transformation [22]. Still, Fiat-Shamir remains in use, due to its efficiency and strong practical security record.

Some of the latest development includes a work by Groth, Ostrovsky and Sahai. In [21], they develop a perfect and universally composable NIZK argument based on commitment schemes. In contrast to Fiat-Shamir, this approach is proven secure in the common reference string model.

2.7 Exponentiation

In order for a cryptographic system to be usable, we need it to be practical as well as secure. Even low-degree polynomial algorithms may be forbiddingly expensive so we are typically less interested in asymptotic runtime. Instead, we are interested in what we can expect with reasonable input, so the big-O notation will be too wide for us. We will rather count the specific number. Exponentiation is typically the most expensive operation in cryptography, and it is therefore also the most interesting measure for how time-consuming an algorithm is.

The most important factors are the length of the exponent and the size of the group. The latter must be large enough to preserve security, hence it cannot be sacrificed to improve performance. In certain cases, exponents can be chosen short, when we don't need the discrete logarithm problem to be hard. In our case, a short exponent will be of length ℓ, and can be between 128 and 256 bit, while a long exponent is about 2000 bit.

2.7.1 Precomputation

Precomputation is a very efficient technique when the base is known in advance [17], and the idea is used in numerous algorithms. One such algorithm is [24], where the authors demonstrate that a computation with a 512 bit exponent can be done in at most 64 multiplications. To comparison, the square-and-multiply algorithm will use 1022 multiplications in worst-case.

The idea is to take an exponent x of length n and split it into h blocks of size $a = \lceil \frac{n}{h} \rceil$. Each block is then divided into another v blocks of size $b = \lceil \frac{a}{v} \rceil$. We call $h \times v$ the configuration of the algorithm. Let $g_0 = g$ and for each $0 \leq i \leq h$, compute $g_i = g_{i-1}^{2^a} = g^{2^{ia}}$.

We can then compute

$$g^x = \prod_{j=0}^{v-1} \prod_{i=0}^{h-1} \left(g_i^{2^{jb}} \right)^{x_{i,j}}$$

where $x_{i,j}$ is a block of x. Lim-Lee changes the expression further in order to optimise the execution [24].

The speed-up comes at a cost, as it requires that a number of exponentiations are computed in advance, and hence storage. Based on formulas given in [24], we can fill in Table 2.1. For reference, the expected number of multiplications is 3070.5, with worst-case being 4094 for a 2048 bit exponent, when using square-and-multiply.

The size of the precomputation should be limited so that it can fit in the CPU cache. Level 2 cache can typically be at most a couple of megabytes, while newer processors also have a level 3 cache with up to 20 MB capacity.

Configuration	Precomp. values	Storage	Average mult.	Worst-case
3×1	7	1.75 kB	1278.6	1364
4×4	60	15 kB	606.0	638
6×3	189	47 kB	448.7	454
8×7	1 785	446 kB	290.0	291
13×9	73 719	18 MB	174.0	174
15×13	425 971	104 MB	146.0	146

Table 2.1: Selected parameters for 2048 bit exponents. The notation is as in [24].

For later reference, we can achieve an efficient speed-up of at least 80 % using precomputation over ordinary square-and-multiply.

2.7.2 Multi-exponentiation

The final technique we mention is multi-exponentiation. This is developed to compute the product of exponents

$$\prod_{i=0}^{n-1} g_i^{e_i}$$

more efficiently than ordinary square-and-multiply, and then multiplication. According to an unpublished manuscript of Lim [23], an algorithm based on [6] can give a 2 to 4 times speed-up compared to square-and-multiply. The idea of the algorithm is to use sliding windows.

3 Roots of polynomials

In the introduction, we mentioned that there are two major ideas for shuffling. One is based on permutation of roots of a polynomial. The fundamental idea in this chapter is that two polynomials are equal if and only if they always evaluate to the same. Hence, if two polynomials have the same value at a random point, then they are with high probability equal.

The idea was introduced by Neff [27, 28], and has in particular been used by Groth, for instance [18, 19].

In particular, we note the following lemmas.

Lemma 3.1. *Let $f(x), g(x) \in \mathbb{Z}_q[x]$ be monic polynominals of degree at most d, with $f \neq g$. Then there are at most $d-1$ values $x_1, \ldots, x_{d-1} \in \mathbb{Z}_q$ for which $f(x_i) = g(x_i)$. Furthermore, if t is selected at random from $\mathbb{Z}_q$, then*

$$\Pr\left[f(t) = g(t) \mid t \xleftarrow{r} \mathbb{Z}_q\right] \leq \frac{d-1}{q}$$

The proof is based on the fact that a polynomial of degree d over a field has at most d roots.

Lemma 3.2. *Let $f(x), g(x) \in \mathbb{Z}_q[x]$ be any two polynomials of degree at most d. Fix non-zero contants R, γ, δ. Then*

$$\Pr\left[f(\gamma t) = Rg(\delta t) \mid t \xleftarrow{r} \mathbb{Z}_q\right] \leq \frac{d}{q}$$

We also note that evaluation of polynomials as product of roots is stable under permutations. Given a polynomial f over a finite field, with d roots $x_1, \ldots x_d$ and a permutation $\pi : \{1, \ldots d\} \to \{1, \ldots d\}$, then

$$f(x) = a\prod_{i=1}^{d}(x - x_i) = a\prod_{i=1}^{d}(x - x_{\pi(i)})$$

where a is the leading coefficient.

Therefore, we want to come to the situation where the prover can prove that he has a polynomial which consist of the relevant data, and allow the verifier to evaluate it at a random point. The lemmas above then guarantee that the polynomial is correct.

We describe a very simple demonstration of this idea.

Let x_i, y_i and γ be elements such that $y_i = \gamma x_{\pi(i)}$. Then $\gamma^k \prod_{i=1}^{k}(x_i - t) = \prod_{i=1}^{k}(y_i - \gamma t)$. Also, let X_i, Y_i and Γ be public commitments to x_i, y_i and γ respectively.

Now, we construct commitments $\hat{X}_i$ and $\hat{Y}_i$ such that an evaluation point t, chosen by the verifier, is included in some way.

Construct (recall Example 2.16) $C_1 = \hat{X}_1, C_2 = \hat{X}_2 C_1, \ldots, C_k = \hat{X}_k C_{k-1}$ and $D_1, \ldots, D_k$ similarily. Then $C_k = \prod_{i=1}^{k} \hat{X}_i$ and $D_k = \prod_{i=1}^{k} \hat{Y}_i$.

Finally, we produce a proof that C_k and D_k are equal polynomials, and hence that the roots are equal. Therefore, we have proved that $y_i = \gamma x_{\pi(i)}$. The primitives provided by Neff and Groth are both more efficient ways of implementing the above idea.

3.1 Existing verifiable shuffles

This section contains two summaries, one of Neff's articles [27, 28], and next of Groth's improvements.

The first step is to describe a simple k-shuffle protocol. In this setting, the prover will know everything about the shuffled data. This is unrealistic in most real applications, but is later used as a building block in the general setting. More specifically, the protocol works as a commitment to a permutation.

The simple shuffle protocol in turn based on a generalisation of the Chaum-Pedersen protocol (p. 15), called *Iterated Logarithmic Multiplication Proof Protocol* (ILMPP), which is used to prove that given two sets of group elements $\{X_i\}$ and $\{Y_i\}$, then

$$\prod_{i=1}^{k} \log_g X_i = \prod_{i=1}^{k} \log_\Gamma Y_i.$$

The security of the simple k-shuffle is based on unlikely event of picking a zero of a polynomial at random, and picking two vectors x, y at random such that $x \cdot y = 0$.

Problem 2 (Simple k-Shuffle Problem, [28]). Suppose that g and Γ, and two sequences $X_1, \ldots, X_k$, and $Y_1, \ldots, Y_k$ are all publicly known elements of $G \subseteq \mathbb{Z}_p^*$. Suppose also, that the prover $\mathcal{P}$ knows $\gamma = \log_g \Gamma$, $x_i = \log_g X_i$ and $y_i = \log_g Y_i$ for all $1 \leq i \leq k$, where g is some generator of G, but that all these logarithms are unknown to $\mathcal{V}$.

$\mathcal{P}$ is required to convince $\mathcal{V}$ that there is some permutation $\pi \in \Sigma_k$ with the property that for all $1 \leq i \leq k$,

$$Y_i = X_{\pi(i)}^{\gamma} \tag{3.1}$$

without revealing any information about x_i, y_i, γ or π.

The protocol is quite technical, and is given in Figure 3.1. The main idea is to set $\hat{x}_i = x_i - t$ and $\hat{y}_i = y_i - \gamma t$. Then the products of all $\hat{x}_i$ and $\hat{y}_i$ make two polynomials in t and with the same zeros, since $y_i = \gamma x_{\pi(i)}$.

The following two sets of equations are used in the protocol,

$$\begin{aligned}
\Theta_1 &\leftarrow g^{-\theta_1 \hat{y}_1} \\
\Theta_2 &\leftarrow g^{\theta_1 \hat{x}_2 - \theta_2 \hat{y}_k} \\
&\vdots \\
\Theta_k &\leftarrow g^{\theta_{k-1} \hat{x}_k - \theta_k \hat{y}_k} \\
\Theta_{k+1} &\leftarrow g^{\gamma \theta_k - \theta_{k+1}} \\
&\vdots \\
\Theta_{2k} &\leftarrow g^{\theta_{2k-1}}
\end{aligned} \tag{3.2}$$

for the prover, and

$$\begin{aligned}
\hat{X}_1^c \, \hat{Y}_1^{-\alpha_1} &= \Theta_1 \\
\hat{X}_2^{\alpha_1} \, \hat{Y}_2^{-\alpha_2} &= \Theta_2 \\
&\vdots \\
\hat{X}_k^{\alpha_{k-1}} \, \hat{Y}_k^{-\alpha_k} &= \Theta_k \\
\Gamma^{\alpha_k} \, g^{-\alpha_{k+1}} &= \Theta_{k+1} \\
&\vdots \\
\Gamma^{\alpha_{2k-1}} \, g^{-c} &= \Theta_{2k}
\end{aligned} \tag{3.3}$$

for the verifier.

For shorthand, we will refer to the protocol as $\mathrm{SS}(Y_i = X_{\pi(i)}^{\gamma})$.

Neff summarised the properties in a theorem.

Theorem 3.3 (Theorem 1, [28])**.** *The Simple k-Shuffle Proof Protocol satisfies the following properties:*

1. *It is a four-move, public coin proof of knowledge for the relationships in equation* (3.1)*.*

2. *It is complete.*

3. *The protocol is sound. If $\mathcal{V}$ generates challenges randomly, the* unconditional *probability of a forged proof is less than or equal to k/q.*

4. *It is honest verifier zero knowledge.*

Public input: $\{X_i\}$, $\{Y_i\}$, g and Γ.
Private input to $\mathcal{P}$: γ and π such that $\Gamma = g^\gamma$ and $Y_i = X_{\pi(i)}^\gamma$.

Prover		Verifier
	$\xleftarrow{\quad t \quad}$	$t \xleftarrow{r} \mathbb{Z}_q$
$\hat{x}_i \leftarrow x_i - t$		
$\hat{y}_i \leftarrow y_i - \gamma t$		
$\theta_1, \ldots, \theta_{2k-1} \xleftarrow{r} \mathbb{Z}_q$		
equations (3.2)	$\xrightarrow{\quad \Theta_1, \ldots, \Theta_{2k} \quad}$	
	$\xleftarrow{\quad c \quad}$	$c \xleftarrow{r} \mathbb{Z}_q$
$\alpha_i \leftarrow \theta_i + c \prod_{j=1}^{i} \left(\frac{\hat{x}_j}{\hat{y}_j} \right)$ for $1 \leq i \leq k$		
$\alpha_i \leftarrow \theta_i + c\gamma^{i-2k}$ for $k+1 \leq i \leq 2k-1$		
	$\xrightarrow{\quad \alpha_1, \ldots, \alpha_{2k-1} \quad}$	$U \leftarrow g^{-t}$
		$W \leftarrow \Gamma^{-t}$
		$\hat{X}_i \leftarrow X_i U$
		$\hat{Y}_i \leftarrow Y_i W$
		Check equations (3.3)

Figure 3.1: Simple k-Shuffle, $\mathrm{SS}(Y_i = X_{\pi(i)}^\gamma)$.

