KTH Numerical Analysis
and Computer Science

On the Security of Mix-Nets and Hierarchical Group Signatures

DOUGLAS WIKSTRÖM

Doctoral Thesis
Stockholm, Sweden 2005

Skolan för datavetenskap och kommunikation
Kungliga Tekniska högskolan
SE-100 44 Stockholm
SWEDEN

TRITA NA 05-38
ISSN 0348-2952
ISRN KTH/NA/R--05/38--SE
ISBN 91-7178-200-1

Tryck: Universitetsservice US AB

Abstract

In this thesis we investigate two separate cryptographic notions: mix-nets and hierarchical group signatures. The former notion was introduced by Chaum (1981). The latter notion is introduced in this thesis, but it generalizes the notion of group signatures which was introduced by Chaum and Heyst (1991).

Numerous proposals for mix-nets are given in the literature, but these are presented with informal security arguments or at best partial proofs. We illustrate the need for a rigorous treatment of the security mix-nets by giving several practical attacks against a construction of Golle et al. (2002). Then we provide the first definition of security of a mix-net in the universally composable security framework (UC-framework) introduced by Canetti (2001). We construct two distinct efficient mix-nets that are provably secure under standard assumptions in the UC-framework against an adversary that corrupts any minority of the mix-servers and any set of senders. The first construction is based on the El Gamal cryptosystem (1985) and is secure against a static adversary, i.e., an adversary that decides which parties to corrupt before the execution of the protocol. This is the first efficient UC-secure mix-net in the literature and the first sender verifiable mix-net that is robust. The second construction is based on the Paillier cryptosystem (1999) and secure against an adaptive adversary, i.e., an adversary that decides which parties to corrupt during the execution of the protocol. This is the first efficient adaptively secure mix-net in any model. An important subprotocol in the above constructions is a zero-knowledge proof of knowledge of a witness that a party behaves as expected. There are two known approaches for constructing such a protocol given by Neff (2002) and Furukawa and Sako (2002) respectively. We present a third independent approach.

We introduce the notion of hierarchical group signatures. This is a generalization of group signatures. There are several group managers, and the signers and group managers are organized in a tree in which the signers are the leaves and the group managers are internal nodes. Given a signature, a group manager learns if it is an ancestor of the signer, and if so to which of its immediate subtrees the signer belongs, but it learns nothing else. Thus, the identity of the signer is revealed in a hierarchical way. We provide a definition of security of hierarchical group signatures and give two provably secure constructions. The first construction is secure under general assumptions. It is impractical and of purely theoretical interest. The second construction is provably secure under standard complexity assumptions and almost practical.

Sammanfattning

Vi undersöker två olika kryptografiska begrepp: mixnät och hierarkiska gruppsignaturer. Det första begreppet introducerades av Chaum (1981). Det senare introduceras i denna avhandling, men det är en generalisering Chaums och Heysts gruppsignaturer (1991).

Det finns många förslag på hur man konstruerar mixnät i literaturen, men de ges endast med informella säkerhetsresonemang eller i bästa fall ofullständiga bevis. Vi illustrerar hur viktigt det är med en rigorös behandling av säkerhet för mixnät genom att presentera flera olika praktiska attacker mot en konstruktion av Golle et al. (2002). Sedan presenterar vi den första definitionen av ett mixnät i Canettis (2001) "universally composable security"-ramverk (UC-ramverket). We konstruerar två effektiva mixnät som är bevisbart säkra i UC-ramverket. Den första konstruktionen är baserad på El Gamals kryptosystem (1985) och är säker mot en statisk angripare, dvs en angripare som bestämmer sig för vilka parter den skall korrumpera innan protokollet börjar köras. Detta är det första kända effektiva och UC-säkra mixnätet. Det är också det första robusta mixnätet som även är avsändarverifierbart. Den andra konstruktionen är baserad på Pailliers kryptosystem (1999) och är säker mot en adaptiv angripare, dvs en angripare som bestämmer sig för vilka parter den skall korrumpera under körningen av protokollet. Det är det första effektiva adaptivt säkra mixnätet i någon modell över huvud taget. Ett viktigt delprotokoll i konstruktionerna är ett nollkunskapsbevis av kunskap att varje deltagare följer protokollbeskrivningen. Det finns två kända metoder för att konstruera ett sådant protokoll beskrivna av Neff (2002) respektive Furukawa och Sako (2002). Vi beskriver ett tredje oberoende alternativ.