5. *The number of exponentiations required to construct the proof is* $2k$*, and the number of exponentiations required to verify it is* $4k + 2$*.*

We note that the $2k$ exponentiations for the prover is to base g, whereas the $4k$ exponentiations for the verifier can be computed using multiexponentiation.

He then proceeds to solve the following problem.

Problem 3 (ElGamal k-Shuffle Problem, [28]). Suppose that two sequences of pairs $(X_1, Y_1), \ldots, (X_k, Y_k)$ (input) and $(\bar{X}_1, \bar{Y}_1), \ldots, (\bar{X}_k, \bar{Y}_k)$ (output), as well as encryption parameters, g and h are all publicly known elements of $G \subseteq \mathbb{Z}_p^*$. Suppose also that the prover $\mathcal{P}$ knows $\beta_1, \ldots, \beta_k$ in $\mathbb{Z}_q$ and $\pi \in \Sigma_k$ such that for all $1 \leq i \leq k$

$$(\bar{X}_i, \bar{Y}_i) = (g^{\beta_{\pi(i)}} X_{\pi(i)}, h^{\beta_{\pi(i)}} Y_{\pi(i)}). \tag{3.4}$$

P is required to convince of $\mathcal{V}$ of this fact – that is, convince $\mathcal{V}$ of the existence of β_i and π satisfying equation (3.4) – without revealing any information about β_i or π.

Neff's strategy for solving this problem is to assume that there might be different reencryption exponents in first and second coordinate, say β_i and ξ_i and prove that there must exist *some* permutation such that $\beta_i = \xi_i$. This can be proven by showing that for any random vector s, $s \cdot \beta = s \cdot \xi$, where $\beta = (\beta_1, \ldots, \beta_k)$ and $\xi = (\xi_1, \ldots, \xi_k)$. However, since the components of s are permuted using π to form r, s and r must be kept secret. This is solved using the simple k-shuffle above.

We don't present the whole protocol here, but give the summarising result.

Theorem 3.4 (Theorem 2, [28])**.** *The ElGamal k-Shuffle Proof Protocol satisfies the following properties:*

1. *It is a seven-move, public coin proof of knowledge for the relationship in equation* (3.4)*.*

2. *It is complete.*

3. *The protocol is sound for* $k < q/2$*. Specifically, if* $\mathcal{V}$ *generates challenges randomly, the unconditional probability of a forged proof is less than* $(2k + 3)/q$*.*

4. *Assuming the Diffie-Hellman Decision Problem is hard, it is computationally zero-knowledge. The number of exponentiations required to construct the proof is* $8k + 4$*, and the number of exponentiations required to verify it is* $12k + 4$*.*

3.1.1 Groth's improvements

Groth [19] applies Neff's main ideas, but is able to move much of the computation from the exponents into products, saving many exponentiations. In particular, the shuffle of known contents and the preparations for it is much more efficient in Groth.

The structure of Groth's protocol for verifiable shuffle is much the same as Neff's. The main building block is a protocol for proving knowledge of a permutation π such that the shuffle of a known content is correct. The permutation of the real data is bound to the data used in the shuffle of known content.

Theorem 3.5 (Theorem 1, [19]). *The protocol in Figure 3.2 is a 4-move public coin special honest verifier zero-knowledge argument with witness-extended emulation for c for being a commitment to a permutation of the messages $m_1, \ldots, m_k$. If the commitment scheme is statistically hiding, then the argument is statistical honest verifier zero-knowledge. If the commitment scheme is statistically binding, then we have unconditional soundness, i.e., the protocol is an SHVZK proof.*

There are no given soundness bound in the original article. The soundness is based on $\mathcal{P}$ not being able to select an x such that a polynomial evaluates to a specific point. The probability of guessing such x is by Lemma 3.2 $\frac{k}{2^\ell}$. Also, $\mathcal{P}$ can guess e, so we get a soundness bound of

$$\frac{k+1}{2^\ell}.$$

3.2 Neff-based solution

In this section we propose a working, although inefficient solution to the Verifiable Shuffled Decryption problem. The protocol is an adaptation and modification of [28]. As with Neff, the main idea is to use a linear combination to hide the permutation, and the Simple k-shuffle to prove knowledge of it. The permutation is bound to the ciphertexts so that only the correct decryption will open the commitment.

Although too inefficient to be of any practical use, it is included because it displays the technique well.

Soundness is based on $\mathcal{V}$ picking his challenges randomly. Honest verifier zero knowledge holds if Computational Diffie-Hellman is hard.

Protocol 2. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

Public input: $c, m_1, \ldots, m_k$.
Private input to $\mathcal{P}$: π and r such that $c = \text{commit}(m_{\pi(1)}, \ldots, m_{\pi(k)}; r)$.

Prover		Verifier
	$\xleftarrow{\quad x \quad}$	$x \xleftarrow{r} \{0,1\}^\ell$
$d_1, \ldots, d_k, r_d, r_\Delta, r_a \xleftarrow{r} \mathbb{Z}_q$		
$\Delta_1 \leftarrow d_1, \Delta_2, \ldots, \Delta_{k-1} \leftarrow \mathbb{Z}_q, \Delta_k = 0$		
$a_i \leftarrow \prod_{j=1}^{i} (m_{\pi(j)} - x)$		
$c_d \leftarrow \text{commit}(d_1, \ldots, d_k; r_d)$		
$c_\Delta \leftarrow \text{commit}(-\Delta_1 d_2, \ldots, -\Delta_{k-1} d_k; r_\Delta)$		
$c_a \leftarrow \text{commit}(\Delta_2 - (m_{\pi(2)} - x)\Delta_1 - a_1 d_2, \ldots,$ $\Delta_k - (m_\pi(k) - x)\Delta_{k-1} - a_{k-1} d_k; r_a)$		
	$\xrightarrow{\quad c_d, c_\Delta, c_a \quad}$	
	$\xleftarrow{\quad e \quad}$	$e \xleftarrow{r} \{0,1\}^\ell$
$f_i \leftarrow e m_{\pi(i)} + d_i$		
$z \leftarrow er + r_d$		
$f_{\Delta_i} \leftarrow e(\Delta_{i+1} - (m_{\pi(i+1)} - x)\Delta_i + a_i d_{i+1})$ $- \Delta_i d_{i+1}$		
$z_\Delta \leftarrow e r_a + r_\Delta$		
	$\xrightarrow{\quad f_1, \ldots, f_n, z, f_{\Delta_1}, \ldots, f_{\Delta_{k-1}}, z_\Delta \quad}$	
		Check group membership
		$c^e c_d \stackrel{?}{=} \text{commit}(f_1, \ldots, f_n; z)$
		$c_a^e c_\Delta \stackrel{?}{=} \text{commit}(f_{\Delta_1}, \ldots, f_{\Delta_{k-1}}; z_\Delta)$
		$F_1 \leftarrow f_1 - ex$
		$eF_i \leftarrow F_{i-1}(f_i - ex) + f_{\Delta_{i-1}}$
		$F_k \stackrel{?}{=} e \prod_{i=1}^{k} (m_i - x)$

Figure 3.2: Shuffle of known content, due to [19].

1. For each i, $\mathcal{P}$ selects a_i, u_i and w_i at random from $\mathbb{Z}_q$. He also selects a random $\gamma \leftarrow \mathbb{Z}_q$, and computes

$$\Lambda_1 \leftarrow \prod_{i=1}^{k} x_i^{u_i} \tag{3.5}$$

$$\Lambda_2 \leftarrow \prod_{i=1}^{k} z_i^{-w_i} y_i^{u_i} \tag{3.6}$$

$$\Gamma \leftarrow g^{\gamma}$$
$$A_i \leftarrow g^{a_i}$$
$$C_i \leftarrow \Gamma^{a_{\pi(i)}}$$
$$U_i \leftarrow g^{u_i}$$
$$W_i \leftarrow \Gamma^{w_i}$$

   $\mathcal{P}$ sends $\Lambda_1, \Lambda_2, \{A_i\}, \{C_i\}, \{U_i\}, \{W_i\}$ and Γ to $\mathcal{V}$.

2. For each i, $\mathcal{V}$ selects small ρ_i at random from $\mathbb{Z}_q$, computes

$$B_i \leftarrow g^{\rho_i} U_i^{-1}$$

   and sends $\{\rho_i\}$ to $\mathcal{V}$.

3. $\mathcal{P}$ computes

$$b_i \leftarrow \rho_i - u_i$$
$$\sigma_i \leftarrow w_i + b_{\pi(i)}$$

   He also computes $d_i \leftarrow \gamma b_{\pi(i)}$ and $D_i \leftarrow g^{d_i} = B_{\pi(i)}^{\gamma}$. $\mathcal{P}$ then sends D_i to $\mathcal{V}$.

4. $\mathcal{V}$ selects a small random λ. He sends λ to $\mathcal{P}$, and computes

$$R_i \leftarrow A_i B_i^{\lambda}$$
$$S_i \leftarrow C_i D_i^{\lambda}$$

5. $\mathcal{P}$ computes $r_i \leftarrow a_i + \lambda b_i$ and $s_i \leftarrow a_{\pi(i)} + \lambda b_{\pi(i)}$. He then sends $\{\sigma_i\}$ to the verifier.

6. $\mathcal{V}$ computes

$$\Phi_1 \leftarrow \prod_{i=1}^{k} x_i^{-\rho_i}$$
$$\Phi_2 \leftarrow \prod_{i=1}^{k} z_i^{\sigma_i} y_i^{-\rho_i}$$

7. $\mathcal{P}$ and $\mathcal{V}$ execute the Chaum-Pedersen protocol to prove that

$$\log_g h = \log_{\Phi_1\Lambda_1} \Phi_2\Lambda_2$$

and the Simple k-shuffle to prove that

$$S_i = R_{\pi(i)}^{\gamma}$$

8. $\mathcal{V}$ accepts if he accepts the proofs in the previous step and if

$$\Gamma^{\sigma_i} = W_i D_i$$

for all i.

The properties of this protocol are given through the following propositions.

Proposition 3.6 (Completeness)**.** *Protocol 2 is complete.*

Proof. Both Chaum-Pedersen and the Simple k-shuffle are complete. Completeness of the whole protocol is clear from the following computation

$$\begin{aligned}\Phi_2\Lambda_2 &= \prod_{i=1}^{k} z_i^{\sigma_i} y_i^{-\rho_i} z_i^{-w_i} y_i^{u_i} = \prod_{i=1}^{k} m_{\pi(i)}^{w_i+\rho_{\pi(i)}-u_{\pi(i)}-w_i} \left(h^{r_i} m_i\right)^{-\rho_i+u_i} \\ &= \prod_{i=1}^{k} m_i^{\rho_i-u_i} m_i^{-\rho_i+u_i} \left(g^{ar_i}\right)^{-\rho_i+u_i} = \prod_{i=1}^{k} x^{a(-\rho_i+u_i)} \\ &= (\Phi_1\Lambda_1)^a\end{aligned}$$

□

Proposition 3.7 (Soundness)**.** *Protocol 2 is unconditionally sound.*

Proof. Recall that both the Simple k-shuffle and the Chaum-Pedersen protocol are unconditionally sound, with probability $\frac{k}{q}$ and $\frac{1}{2^\ell}$. Assume that $\mathcal{P}^*$ cheats, so that for no permutation τ, $z_i = m_{\tau(i)}$. This means that $\mathcal{P}^*$ can do anything from replacing the whole set of ciphertexts, to just modifying a few. Furthermore, assume that the protocol has been accepted. Especially, this means that

$$\Gamma^{\sigma_i} = W_i D_i \qquad \text{for all } i \tag{3.7}$$

$$\log_g h = \log_{\Phi_1\Lambda_1} \Phi_2\Lambda_2 \tag{3.8}$$

$$S_i = R_{\pi(i)}^{\gamma} \qquad \text{for a permutation } \pi \tag{3.9}$$

The discrete log in (3.8) is known to be a, and the γ in (3.9) is known to be $\log_g \Gamma$ by the input to the Simple k-shuffle. Also, there must exist values $\{a_i\}$, $\{c_i\}$, $\{u_i\}$

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

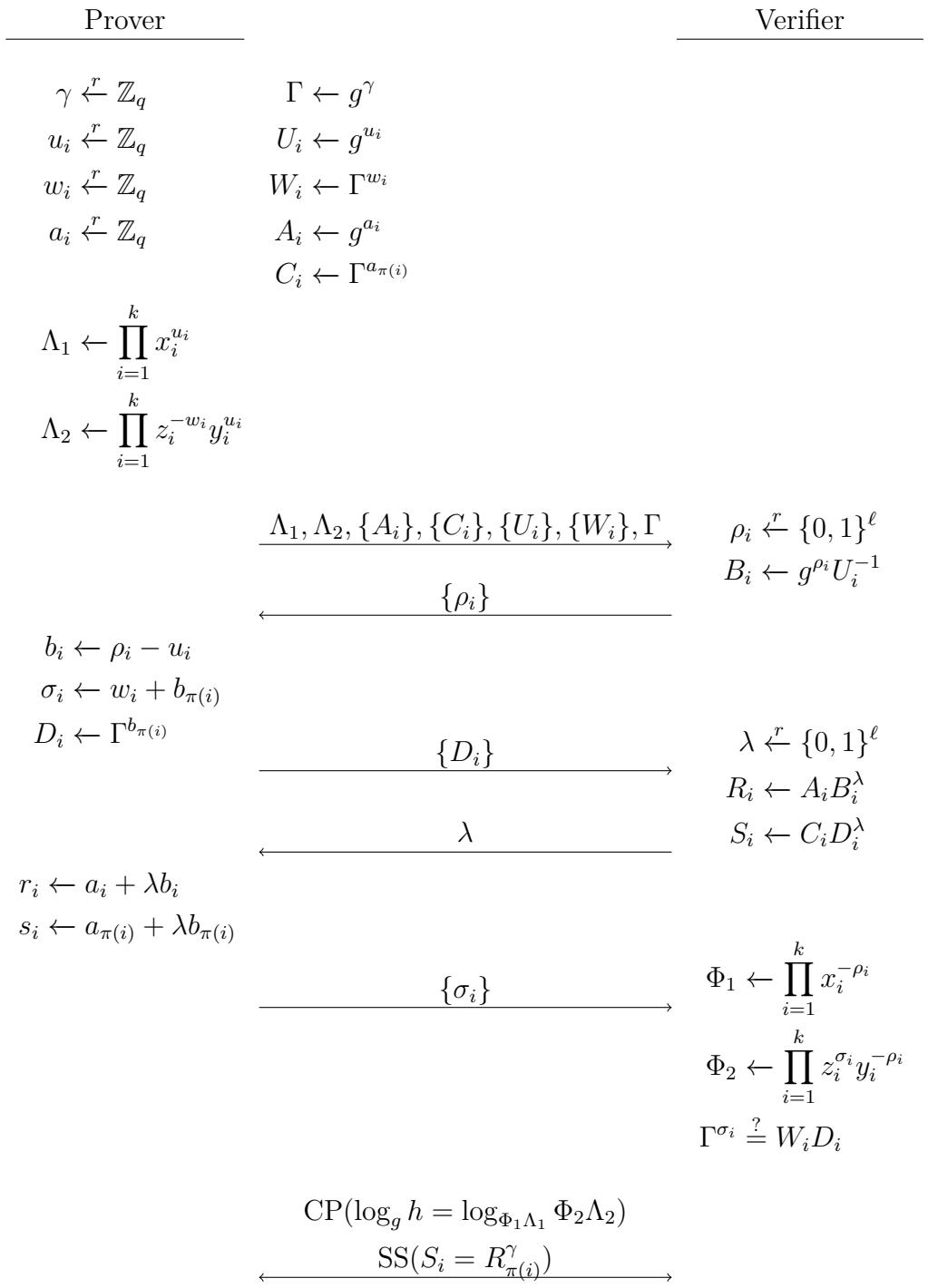

Figure 3.3: Protocol 2.

and $\{w_i\}$ such that

$$\begin{aligned} A_i &= g^{a_i} \\ B_i &= g^{b_i} \\ C_i &= \Gamma^{c_i} \\ D_i &= \Gamma^{d_i} \\ U_i &= g^{u_i} \\ W_i &= \Gamma^{w_i}. \end{aligned}$$

Also,

$$\begin{aligned} s_i &= r_{\pi(i)} \\ r_i &= a_i + \lambda b_i \\ s_i &= c_i + \lambda d_i \\ b_i &= \rho_i - u_i \\ \sigma_i &= w_i + d_i. \end{aligned}$$

Since λ is selected at random after the others have been chosen, $c_i = a_{\pi(i)}$ and $d_i = b_{\pi(i)}$ holds with probability $1 - 2^{-\ell}$. Therefore, $\sigma_i = w_i + \rho_{\pi(i)} - u_{\pi(i)}$.

We can insert this into (3.8).

$$\begin{aligned} \left(\Lambda_1 \prod_{i=1}^{k} x_i^{-\rho_i}\right)^a &= \Lambda_1^a \prod_{i=1}^{k} x_i^{-a\rho_i} \\ &= \Lambda_2 \prod_{i=1}^{k} z_i^{w_i + \rho_{\pi(i)} - u_{\pi(i)}} y_i^{-\rho_i} \\ &= \Lambda_2 \prod_{i=1}^{k} z_i^{w_i} z_i^{\rho_{\pi(i)}} z_i^{-u_{\pi(i)}} x_i^{-a\rho_i} m_i^{-\rho_i} \end{aligned}$$

Let $\beta = \prod_{i=1}^{k} z_i^{w_i} z_i^{-u_{\pi(i)}}$. We can then write

$$\Lambda_1^a \Lambda_2^{-1} \beta^{-1} = \prod_{i=1}^{k} z_i^{\rho_{\pi(i)}} m_i^{-\rho_i}$$

The left side of the equation as well as all z_i and m_i are fixed after step 1, so we can consider it as a constant. A cheating prover must therefore select z_i in advance such that the random linear combination on the right side equals the constant. The unconditional probability for this is $\frac{1}{q}$, so in total we have a soundness bound of

$$\frac{k}{q} + \frac{1}{q} + \frac{2}{2^\ell} < \frac{k+3}{2^\ell}.$$ □

Proposition 3.8 (Zero knowledge)**.** *Protocol 2 is honest verifier zero knowledge.*

Proof. We describe simulators that output valid transcripts. The final simulator is only given access to the public input. We then proceed to prove that the all the simulated transcripts are indistinguishable from a real transcript.

The simulators described below are cumulative: Unless stated otherwise, they will incorporate the changes from the previous simulators.

Simulator I Simulate the transcripts of Chaum-Pedersen and the Simple k-shuffle.

Simulator II Choose $\{\sigma_i\}$ at random, and compute $D_i \leftarrow \Gamma^{\rho_{\pi(i)}-u_i}$, $W_i \leftarrow \Gamma^{\sigma_i} D_i^{-1}$.

Simulator III Choose Λ_1 at random.

Simulator IV Choose Λ_2 at random.

Simulator V Use a random permutation τ in place of π for $\{C_i\}$ and $\{D_i\}$.

Simulator V will not use any of the secret input, and all checks will hold, due to the construction of W_i and D_i, and the simulation of the Chaum-Pedersen and Simple k-shuffle protocols. The transcripts are given in Table 3.1.

The result follows from the following chain of claims, due to transitivity of indistinguishability.

Claim 3.9. *The distribution of real transcripts is indistinguishable from the distribution of Simulator I transcripts.*

Proof. Both Chaum-Pedersen and the Simple k-shuffle are honest verifier zero knowledge. □

Claim 3.10. *The distribution of Simulator I transcripts is indistinguishable from the distribution of Simulator II transcripts.*

Proof. The only difference is which of w_i and σ_i is random, and which is computed using the other. □

Claim 3.11. *The distribution of Simulator II transcripts is indistinguishable from the distribution of Simulator III transcripts.*

Proof. It is clear that in order to distinguish, an adversary needs to decide whether Λ_1 and Λ_2 are well-formed or random.

We show that distinguishing real and simulated Λ_1 is at least as hard as DDH. Recall from the ElGamal scheme that $x_i = g^{r_i}$ for some random r_i. Then we have two distributions

$$\mathcal{IP} = \left\{ \left(g^{r_1}, \ldots, g^{r_k}, g^{u_1}, \ldots, g^{u_k}, g^{\sum_{i=1}^{k} r_i u_i} \right) \right\}$$
$$\mathcal{IP}' = \left\{ \left(g^{r_1}, \ldots, g^{r_k}, g^{u_1}, \ldots, g^{u_k}, g^{R} \right) \mid R \text{ random} \right\}$$

Symbol	CP	SS	$\{A_i\}$	$\{C_i\}$	$\{U_i\}$	Γ	$\{W_i\}$	Λ_1	Λ_2	$\{\rho_i\}$	$\{\sigma_i\}$	$\{D_i\}$
Real	real	real	g^{a_i} $a_i \xleftarrow{r} \mathbb{Z}_q$	$\Gamma^{a_{\pi(i)}}$	g^{u_i} $u_i \xleftarrow{r} \mathbb{Z}_q$	g^{γ} $\gamma \xleftarrow{r} \mathbb{Z}_q$	Γ^{w_i} $w_i \xleftarrow{r} \mathbb{Z}_q$	(3.5)	(3.6)	r	$w_i + d_i$	Γ^{d_i}
Sim. I	sim.	sim.	”	”	”	”	”	”	”	”	”	
Sim. II	”	”	”	”	”	”	$\Gamma^{\sigma_i} D_i^{-1}$	”	”	”	r	$\Gamma^{\rho_{\pi(i)} - u_i}$
Sim. III	”	”	”	”	”	”	”	r	”	”	”	”
Sim. IV	”	”	”	”	”	”	”	”	r	”	”	”
Sim. V	”	”	”	$\Gamma^{a_{\tau(i)}}$	”	”	”	”	”	”	”	$\Gamma^{\rho_{\tau(i)} - u_i}$

Table 3.1: Elements in the real and simulated transcripts. The symbol r denotes that the value is selected at random.

where the last element of $\mathcal{IP}$ is equal to a real Λ_1. The exponent of the last element is the inner product of the vectors created from the exponents. We may call this problem Decision Inner Product (DIP).