Vi inför begreppet hierarkiska gruppsignaturer som är en generalisering av gruppsignaturer. Det finns flera gruppchefer, och signatörerna och gruppcheferna är ordnade i ett träd där signatörerna är löv och gruppcheferna är inre noder. En gruppchef kan given en signatur avgöra till vilket av dess direkta delträd signatören tillhör, men erhåller ingen annan kunskap om vem signatören är. Alltså avslöjas signatörens identitet på ett hierarkiskt vis. Vi ger en formell definition av säkerheten hos hierarkiska gruppsignatursystem och beskriver två bevisbart säkra konstruktioner. Den första konstruktionen är bevisbart säker under allmänna antaganden, men inte praktisk användbar. Den andra konstruktionen är säker under standardantaganden och nästan praktisk användbar.

Acknowledgments

First of all I would like to thank my advisor Johan Håstad. It is hard to imagine a better teacher and guide to the world of science. He has an amazing ability to quickly understand new ideas and suggest improvements, and I can not recall him ever saying that he has not the time to discuss an idea. I would also like to express my deepest gratitude to Nada and Johan for giving me the opportunity to complete my studies at Nada. I thank Mikael Goldmann for his advise during Johan's stay at Princeton.

I am grateful to my co-authors Mårten Trolin and Jens Groth. The work on hierarchical group signatures is joint work with Mårten, and the construction of the adaptively secure mix-net is joint work with Jens.

During my first years of study I worked at the Swedish Institute of Computer Science (SICS). I thank Torsten Ekedahl, Björn Grönvall and Gunnar Sjödin for many discussions on mix-nets. A special thanks also goes to Torsten for his advise on computing greatest common divisors and residue symbols in rings of integers. My work on this topic did not fit in this thesis.

My fellow students Gustaf Hast, Rafael Pass, and Mårten Trolin deserve my gratitude for listening to my unfinished ideas and sharing their own. Mårten and Jakob Nordström also deserves my gratitude for proof reading parts of the thesis. It was a pleasure to share my office with Jesper Fredriksson and Klas Wallenius. Klas also kindly answered many of my questions about writing in English.

I thank Ran Canetti for explaining the universally composable framework to me at an early stage and Ronald Cramer for giving me the chance to present my work at the ECRYPT provable security workshop 2004. I also thank Jun Furukawa, Markus Jakobsson, and Andrew Neff for answering my questions about their respective constructions and for interesting discussions on mix-nets in general.

I gratefully acknowledge that my research was funded by the Swedish Research Institute for Information Technology (SITI), Verket för innovationssystem (VINNOVA), and Vetenskapsrådet (VR).

During my studies I enjoyed the company of numerous other colleagues, both at SICS and at Nada. I thank all of you for making my everyday life interesting!

Finally, I thank Sofie for her unending support and for believing in me those days when I did not.

Contents

List of Figures

Part I

Background

Chapter 1

Introduction

1.1 Cryptographic Protocols

Only 30 years ago, cryptography was still considered a subject almost exclusively of military interest. Since then not only have the applications changed, but also the tools. The original application of cryptography was to allow two parties to communicate secretly over an insecure channel, i.e., no intermediary should be able to deduce the cleartext despite having access to the cryptotext. Today there are complex protocols for key exchange, commerce, auctions, elections, contract signing, etc., each with its own set of security properties.

Diffie and Hellman [60], and Rivest, Shamir and Adleman [132], revolutionized the subject of cryptography when they introduced public key cryptography. Although the notion was first discovered by Merkle [103], his idea was less practical and was not published until later. Interestingly, recently de-classified information suggests that British governmental organizations discovered public key cryptography already in the beginning of the 70's. See [137] for an account of this.