Now assume that $\mathcal{A}$ is an adversary with $\mathrm{Adv}_{DIP}(\mathcal{A}) = \epsilon$. We define $\mathcal{B}$ such: Given two Diffie-Hellman triples (A, B, C) and (A, B, C'), pick $\{r_2, \ldots, r_k\}$ and $\{u_2, \ldots, u_k\}$ at random, form two $2k + 1$ tuples as

$$(A, g^{r_2}, \ldots, g^{r_k}, B, g^{u_2}, \ldots, g^{u_k}, Cg^{\sum_{i=2}^{k} r_i u_i})$$
$$(A, g^{r_2}, \ldots, g^{r_k}, B, g^{u_2}, \ldots, g^{u_k}, C'g^{\sum_{i=2}^{k} r_i u_i})$$

and submit these to $\mathcal{A}$. He will then respond with a bit $b \in \{0, 1\}$, which $\mathcal{B}$ uses as his answer. It is clear that $\mathrm{Adv}_{DDH}(\mathcal{B}) = \epsilon$, and so $\mathcal{B}$ is successful if and only if $\mathcal{A}$ is successful. □

Claim 3.12. *The distribution of Simulator III transcripts is indistinguishable from the distribution of Simulator IV transcripts.*

Proof. We have

$$\begin{aligned}\Lambda_2 &= \prod_{i=1}^{k} z_i^{w_i} y_i^{-u_i} = \prod m_{\pi(i)}^{w_i} h^{-u_i r_i} m_i^{u_i} \\ &= \prod m_{\pi(i)}^{\sigma_i - \rho_{\pi(i)}} x_i^{-u_i a} = \Lambda_1^a \prod m_{\pi(i)}^{\sigma_i - \rho_{\pi(i)}}.\end{aligned}$$

Hence, loosely speaking, if Λ_1^a looks random, Λ_2 will too. More precisely, define two distributions as follows

$$\begin{aligned}\mathcal{D} &= \{(g, g^a, \Lambda_1, \Lambda_1^a)\} \\ \mathcal{D}' &= \{(g, g^a, \Lambda_1, r) \mid r \text{ random group element}\}\end{aligned}$$

Since Λ_1 is a group element, it is equal to g^β for some $\beta \in \mathbb{Z}_q$. It is then clear that the above problem is exactly DDH, which is assumed to be hard in G. □

Claim 3.13. *The distribution of Simulator IV transcripts is indistinguishable from the distribution of Simulator V transcripts.*

Proof. The only change is that we use a random permutation. The change only alters $\{C_i\}$ and $\{D_i\}$, since the rest of the protocol is simulated, and in particular independent of π. Equation (3.7) still holds after the alteration. □

The number of exponentiations for $\mathcal{P}$ is $11k$ and for $\mathcal{V}$ $9k$, plus a low number of single exponentiations. Most of the exponentiations may be computed quicker.

In order to get the number of exponentiations, note that $\mathcal{P}$ needs to compute Φ_1 in order to execute the Chaum-Pedersen protocol, as well as the cost for the Simple k-shuffle.

For $\mathcal{P}$, $7k$ of the exponentiations can be computed with base g, and we may then employ precomputation techniques to reduce the cost with a factor 0.2, see Chapter 2.7. Next, we can use multiexponentiation to compute Λ_1 and Λ_2 ($3k$ exponentiations), and so we can gain at least a double speed-up here as well, compared to square-and-multiply. This also applies to the verifier's computation in the Simple k-shuffle ($2k$ exponentiations).

Finally, the verifier should choose ρ_i and λ small. This will reduce the cost of computing B_i, Φ_1, Φ_2, R_i and S_i significantly.

We summarise this discussion in a proposition.

Proposition 3.14. *The exponentiation cost for $\mathcal{P}$ is $7c_1k + 3c_2k + c_3k \simeq 3.0k$, where $c_1 \simeq 0.2$ (Lim-Lee-exponentiation), $c_2 < 0.5$ (multi-exponentiation) and $c_3 \simeq 0.1$ (short exponents). The exponentiation cost for $\mathcal{V}$ is $5c_2k+5c_3k+k \simeq 4.0k$.*

3.3 Groth-based solution

This idea is largely following the same lines at the solution based on Neff's articles. However, Groth [19] has improved substantially on Neff's results, which gives us better runtimes.

Another advantage is the possibility to use different commitment schemes to get different properties for the protocol. We will use one that is unconditionally binding, so that we will end up with an unconditionally sound proof. The following commitment scheme is described in [19].

Let p and q be as before, and let g be a generator for G, the order q subgroup of $\mathbb{Z}_p^*$. Select $g_1, \ldots g_N$ and h at random from G, where N is larger than the number of messages we would like to commit to in one batch. In practice, we may want to select exponents at random instead of group elements, so that we may use Lim-Lee exponentiation with respect to base g. Since the following scheme is computationally hiding and statistically binding, it will not be a problem.

To commit to k messages $(m_1, \ldots, m_k)$, select a random number r from $\mathbb{Z}_q$. Then form the $k+1$ tuple $(g_1^{m_1+r}, \ldots, g_k^{m_k+r}, h^r)$.

Protocol 3. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

We require a commitment scheme commit working on the same order q group as the ElGamal cryptosystem, as well as a commitment scheme commit$'$ working over an order Q group. Both schemes will be assumed to be as above.

1. $\mathcal{P}$ selects $d_1, \ldots, d_k, r$ and r_d at random from $\mathbb{Z}_q$ and d_a, r_Y, d_Y, r_1 and r_2 from $\mathbb{Z}_Q$. He then computes

$$\begin{aligned} c &\leftarrow \text{commit}(\pi(1), \ldots, \pi(k); r) \\ c_d &\leftarrow \text{commit}(-d_1, \ldots, -d_k; r_d) \\ D &\leftarrow g^{d_a} \\ Y &\leftarrow g^{r_Y} \prod_{i=1}^{k} z_i^{d_i} \\ C_1 &\leftarrow \text{commit}'(r_Y; r_1) \\ C_2 &\leftarrow \text{commit}'(d_Y; r_2) \end{aligned}$$

   and sends c, c_d, D, Y, C_1 and C_2 to $\mathcal{V}$.

2. The verifier selects coefficients $t_1, \ldots, t_k$ of length ℓ at random, and sends to $\mathcal{P}$.

3. The prover now computes f_i as $t_{\pi(i)} + d_i$ and

$$X \leftarrow g^{d_Y} \left(\prod_{i=1}^{k} x_i^{t_i} \right)^{d_a}$$

   and sends $f_1, \ldots, f_k$ and X to $\mathcal{V}$.

4. $\mathcal{V}$ now selects λ and e of length ℓ_e at random, and sends to $\mathcal{V}$.

5. $\mathcal{P}$ and $\mathcal{V}$ carries out Groth's shuffle for known contents to prove that $\mathcal{P}$ knows π and a randomiser ρ such that

$$c^{\lambda} c_d \, \text{commit}(f_1, \ldots, f_k; 0) = \text{commit}(\lambda\pi(1) + t_{\pi(1)}, \ldots \lambda\pi(k) + t_{\pi(k)}; \rho)$$

6. $\mathcal{P}$ computes $f \leftarrow ea + d_a$, $f_Y = er_Y + d_Y$ and $er_1 + r_2$, and sends to $\mathcal{V}$.

7. The verifier accepts if and only if all transmitted values are members of their respective sets or groups and

$$Y^e X \prod_{i=1}^{k} x_i^{f t_i} y_i^{-t_i e} z_i^{f_i e} = g^{f_Y} \tag{3.10}$$

$$h^e D = g^f \tag{3.11}$$

$$C_1^e C_2 = \text{commit}'(f_Y; z_Y) \tag{3.12}$$

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

Prover		Verifier
$r, r_d, d_1, \ldots, d_k \xleftarrow{r} \mathbb{Z}_q$		
$c \leftarrow \text{commit}(\pi(1), \ldots, \pi(k); r)$		
$c_d \leftarrow \text{commit}(-d_1, \ldots, -d_k; r_d)$		
$d_a, r_Y, d_Y, r_1, r_2 \xleftarrow{r} \mathbb{Z}_Q$		
$D \leftarrow g^{d_a}$		
$Y \leftarrow g^{r_Y} \prod_{i=1}^{k} z_i^{-d_i}$		
$C_1 \leftarrow \text{commit}'(r_Y; r_1)$		
$C_2 \leftarrow \text{commit}'(d_Y; r_2)$		
	$\xrightarrow{c, c_d, D, Y, C_1, C_2}$	$t_i \xleftarrow{r} \{0,1\}^{\ell_e}$
	$\xleftarrow{\{t_i\}}$	
$f_i \leftarrow t_{\pi(i)} + d_i$		
$X \leftarrow g^{d_Y} \left(\prod_{i=1}^{k} x_i^{-t_i} \right)^{d_a}$	$\xrightarrow{\{f_i\}, X}$	$\lambda, e \xleftarrow{r} \{0,1\}^{\ell_e}$
	$\xleftarrow{\lambda, e}$	
	$\xleftarrow{\pi, \rho \text{ s.t. } c^\lambda c_d \, \text{commit}(f_1, \ldots, f_k; 0) = \text{commit}(\lambda\pi(i) + t_{\pi(i)}; \rho)}$	
$f \leftarrow ea + d_a$		
$f_Y \leftarrow e r_Y + d_Y$		
$z_Y \leftarrow e r_1 + r_2$		Check group membership
	$\xrightarrow{f, f_Y, z_Y}$	$Y^e X \prod_{i=1}^{k} x_i^{f t_i} y_i^{-t_i e} z_i^{f_i e} \stackrel{?}{=} g^{f_Y}$
		$h^e D \stackrel{?}{=} g^f$
		$C_1^e C_2 \stackrel{?}{=} \text{commit}'(f_Y; z_Y)$

Figure 3.4: Protocol 3.

As before, the properties of this protocol are given by the following propositions.

Proposition 3.15 (Completeness)**.** *Protocol 3 is complete.*

Proof. The shuffle of known content is complete. It is easy to verify that the remaining equations hold. □

Proposition 3.16 (Soundness)**.** *Protocol 3 is unconditionally sound.*

Proof. Since the protocol in Figure 3.2 is unconditionally sound, there exists a permutation π and a number ρ such that $c^\lambda c_d g_i^{f_i} = g_i^{\lambda\pi(i)+t_{\pi(i)}+\rho}$. Assume that $c = \text{commit}(c_1, \ldots, c_k, r)$ and $c_d = \text{commit}(-d_1, \ldots, -d_k, r_d)$ for some c_i and d_i. Then it must hold that

$$\lambda c_i - d_i + f_i + r + r_d = \lambda\pi(i) + t_{\pi(i)} + \rho.$$

and $\rho = r + r_d$, by the last component of the commitments, so we can cancel on both sides.

Since λ is chosen by $\mathcal{V}$ after all the commitments have been made, it holds except with probability $2^{-\ell}$ that $c_i = \pi(i)$ and $f_i = t_{\pi(i)} + d_i$.

Also, since $h^e D = g^f$ and $C_1^e C_2 = \text{commit}'(f_Y; z_Y)$, then there must exist a d_a such that $f = ae + d_a$, and there must exist r_Y, d_Y, r_1 and r_2 such that $f_Y = er_Y + d_Y$ and $z_Y = er_1 + r_2$.

We can then rewrite equation (3.10) into two parts: One that depends on the exponent e, and one that doesn't.

$$\begin{aligned}
1 &= g^{-f_Y} Y^e X \prod_{i=1}^{k} x_i^{ft_i} y_i^{-t_i e} z_i^{f_i e} \\
&= g^{-er_Y - d_Y} Y^e X \prod_{i=1}^{k} x_i^{(ae+d_a)t_i} y_i^{-t_i e} z_i^{(t_{\pi(i)}+d_i)e} \\
&= g^{-er_Y - d_Y} Y^e X \prod_{i=1}^{k} x_i^{(ae+d_a)t_i} \left(x_i^a m_i\right)^{-t_i e} z_i^{(t_{\pi(i)}+d_i)e} \\
&= \left(Y g^{-r_Y} \prod_{i=1}^{k} \left(m_{\pi(i)}^{-1} z_i\right)^{t_{\pi(i)}} z_i^{d_i}\right)^e X g^{-d_Y} \left(\prod_{i=1}^{k} x_i^{t_i}\right)^{d_a}
\end{aligned}$$

All other values above are fixed when e is chosen, so both parts must be 1. The part that doesn't depend on e shows that X is well-formed.

Since e is arbitrary, we must with probability $1 - 2^{-\ell}$ have

$$Y g^{-r_Y} \prod_{i=1}^{k} \left(m_{\pi(i)}^{-1} z_i\right)^{t_{\pi(i)}} z_i^{d_i} = 1$$

This again splits into two parts, where one is dependent of $\{t_i\}$, while the other is fixed after the first round. $\mathcal{P}$ must then have guessed $\{t_i\}$ such that a special group element was hit, with probability $\frac{1}{q}$.

$$Yg^{-r_Y}\prod_{i=1}^{k} z_i^{d_i} = 1$$
$$\prod_{i=1}^{k}\left(m_{\pi(i)}^{-1} z_i\right)^{t_{\pi(i)}} = 1$$

which implies that Y is well-formed, and that $z_i = m_{\pi(i)}$. The probability of $\mathcal{P}^*$ winning is then at most

$$\frac{k+1}{2^\ell} + \frac{2}{2^\ell} + \frac{1}{q} < \frac{k+4}{2^\ell}.$$

□

Proposition 3.17 (Zero knowledge)**.** *Protocol 3 is special honest verifier zero knowledge.*

Proof. We create two simulators, where the second is without access to the prover's private input. These are the same simulators as Groth uses to sketch a proof of Theorem 3 in [19].

Simulator I

1. Simulate the shuffle of known contents.
2. Select $f_1, \ldots, f_k, f, f_Y, z_Y$ and X at random.
3. Compute c, c_d and C_1 as in the real protocol.
4. Compute D, Y and C_2 such that all equations hold.

Simulator II

1. Simulate the shuffle of known contents.
2. Select $f_1, \ldots, f_k, f, f_Y, z_Y$ at random.
3. Compute c, c_d and C_1 as commitments to 0.
4. Select Y at random.
5. Compute D, C_2 and X such that all equations hold.

The result follows by the following two claims.

Symbol	SKC	c	c_d	D	Y	C_1	C_2	$\{t_i\}$	$\{f_i\}$	X	λ	e	f	f_Y	z_Y
Real									real				real		
Sim I	sim	real	real	sim	sim	real	sim	r	r	r	r	r	r	r	r
Sim II	"	sim	sim	"	"	sim	"	r	"	"	r	r	"	"	"

Table 3.2: Elements in the real and simulated transcripts. The symbol r denotes that the value is selected at random. "Sim" means that the value is computed based on the formulas that need to hold, or selected in a particular way.

Claim 3.18. *The transcript of Simulator I is indistinguishable from the real transcript.*

Proof. The shuffle of known contents is HVZK, so we can safely simulate its transcript. Since f, f_Y and z_Y are computed based on random values, these distributions are indistinguishable from random f, f_Y and z_Y. Now we choose a random group element X. If an adversary is able to distinguish a real and random X, he will also be able to solve DDH. Since f now is random, we can choose D as $h^{-e}g^f$, and Y and C_2 similarly. Hence the distributions of real and forged D, Y, and C_2 are indistinguishable. □

Claim 3.19. *The transcript of Simulator II is indistinguishable from the transcript of Simulator I.*

Proof. Both c and c_d can be simulated since the commitment scheme is hiding and the shuffle is simulated. It is easy to compute a convincing C_1. □

The above simulators do not depend on the challenges from $\mathcal{V}$, hence we have special HVZK. □

As before, several exponentiations can be done with optimisations.

Proposition 3.20. *The exponentiation cost for $\mathcal{P}$ is $5c_1k+c_2k+c_3k \simeq 1.6k$, where $c_1 \simeq 0.2$ (Lim-Lee exponentiation), $c_2 < 0.5$ (multi-exponentiation) and $c_3 \simeq 0.1$ (short exponents). The exponentiation cost for $\mathcal{V}$ is $2k + c_2k + 2c_3k \simeq 2.7k$.*

Note that the zero knowledge property fails to hold if $\mathcal{V}$ knows the logarithms of the commitment key.

A lighter commitment scheme could occasionally be used to reduce communication costs. For example, it could be sufficient to only use a computationally binding scheme for $\{d_i\}$. This is impossible due to the computations later in the protocol.

3.4 A better Groth-based solution

The solution presented now also builds on Groth's shuffle of known content, but with a more direct approach. The idea is that the verifier computes the permuted decryption using commitments from the prover, and then compares with the claimed decryption.

In the first variant, we will use the following commitment scheme. Assume g is the generator from the ElGamal scheme. Select a basis $\{x_1, \ldots, x_k\}$ of group

elements. The commitment key is then $(g, x_1, \ldots, x_k)$. In order to commit to a permutation π on a set $\{t_1, \ldots, t_k\}$ of integers, one computes

$$c \leftarrow g^r \prod_{i=1}^{k} x_i^{t_{\pi(i)}}$$

where r is a random number. This scheme is similar to the one used by [19]. They require that the basis consists of random generators of G. The commitment scheme is perfectly hiding and computationally binding.

We will use the commitment scheme somewhat off the standard approach, namely with the two specific keys $(g, x_1, \ldots x_k)$ and $(g, y_1, \ldots, y_k)$ where (x_i, y_i) is a ciphertext. We will assume that the cipertexts are random, and that both components are group members. In particular, this means that the encrypted messages must be group elements. We denote the variants as commit_x and commit_y respectively.

Protocol 4. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

1. $\mathcal{V}$ selects t_1 to t_k at random, and sends to $\mathcal{P}$.
2. $\mathcal{P}$ selects r_0 at random, computes the following commitments

$$\begin{aligned} X_0 &\leftarrow \text{commit}_x(-t_{\pi^{-1}(1)}, \ldots, -t_{\pi^{-1}(k)}; -r_0) \\ X &\leftarrow X_0^a \\ Y_0 &\leftarrow \text{commit}_y(t_{\pi^{-1}(1)}, \ldots, t_{\pi^{-1}(k)}; ar_0) \end{aligned}$$

   and sends X_0, X and Y_0 to $\mathcal{V}$.
3. $\mathcal{P}$ and $\mathcal{V}$ carries out Groth's shuffle of known content (SKC) for X_0 and Y_0, and Chaum-Pedersen for $X = X_0^a$.
4. $\mathcal{V}$ accepts if and only if the protocols in the previous step are accepted and $XY_0 = \prod_{i=1}^{k} z_i^{t_i}$.

Proposition 3.21 (Completeness)**.** *Protocol 4 is complete.*

Proof. This is easily seen with the following computation,

$$\begin{aligned} XY_0 &= \left(g^{-r_0} \prod_{i=1}^{k} x_i^{-t_{\pi^{-1}(i)}}\right)^a \left(g^{ar_0} \prod_{i=1}^{k} y_i^{t_{\pi^{-1}(i)}}\right) \\ &= \prod_{i=1}^{k} \left(x_{\pi(i)}^{-a} y_{\pi(i)}\right)^{t_i} = \prod_{i=1}^{k} m_{\pi(i)}^{t_i} = \prod_{i=1}^{k} z_i^{t_i} \end{aligned}$$

□

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

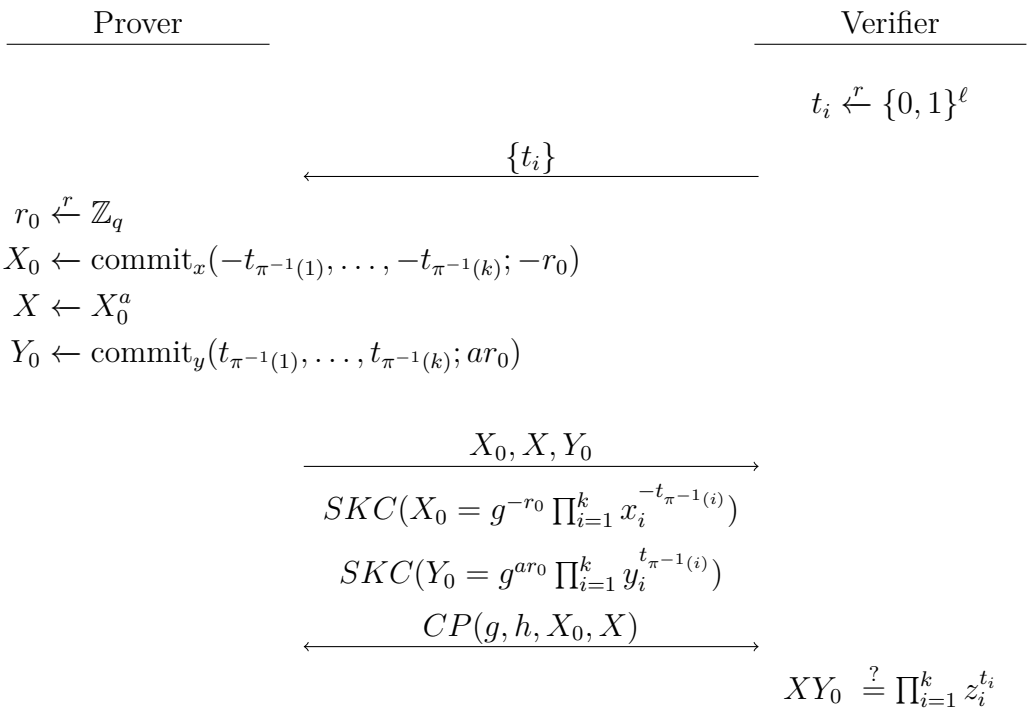

Figure 3.5: Protocol 5.

Proposition 3.22 (Soundness)**.** *Protocol 4 is computationally sound.*

Proof. The commitment scheme is perfectly hiding and computationally binding, and hence the shuffle of known content is computationally sound. Also, the Chaum-Pedersen protocol is sound. This implies that all commitments in step 2 above are well-formed, and hence that the protocol itself is sound. □

Proposition 3.23 (Zero knowledge)**.** *Protocol 4 is special honest verifier zero knowledge.*

Proof. The protocol cannot be statistical HVZK, due to inclusion the Chaum-Pedersen protocol. The following simulator shows that this is SHVZK.

1. Select X_0 and X at random.
2. Compute Y_0 as $\prod_{i=1}^{k} z_i^{t_i} x^{-1}$
3. Simulate the subprotocols.

It is clear that this distribution is computationally indistinguishable from the real distribution. □

Proposition 3.24 (Runtime)**.** *Protocol 4 requires* $8k$ *exponentiations for* $\mathcal{P}$ *and* $5k$ *exponentiations for* $\mathcal{V}$*, all of which can be computed using multi-exponentiations techniques, giving a cost of approximately* $4k$ *and* $3.5k$ *respectively.*

We would like to achieve unconditional soundness. This can be achieved by using a similar commitment scheme as in Chapter 3.3. As above, we use the components of the cipertexts as bases. In addition, we select two group elements x_0 and y_0 at random. We then define the commitment scheme commit_x as

$$\text{commit}_x(t_{\pi(i)}, \ldots, t_{\pi(k)}; r) = (x_1^{t_{\pi(1)}+r}, \ldots, x_k^{t_{\pi(k)}+r}, x_0^r).$$

The scheme commit_y is defined similarly.

Protocol 5. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

1. $\mathcal{V}$ selects t_1 to t_k at random, and sends to $\mathcal{P}$.
2. $\mathcal{P}$ selects r_0 at random, computes the following commitments

$$\begin{aligned} X_0 &\leftarrow \text{commit}_x(-t_{\pi^{-1}(1)}, \ldots, -t_{\pi^{-1}(k)}; -r_0) \\ X &\leftarrow \left(\prod_{i=1}^{k} (X_0)_i\right)^a \\ Y_0 &\leftarrow \text{commit}_y(t_{\pi^{-1}(1)}, \ldots, t_{\pi^{-1}(k)}; r_0) \\ R &\leftarrow \left(\prod_{i=1}^{k} z_i\right)^{r_0} \end{aligned}$$

and sends X_0, X, Y_0 and R to $\mathcal{V}$.

3. $\mathcal{P}$ and $\mathcal{V}$ carries out Groth's shuffle of known content (SKC) for X_0 and Y_0, and Chaum-Pedersen for $X = X_0^a$ and $R = \left(\prod_{i=1}^k z_i\right)^{r_0}$.

4. $\mathcal{V}$ accepts if and only if the protocols in the previous step are accepted and

$$XR^{-1}\prod_{i=1}^{k}(Y_0)_i = \prod_{i=1}^{k} z_i^{t_i} \tag{3.13}$$

Proposition 3.25 (Completeness)**.** *Protocol 5 is complete.*

Proof. Again, this is the result of an easy computation.

$$\begin{aligned} XR^{-1}\prod_{i=1}^{k}(Y_0)_i &= \left(\prod_{i=1}^{k} x_i^{-t_{\pi^{-1}(i)}-r_0}\right)^a \left(\prod_{i=1}^{k} z_i\right)^{-r_0} \prod_{i=1}^{k} y_i^{t_{\pi^{-1}(i)}+r_0} \\ &= \left(\prod_{i=1}^{k} x_i^{-1} y_i\right)^{r_0} \left(\prod_{i=1}^{k} z_i\right)^{-r_0} \prod_{i=1}^{k} \left(x_i^{-1} y_i\right)^{t_{\pi^{-1}(i)}} \\ &= \prod_{i=1}^{k} m_{\pi(i)}^{t_i} = \prod_{i=1}^{k} z_i^{t_i} \end{aligned}$$

□

Recall that we assume that all cipertexts are random. This assumption is reasonable as long as neither $\mathcal{P}$ nor $\mathcal{V}$ are able to influence the selection of these.

Proposition 3.26 (Soundness)**.** *Protocol 5 is unconditionally sound.*

Proof. The commitment scheme is perfectly binding, and hence the shuffle of known content is unconditionally sound. Also, the Chaum-Pedersen protocol is unconditionally sound. This implies that all commitments in step 2 above are well-formed, and hence that the protocol itself is sound, bounded by $\frac{k+2}{2^\ell}$. □

Proposition 3.27 (Zero knowledge)**.** *Protocol 5 is special honest verifier zero knowledge.*

Proof. It is easy to show that this is SHVZK, using the following simulator.

1. Simulate all subprotocols

2. Select X_0, X and Y_0 at random.

3. Select R in accordance with (3.13).

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

Prover		Verifier
		$t_i \xleftarrow{r} \{0,1\}^\ell$
	$\xleftarrow{\{t_i\}}$	
$r_0 \xleftarrow{r} \mathbb{Z}_q$		
$X_0 \leftarrow \text{commit}_x(-t_{\pi^{-1}(1)}, \ldots, -t_{\pi^{-1}(k)}; -r_0)$		
$X \leftarrow \left(\prod_{i=1}^{k} (X_0)_i\right)^a$		
$Y_0 \leftarrow \text{commit}_y(t_{\pi^{-1}(1)}, \ldots, t_{\pi^{-1}(k)}; r_0)$		
$R \leftarrow \left(\prod_{i=1}^{k} z_i\right)^{r_0}$		
	$\xrightarrow{X_0, X, Y_0, R}$	
	$SKC(X_0 = g^{-r_0} \prod_{i=1}^{k} x_i^{-t_{\pi^{-1}(i)}})$	
	$SKC(Y_0 = g^{ar_0} \prod_{i=1}^{k} y_i^{t_{\pi^{-1}(i)}})$	
	$CP(g, h, X_0, X)$	
	$\xleftarrow{CP(x_0, (X_0)_{k+1}, \prod_{i=1}^{k} z_i, R)}$	
		$XR^{-1} \prod_{i=1}^{k} (Y_0)_i \stackrel{?}{=} \prod_{i=1}^{k} z_i^{t_i}$

Figure 3.6: Protocol 5.

It is clear that this distribution is computationally indistinguishable from the real distribution, since R looks random, and X_0 and Y_0 are ElGamal encryptions. □

Proposition 3.28 (Runtime)**.** *Protocol 5 requires $8k$ exponentiations for $\mathcal{P}$ and $5k$ exponentiations for $\mathcal{V}$. No elementary exponentiations techniques can be used to compute the commitments.*

The runtime can be improved by running one instance of the shuffle instead of two. Since the exponents are the same, we can instead prove that $x_0y_0 = \text{commit}_{xy}(x_oy_0 = g^{-r_0+ar_0}\prod_{i=1}^{k}\left(x_i^{-1}y_i\right)^{t_{\pi^{-1}(i)}})$. This does not affect the security, but reduces the runtime to $5k$ and $3k$.

The same idea can also be applied to the previous protocol, giving a cost of $2.5k$ and $1.5k$.

3.4.1 Planted ciphertexts

Assume that an attacker has been able to choose a number of the original ciphertexts. It is clear that this will not affect the soundness of the proof for Protocol 5. This is due to the fact that the commitment scheme is unconditionally binding.

However, if the scheme is just computationally binding, it will be easy to create a new opening if $\mathcal{P}^*$ have created the ciphertexts. One can avoid this by using a hash function, and use the hashes of the ciphertexts as basis for the commitment schemes. The security argument is then moved to the random oracle model, since we need to assume that the hashes are random.

Now assume that $\mathcal{V}$ has been able to insert s ciphertexts. If we rearrange the tuple, we have a new ElGamal encryption, and we can assume that the chosen generators are at the front. We then get

$$\begin{aligned}
&\left((x_0y_0)^r, (x_1y_1)^{t_{\pi(1)}+r}, \ldots, (x_sy_s)^{t_{\pi(s)}+r}, (x_{s+1}y_{s+1})^{t_{\pi(s+1)}+r}, \ldots, (x_ky_k)^{t_{\pi(k)}+r}\right)\\
&= \left(\alpha_0^r, g^{r_1t_{\pi(1)}}g^{r_1r}, \ldots, g^{r_st_{\pi(s)}}g^{r_sr}, (\alpha_{s+1})^{t_{\pi(s+1)}+r}, \ldots, \alpha_k^{t_{\pi(k)}+r}\right)\\
&= \left(\alpha_0^r, (g^{t_{\pi(1)}}g^r)^{r_1}, \ldots, (g^{t_{\pi(s)}}g^r)^{r_s}, (\alpha_{s+1})^{t_{\pi(s+1)}+r}, \ldots, \alpha_k^{t_{\pi(k)}+r}\right)
\end{aligned}$$

where α_i is a random generator and r_i is an exponent chosen by $\mathcal{V}$. Now, we are only interested in the components 2 to $s+1$, and we take the respective roots, getting

$$\left(g^{t_{\pi(1)}}g^r, \ldots, g^{t_{\pi(s)}}g^r\right).$$

We now divide each component with the first to cancel out g^r, and by insertion, we can easily find the part of π that works on the ciphertexts chosen by $\mathcal{V}$.

However, we are unable to compute r and the part of π working on the rest of the shuffle, so $\mathcal{V}$ does not get any new information, due to the zero knowledge property of the protocol.

4 Permutation matrices

The idea of this chapter is to use a permutation matrix (A_{ij}) over $\mathbb{Z}_q$, defined as $A_{ij} = 1 \pmod q$ when $\pi(i) = j$ and 0 otherwise, where π is a permutation and q is a prime [11]. There has been built a number of protocols, each using a theorem stating that A is a permutation matrix if and only if certain conditions hold. This is the other main paradigm for shuffling.

There are many protocols using permutation matrices as a basis. The overarching idea is to commit to the matrix, and then prove that the commitment is well formed. Finally, one prove that the matrix has been used to produce a specific vector [25].

The difference between the various approaches is then to find a theorem of the form "A is a permutation matrix if and only if (some condition)". Some such conditions are possible to prove efficiently in zero-knowledge. Each new protocol is based on a different set of conditions, so there are few common primitives, apart from Chaum-Pedersen.

We have used works by Furukawa [12, 11], Peng, Dawson and Bao [30] and Terelius and Wikström [33]

4.1 Existing verifiable shuffles

We start by describing the original work of Furukawa et al. They are able to create a 3-round computationally sound and honest verifier zero knowledge protocol. The protocol requires about $15k$ exponentiations in total.

The main idea comes from the following theorem

Theorem 4.1 (Theorem 1, [12] and Theorem 2, [11])**.** *Let*

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$
$$\delta_{ijg} = \delta_{ij}\delta_{jg}$$

and let q be a prime. A $k \times k$ matrix A is a permutation matrix over $\mathbb{Z}_q$ if and

only if

$$\sum_{h=1}^{k} A_{hi}A_{hj}A_{hg} = \delta_{ijg}$$
$$\sum_{h=1}^{k} A_{hi}A_{hj} = \delta_{ij}$$

for all i, j and g.

Furthermore, if $q \equiv 2 \pmod 3$, then A is a permutation matrix if and only if only the first condition holds.

In the second part of the article, he customises the protocol for decrypting mixnets, i.e. where each computer has a share in the decryption key, and both shuffles and partly decrypts, *shuffle-decryption*. The exponentiation count is reduced to $14k$, but on the expense of security: Although the shuffle-decryption algorithm hides the permutation for almost all cases, it is not guaranteed to be zero knowledge. Indeed, Furukawa notes that his approach *cannot* be zero-knowledge.

This solution is directly applicable to our problem by letting the mix-net consist of a single computer. This does not alter the runtime, and as with the solution based on Neff's work, it is also inefficient.

There has been done further work based on Furukawa's idea. In a 2010 paper, a four-move honest verifier zero knowledge scheme for shuffling was proposed [30]. The exponentiation *cost* for both prover and verifier was given to be $3k$. However, it is only computationally sound. The computational assumption is given in the following definition.

Definition 4.2 (Definition 1, [30])**.** A logarithm relation of $m_1, \ldots, m_n$ all in G, is found if non-negative integers $l_1, \ldots, l_n$, not all zero (modulo q), are found such that $\prod_{i=1}^{n} m_i^{l_i} \equiv 1 \pmod p$. *Multiple discrete logarithm assumption on a polynomial party with respect $m_1, \ldots, m_n$* states that this party cannot find a logarithm relation of $m_1, \ldots, m_n$ with a non-negligible probability.

The soundness is then based on the following theorem.

Theorem 4.3 (Theorem 2, [30])**.** *If the verifier chooses his challenges s_i randomly and the shuffling node can find integers t_i for $i = 1, \ldots, k$ in polynomial time to satisfy*

$$\prod_{i=1}^{k} \mathcal{D}(c_i)^{s_i} = \prod_{i=1}^{k} \mathcal{D}(c'_i)^{t_i}$$
$$\prod_{i=1}^{k} \mathcal{D}(c_i)^{s_i^2} = \prod_{i=1}^{k} \mathcal{D}(c'_i)^{t_i^2}$$

with a non-negligible probability, then there exists an $k \times k$ permutation matrix M such that $(\mathcal{D}(c'_1), \ldots, \mathcal{D}(c'_k))^M = (\mathcal{D}(c_1), \ldots, \mathcal{D}(c_k))$ under multiple discrete logarithm assumption on the shuffling node with respect to $\mathcal{D}(c_1), \ldots, \mathcal{D}(c_k)$.

Also in 2010, Terelius and Wikström [33] published an article in which they present a basic protocol for proving that a matrix is in fact a permutation matrix. The protocol uses a Pedersen-like commitment scheme. Let $g, g_1, \ldots, g_k$ all be generators of the group G. A vector v in $\mathbb{Z}_q^k$ is committed to by computing

$$\text{commit}(v; s) = g^s \prod_{i=1}^{k} g_i^{v_i}$$

where s is a random number from $\mathbb{Z}_q$.

In order to commit to a matrix, one simply commits column-wise, so that if $M = (m_{ij})$ is a matrix and s is a vector of random numbers, then $\text{commit}(M; s)$ is the vector $(\text{commit}((m_{i,1}); s_1), \ldots, \text{commit}((m_{i,k}); s_k))$.

If v, u are vectors, let $\langle v, u \rangle$ be the ordinary inner product, and let $v^u = \prod_{i=1}^{k} v_i^{u_i}$. Note the useful identity

$$\text{commit}(M, s)^e = \text{commit}(Me, \langle s, e \rangle).$$

Theorem 4.4 (Theorem 1, [33])**.** *Let M be an $k \times k$ matrix over $\mathbb{Z}_q$ and x a vector of k independent variables. Then M is a permutation matrix if and only if*

$$\prod_{i=1}^{k} \langle m_i, x \rangle = \prod_{i=1}^{k} x_i$$

and $M1 = 1$.

The proof is straightforward, since each inner product will pick one distinct x_j for each i, and conversely, since $\mathbb{Z}_q[x]$ is a unique factorisation domain. The last condition guarantees that all non-zero elements in the matrix must be 1.

A protocol based on this theorem is then presented. We have not been able to make an estimate of the exponentiation count, although it seems to be low. Later, we will write $TW1(a = \text{commit}(M; s))$ when we use their first protocol to show that a is a commitment to a permutation matrix.

The protocol is a proof of knowledge of either the permutation π or a different opening to the commitment. Since the commitment scheme is computationally binding and statistically hiding, the protocol itself is computationally sound. It is also honest verifier zero-knowledge.

4.2 Attempted adaption of Peng, Dawson, Bao

We spent quite a while trying to adapt [30] to our use, motivated by their low runtimes. Although we didn't succeed, the work is included for completeness, and so that others may do a better attempt.

In order to achieve shuffled decryption, we need to have $\mathcal{V}$ compute a linear combination of the claimed plaintexts. $\mathcal{P}$ must also prove knowledge of $\{t_i\}$ such that

$$\prod_{i=1}^{k} D(c_i)^{s_i} = \prod_{i=1}^{k} z_i^{t_i}$$
$$\prod_{i=1}^{k} D(c_i)^{s_i^2} = \prod_{i=1}^{k} z_i^{t_i^2}$$

holds, where $c_i = (x_i, y_i)$, z_i the claimed decryption and s_i is the challenge from $\mathcal{V}$.

Protocol 6. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

1. $\mathcal{V}$ selects s_i at random from $\{0,1\}^\ell$ and sends to $\mathcal{P}$.

2. $\mathcal{P}$ assigns t_i as $t_{\pi(i)}$, and selects r_i, v_i, u_i, w_i and w_i' at random from $\mathbb{Z}_q$. He then computes

$$z_i' \leftarrow z_i^{t_i} x_i^{r_i}$$
$$\beta_i \leftarrow z_i^{v_i} x_i^{u_i}$$
$$b \leftarrow \left(\prod_{i=1}^{k} x_i\right)^{w_i}$$
$$\beta \leftarrow \left(\prod_{i=1}^{k} x_i\right)^{w_i'} \prod_{i=1}^{k} z_i'^{v_i}$$

   Finally, $z_i', \{\beta_i\}, b$ and β is sent to $\mathcal{V}$.

3. $\mathcal{V}$ sends a random challenge c from $\{0,1\}^\ell$.

4. $\mathcal{P}$ sends the following responses

$$\theta_i \leftarrow u_i + r_i c$$
$$\gamma_i \leftarrow v_i + t_i c$$
$$\delta_i \leftarrow w_i + (a s_i - r_i) c$$
$$\epsilon_i \leftarrow w_i' + (a s_i^2 - r_i t_i) c$$

5. $\mathcal{V}$ accepts if

$$\begin{aligned}\beta_i z_i'^c &\stackrel{?}{=} z_i^{\gamma_i} x_i^{\theta_i} \\ b &\stackrel{?}{=} \prod_{i=1}^{k} x_i^{\delta_i} \left(\prod_{i=1}^{k} z_i'\right)^c \left(\prod_{i=1}^{k} y_i^{-s_i}\right)^c \\ \beta &\stackrel{?}{=} \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i'^{\gamma_i} \left(\prod_{i=1}^{k} y_i^{-s_i^2}\right)^c\end{aligned}$$

Proposition 4.5 (Completeness)**.** *Protocol 6 is complete.*

Proof. Assuming that both $\mathcal{P}$ and $\mathcal{V}$ are honest, we get

$$\beta_i z_i'^c = z_i^{v_i} x_i^{u_i} \left(z_i^{t_i} x_i^{r_i}\right)^c = z_i^{v_i + t_i c} x_i^{u_i + r_i c} = z_i^{\gamma_i} x_i^{\theta_i}$$

$$\begin{aligned}\prod_{i=1}^{k} x_i^{\delta_i} \left(\prod_{i=1}^{k} z_i'\right)^c \left(\prod_{i=1}^{k} y_i^{-s_i}\right)^c &= \prod_{i=1}^{k} x_i^{w_i + (as_i + r_i)c} \left(\prod_{i=1}^{k} z_i^{t_i} x_i^{r_i}\right)^c \left(\prod_{i=1}^{k} y_i^{-s_i}\right)^c \\ &= \prod_{i=1}^{k} x_i^{w_i + (as_i + r_i)c} \prod_{i=1}^{k} \left(x_{\pi(i)}^{-a} y_{\pi(i)}\right)^{t_i c} x_i^{r_i c} \prod_{i=1}^{k} y_i^{-s_i c} \\ &= \prod_{i=1}^{k} x_i^{w_i + (as_i + r_i)c} x_{\pi(i)}^{-at_i c} x_i^{r_i c} \prod_{i=1}^{k} y_{\pi(i)}^{t_i c} y_i^{-s_i c} \\ &= \prod_{i=1}^{k} x_i^{w_i + (as_i + r_i)c} x_i^{-as_i c} x_i^{r_i c} \prod_{i=1}^{k} y_i^{s_i c} y_i^{-s_i c} \\ &= \prod_{i=1}^{k} x_i^{w_i} = b\end{aligned}$$

$$\begin{aligned}\prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i'^{\gamma_i} \left(\prod_{i=1}^{k} y_i^{-s_i^2}\right)^c &= \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} \left(z_i^{t_i} x_i^{r_i}\right)^{\gamma_i} \prod_{i=1}^{k} y_i^{-s_i^2 c} \\ &= \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i^{t_i v_i} x_i^{r_i v_i} z_i^{t_i^2 c} x_i^{r_i t_i c} \prod_{i=1}^{k} y_i^{-s_i^2 c} \\ &= \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i^{t_i v_i} x_i^{r_i v_i} \left(x_{\pi(i)}^{-a} y_{\pi(i)}\right)^{t_i^2 c} x_i^{r_i t_i c} \prod_{i=1}^{k} y_i^{-s_i^2 c} \\ &= \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i^{t_i v_i} x_i^{r_i v_i} x_i^{-as_i^2 c} y_i^{s_i^2 c} x_i^{r_i t_i c} \prod_{i=1}^{k} y_i^{-s_i^2 c} \\ &= \prod_{i=1}^{k} x_i^{\epsilon_i} x_i^{(-as_i^2 + r_i t_i)c} \prod_{i=1}^{k} z_i'^{v_i} \prod_{i=1}^{k} y_i^{s_i^2 c - s_i^2 c} \\ &= \prod_{i=1}^{k} x_i^{w_i'} \prod_{i=1}^{k} z_i'^{v_i} = \beta\end{aligned}$$

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

Prover		Verifier
	$\xleftarrow{\{s_i\}}$	$s_i \xleftarrow{r} \{0,1\}^\ell$
$t_i \leftarrow s_{\pi(i)}$		
$r_i, u_i, v_i, w_i, w_i' \xleftarrow{r} \mathbb{Z}_q$		
$z_i' \leftarrow z_i^{t_i} x_i^{r_i}$		
$\beta_i \leftarrow z_i^{v_i} x_i^{u_i}$		
$b \leftarrow \prod_{i=1}^{k} x_i^{w_i}$		
$\beta \leftarrow \prod_{i=1}^{k} \left(z_i'^{v_i} x_i^{w_i'} \right)$		
	$\xrightarrow{\{z_i'\}, \{\beta_i\}, b, \beta}$	$c \xleftarrow{r} \{0,1\}^\ell$
	$\xleftarrow{c}$	
$\theta_i \leftarrow u_i + r_i c$		
$\gamma_i \leftarrow v_i + t_i c$		
$\delta_i \leftarrow w_i + (a s_i - r_i) c$		
$\epsilon_i \leftarrow w_i' + (a s_i^2 - r_i t_i) c$	$\xrightarrow{\{\theta_i\}, \{\gamma_i\}, \{\delta_i\}, \{\epsilon_i\}}$	
		$\beta_i z_i'^c \stackrel{?}{=} z_i^{\gamma_i} x_i^{\theta_i}$
		$b \stackrel{?}{=} \prod_{i=1}^{k} x_i^{\delta_i} \left(\prod_{i=1}^{k} z_i' \right)^c \left(\prod_{i=1}^{k} y_i^{-s_i} \right)^c$
		$\beta \stackrel{?}{=} \prod_{i=1}^{k} x_i^{\epsilon_i} \prod_{i=1}^{k} z_i'^{\gamma_i} \left(\prod_{i=1}^{k} y_i^{-s_i^2} \right)^c$

Figure 4.1: Protocol 6.

as wished, and so the protocol is complete. □

Proposition 4.6 (Soundness)**.** *Protocol 6 is not sound.*

Proof. If $\mathcal{V}$ accepts, then there must exsist $\{u_i\}, \{r_i\}, \{v_i\}$ and $\{t_i\}$ such that $\gamma_i = v_i + t_i c$ and $\theta_i = u_i + r_i c$, and hence

$$\beta_i = z_i^{v_i} x_i^{u_i} \qquad\qquad z_i' = z_i^{t_i} x_i^{r_i}$$

Also assume that $\delta_i = \delta_i^{(1)} + \delta_i^{(2)} c$ and $\epsilon_i = \epsilon_i^{(1)} + \epsilon_i^{(2)} c$. Since $y_i = x_i^a m_i$, we get

$$b = \prod_{i=1}^{k} x_i^{\delta_i^{(1)}} \qquad\qquad \prod m_i^{s_i} = \prod_{i=1}^{k} z_i^{t_i} \prod_{i=1}^{k} x_i^{\delta_i^{(2)} + r_i - a s_i}$$

$$\beta = \prod_{i=1}^{k} x_i^{\epsilon_i^{(1)}} z_i'^{v_i} \qquad\qquad \prod m_i^{s_i^2} = \prod_{i=1}^{k} z_i^{t_i^2} \prod_{i=1}^{k} x_i^{\epsilon_i^{(2)} + r_i t_i - a s_i^2}$$

By rewinding, we can extract $\{t_i\}$ and a.

1. Run $\mathcal{P}^*$ until he outputs z_i' etc., and save the state. Submit a random c, and run until the end. If we accept the response, rewind and input a new c' until we accept the new response.

2. By the above computations, $\delta_i^{(1)}$ and $\epsilon_i^{(1)}$ must be constant, and hence also $\delta_i^{(2)}$ and $\epsilon_i^{(2)}$. Also, u_i, r_i, v_i and t_i must have been kept constant, so we can compute them based on the responses to c and c' respectively.

3. Since we can extract t_i, it must be a correct shuffle.

However, it is impossible to prove that $\delta_i^{(2)} + r_i - a s_i = \epsilon_i^{(2)} + r_i t_i - a s_i^2 = 0$. Therefore, a cheating prover may modify the decryption with x_i raised to a chosen exponent. □

4.3 Terelius-Wikström-based solution

Recall that we by $TW1(a = \text{commit}(M; s))$ mean the protocol by [33] that proves that a is a commitment to the permutation matrix M.

As we did in Chapter 3.4, let us modify the commitment scheme a bit, so that commit_x means the scheme described above, using $g, x_1, \ldots, x_k$ as key, and similarly commit_y.

Protocol 7. Let $1 \leq i \leq k$. Both $\mathcal{P}$ and $\mathcal{V}$ recieve $\{(x_i, y_i)\} = \{(g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$ as public input. In addition $\mathcal{P}$ knows the decryption key a and a permutation π. $\mathcal{P}$ must convince $\mathcal{V}$ that $z_i = \mathcal{D}(x_{\pi(i)}, y_{\pi(i)}) = m_{\pi(i)}$.

1. $\mathcal{P}$ selects s at random from $\mathbb{Z}_q$, computes

$$X_0 \leftarrow \text{commit}_x(M; s)$$
$$Y_0 \leftarrow \text{commit}_y(M; as)$$
$$X \leftarrow \left(\prod_{i=1}^{k} (X_0)_i\right)^{-a}$$

   and submits X_0, Y_0 and X to $\mathcal{V}$.

2. $\mathcal{P}$ and $\mathcal{V}$ run $TW1$ twice to prove that X_0 and Y_0 are well-formed. They also run Chaum-Pedersen to prove that X is well-formed.

3. $\mathcal{V}$ accepts if $X \prod_{i=1}^{k} (Y_0)_i = \prod_{i=1}^{k} z_i$.

The following propositions are easy to prove, as all properties are inherited from $TW1$.

Proposition 4.7 (Completeness)**.** *Protocol 7 is complete.*

Proof. First note that $TW1$ is complete, and that

$$\text{commit}_x(M; s) = \left(g^{s_1} \prod_{i=1}^{k} x_i^{m_{i,1}}, \ldots, g^{s_k} \prod_{i=1}^{k} x_i^{m_{i,k}}\right) = \left(g^{s_1} x_{\pi^{-1}(1)}, \ldots, g^{s_k} x_{\pi^{-1}(k)}\right)$$
$$\text{commit}_y(M; -as) = \left(g^{as_1} \prod_{i=1}^{k} y_i^{m_{i,1}}, \ldots, g^{as_k} \prod_{i=1}^{k} y_i^{m_{i,k}}\right) = \left(h^{s_1} y_{\pi^{-1}(1)}, \ldots, h^{s_k} y_{\pi^{-1}(k)}\right)$$

Then

$$X = \left(g^{\sum_{i=1}^{k} s_i} \prod_{i=1}^{k} x_i\right)^{-a} = h^{\sum_{i=1}^{k} -s_i} \prod_{i=1}^{k} x_i^{-a} \quad \text{and} \quad \prod_{i=1}^{k} (Y_0)_i = h^{\sum_{i=1}^{k} s_i} \prod_{i=1}^{k} y_i,$$

and so

$$X \prod_{i=1}^{k} (Y_0)_i = h^{\sum_{i=1}^{k} -s_i} \prod_{i=1}^{k} x_i^{-a} h^{\sum_{i=1}^{k} s_i} \prod_{i=1}^{k} y_i = \prod_{i=1}^{k} x_i^{-a} y_i = \prod_{i=1}^{k} z_i$$

□

Proposition 4.8 (Soundness)**.** *Protocol 7 is a proof of knowledge for either π and a, or a different opening to the commitments.*

Proof. Run the knowledge extractor of $TW1$ to produce openings $(M, s), (M, as)$ of the commitment. If M is a permutation matrix, then we have the correct openings [33], and can compute a. Otherwise, we have constructed an oracle for a different opening to the commitments. □

Public input: $\{(x_i, y_i) = (g^{r_i}, h^{r_i} m_i)\}$ and $\{z_i\}$.
Private input to $\mathcal{P}$: π such that $z_i = m_{\pi(i)}$ and a such that $h = g^a$.

Prover		Verifier
$s_i \xleftarrow{r} \mathbb{Z}_q$		
$X_0 \leftarrow \text{commit}_x(M; s)$		
$Y_0 \leftarrow \text{commit}_y(M; as)$		
$X \leftarrow \left(\prod_{i=1}^{k} (X_0)_i\right)^{-a}$		
	$\xrightarrow{X_0, Y_0, X}$	
	$TW1(X_0 = \text{commit}_x(M; s))$	
	$TW1(Y_0 = \text{commit}_y(M; -as))$	
	$CP\left(\log_{\prod_{i=1}^{k} (X_0)_i} X = \log_g h^{-1}\right)$	
	$\xleftarrow{}$	
		$X \prod_{i=1}^{k} (Y_0)_i \stackrel{?}{=} \prod_{i=1}^{k} z_i$

Figure 4.2: Protocol 7.

The discussion on planted ciphertexts from the last section also applies here.

Proposition 4.9 (Zero-knowledge)**.** *Protocol 7 is honest verifier zero-knowledge.*

Proof. $TW1$ and Chaum-Pedersen are HVZK, so we can easily simulate the whole protocol.

1. Let X_0 and Y_0 be random vectors, and create a simulated transcript of $TW1$.
2. Compute X as $\prod_{i=1}^{k} z_i / \prod_{i=1}^{k} (Y_0)_i$, and create a simulated Chaum-Pedersen transcript. □

Let $T_{\mathcal{P}}$ and $T_{\mathcal{V}}$ be the exponentiation cost of $TW1$. The exponentiation count of Protocol 7 is then $2k + 2T_{\mathcal{P}}$ and $2T_{\mathcal{V}}$ for $\mathcal{P}$ and $\mathcal{V}$ respectively. However, we note that the $2k$ exponentiations for the prover are computed in products.

We may also save some time by just running one instance of $TW1$. Let $\odot$ be the pointwise multiplication of two vectors, and let $A = X_0^t \odot Y_0$, where t is a random number from $\mathcal{V}$, raised pointwise. Then A is a commitment to $(M, ts + as)$ with commitment key $\{x_i^t y_i\}$. For the verifier, it will prove that M exists, and has been used to form commitments

$$\text{commit}_{x^t y}(M; s) = \text{commit}_x^t(M; ts') \odot \text{commit}_y(M; s - ts').$$

Since t is random, this is with high probability the only decomposition, and hence are both X_0 and Y_0 well-formed, reducing the cost to $2k$ multiexponentiations + k short + $T_{\mathcal{P}}$ and $T_{\mathcal{V}} + k$ short exponentiations for $\mathcal{P}$ and $\mathcal{V}$.

5 Closing remarks

In this thesis, we have adapted several existing shuffles to shuffled decryption. The most interesting question is whether we have been able to provide an improvement over separate shuffle and decryption. Table 5.1 shows a comparison between our proposed protocols and the protocols they are based upon, plus the cost of verifying the decryption.

Protocol	$\mathcal{P}$	$\mathcal{V}$	$\mathcal{P}$'s cost	$\mathcal{V}$'s cost	Security
[28] + VD	$10k$	$13k$			US/HVZK
Protocol 2	$11k$	$11k$	$3k$	$4k$	US/HVZK
[19] + VD	$7k$	$7k$			CS/HVZK
Protocol 3	$7k$	$5k$	$1.6k$	$2.7k$	US/HVZK
[1] + VD	$(2\log(m)+1)k$	$5k$			CS/HVZK
Protocol 4	$5k$	$3k$	$2.5k$	$1.5k$	CS/HVZK
Protocol 5	$5k$	$3k$	$5k$	$3k$	US/HVZK
[11]	$8k$	$6k$	$1.9k^*$	$2k^*$	CS/CPH
[30] + VD			$3.1k^*$	$3.1k^*$	CS/HVZK
Protocol 7	$3k + T_{\mathcal{P}}$	$k + T_{\mathcal{V}}$	$1.1k + T_{\mathcal{P}}$	$0.1k + T_{\mathcal{V}}$	CS/HVZK

Table 5.1: Comparison of shuffles. VD is short for verifiable decryption, and is assumed to take k extra short exponentiations for both prover and verifier. Numbers marked with * are copied from the respective articles, and may have been computed differently than here. US = Unconditional soundness, CS = Computational soundness, HVZK = Honest verifier zero knowledge, CPH = Complete permutation hiding.

If there was a moral triangle inequality, it would state that proving both the shuffle and the decryption in one protocol ought to be more efficient than using two protocols.

The problem can be studied from three angles.

1. As a more general verifiable decryption. This is the idea of Protocol 4, 5 and 7.

2. As a special case of shuffles, using the negative of the encryption randomiser. However, the exponent cannot easily be extracted, so we need to use the

decryption key instead, and thus modify the protocol. We then need to show that the decryption key was indeed used, and that the decryption of (x_i, y_i) is a permutation of the decryption of $(1, z_i)$. This is the approach of Protocol 6, however not successful.

3. There also exist efficient protocols for shuffle-decryption. Our problem can also be considered as a special case with only one node in the mixnet. This is the basis for Protocol 3. Also [11] reduces to a solution without any modifications.

We believe that the first and third are the best ways to look at the problem. In ordinary shuffles, three things must be proved: That there exists a permutation, that there exists some randomisers, and that the same randomiser has been used for both components. However, in order to prove a decryption, one need to prove that *the* decryption key has been used, which is a stronger requirement. This turned out to be the main problem when adapting [30], as it would either allow the verifier to use the decryption key, or allow the prover to include randomisers – in theory making it possible for a prover to produce any set of messages.

The above argument also suggests that verifiable shuffled decryption should be almost as expensive as first shuffle and then decryption.

The question as to which option is the better remains in limbo as long as the cost of Terelius and Wikström's primitive protocol is unknown. Furukawa [11] is certainly looking good, but we may have been more conservative when estimating the cost of our protocols than they were, so it is again a hard decision. Protocol 4 and 5 may originally look better than Protocol 3, but the latter will probably perform better in practice. The choice may also depend on actual hardware differences between $\mathcal{P}$ and $\mathcal{V}$.

It would be interesting to make a hybrid shuffle. There exist shuffles for known contents, and for completely unknown content. It may be possible to save some work by looking more into the basic shuffles, since we in fact are working with a shuffle of *partly* known contents – the decryption does not contain any additional information.

Finally, the prospect of a hybrid shuffle is very interesting. Shuffles exist for both known and completely unknown content. Given that we are working with a shuffle of *partly* known contents, it may be possible to combine these two to cut costs and increase efficiency. Future research in this area may hence improve the state of the art substantially.

Bibliography

[1] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[2] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[4] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir Heuristic and applications to Helios. In *ASIACRYPT*, pages 626–643. Springer, 2012.

[5] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *STOC*, pages 103–112. ACM, 1988.

[6] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. Fast exponentiation with precomputation (extended abstract). In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer, 1992.

[7] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO '92*, pages 89–105. Springer-Verlag, 1992.

[8] Sherman S.M. Chow, Changshe Ma, and Jian Weng. Zero-knowledge argument for simultaneous discrete logarithms. In MyT. Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 520–529. Springer Berlin Heidelberg, 2010.

[9] Ivan Damgård and Jesper Buus Nielsen. Commitment schemes and zero-knowledge protocols. Available at `https://services.brics.dk/java/courseadmin/CPT/documents/getDocument/ComZK08.pdf?d=30199`. Downloaded 2012-10-13, 2011.

[10] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[11] Jun Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE Transactions*, 88-A(1):172–188, 2005.

[12] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.

[13] Kristian Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010. `http://eprint.iacr.org/`.

[14] Kristian Gjøsteen, George Petrides, and Asgeir Steine. A novel framework for protocol analysis. *Journal of Internet Services and Information Security (JISIS)*, 1(2/3):89–106, 8 2011.

[15] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[17] Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.

[18] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.

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

[20] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11, 2012.

[21] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, June 2012.

[22] Yael Tauman Kalai. *Attacks on the Fiat-Shamir paradigm and program obfuscation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006. AAI0818161.

[23] Chae Hoon Lim. Efficient multi-exponentiation and applications to batch verification of digital signatures. Available at `http://dasan.sejong.ac.kr/~chlim/pub/multi_exp.ps`. Downloaded 2012-11-06, 2000.

[24] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 1994.

[25] Helger Lipmaa and Bingsheng Zhang. A more efficient computationally sound non-interactive zero-knowledge shuffle argument. *IACR Cryptology ePrint Archive*, 2011:394, 2011.

[26] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In B. Preneel, editor, *Advances in Cryptology - AfricaCrypt 2009*, Lecture Notes in Computer Science. Springer-Verlag, June 2009.

[27] 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*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

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

[29] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.

[30] Kun Peng, Ed Dawson, and Feng Bao. Modification and optimisation of a shuffling scheme: stronger security, formal analysis and higher efficiency. *Int. J. Inf. Sec.*, 10(1):33–47, 2011.

[31] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer Berlin / Heidelberg, 1996.

[32] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[33] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.