The notion of public key cryptography led to a practical way for two parties to establish a secret channel without any prior agreement. Furthermore, identification schemes and digital signatures were developed to serve as electronic replacements of conventional methods for identification and signing. These techniques are used millions of times each day all over the world to secure and authenticate communication over the Internet. The new tools were embraced by the scientific community and Yao [151] initiated the study of multiparty computation, i.e., the study of complex cryptographic protocols involving multiple parties.

A simple example of a two-party computation problem that was first studied by Blum [24] is the following. Alice and Bob are planning to go to the movies, but they disagree on the choice of movie. If they were able to meet physically they could resolve the problem by tossing a coin and let the outcome decide which movie to see. Suppose now that they can not meet physically and are only talking to each other over the telephone. We also assume that they each have a copy of the phone

book. Can they still toss a "joint" random coin? Yes, this can be done as follows. Alice first tosses a coin. If it comes out heads, she chooses a random name in the phone book such that the name has an even number of letters, and otherwise she chooses a random name that has an odd number of letters. Then she reads the phone number associated with the chosen name to Bob over the phone. Next Bob tosses a coin and tells Alice the outcome. Finally, Alice tells Bob the name she chose. This discloses if Alice got heads or tails on her coin toss. The joint random coin toss is defined to be heads if Alice's and Bob's coins are equal and defined to be tails otherwise. The idea is that Alice can not change her mind after reading the phone number to Bob, and Bob can not figure out if the name associated with the phone number he is given has an even number of letters or not, since the phone book is sorted by name and not by number. Thus, the coin is random as long as one of them tosses its coin randomly no matter what the other party does. There are of course attacks against the protocol. Bob may simply call the number he gets from Alice and ask for the name of the owner, or he may have a phone book sorted by number and find the name directly. However, if the basic idea is translated into a cryptographic protocol the security of the protocol can be formalized and analyzed rigorously, and it can be shown that it suffices if either Alice or Bob follows the protocol to make the coin essentially random.

In general multiparty computation, a group of parties, each having an input, wish to compute a function of their inputs in such a way that nobody is given any information except the output of the function.

An important tool in the construction of such protocols is the notion of zero-knowledge proofs discovered by Goldwasser, Micali, and Rackoff [77]. A zero-knowledge proof is a protocol involving two parties: a prover and a verifier. The goal of the prover is to convince the verifier of some statement without disclosing any knowledge. This should be contrasted with the traditional way of communicating proofs, where the prover explains in detail why the statement is true. One important application of zero-knowledge proofs in cryptography is to allow the parties in a multiparty computation to prove that they follow a predefined protocol. Thus, if some party tries to modify the output this is detected.

We illustrate the notion of a zero-knowledge proof with an example adapted from Goldreich [72]. Consider a large labyrinth contained in a square such that there is a path from an entry in one corner to an exit in another corner. Alice claims that she knows a way from the entry through the labyrinth to the exit, but she is not willing to disclose the path she knows to Bob, since it took her long to find it. Bob distrusts Alice, so Alice must convince Bob that her claim is true without disclosing her path. They solve this problem as follows. Alice carries a flag on a long stick and secretly enters the labyrinth from either the entry or exit. Then she waves her flag to signify to Bob that she is in the labyrinth. Bob then tosses a coin, and shouts to Alice to wave the flag, without disclosing herself, at the entry or the exit of the labyrinth depending on if the coin comes up heads or tails. Now, if Alice does not know any way from the entry to the exit she can not wave the flag as required on both outcomes of the coin tossing and must fail with

probability at least $1/2$. If the experiment is repeated k times the probability that Alice convinces Bob without knowing the secret path is at most $1/2^k$, which even for moderately sized k is very small. Suppose that Ebba drops by Bob, and Bob claims that he knows a path through the labyrinth. Can Bob convince Ebba? Bob could video-tape the experiments carried out with Alice and show Ebba the video, but that would not convince Ebba. Let us see how Bob can make the video without any help from Alice. He turns on the video-camera, flips a coin and secretly enters the labyrinth holding the flag from the entry if the coin comes up heads and from the exit otherwise. Then he waves the flag so that it is captured on video that somebody is in the labyrinth. He flips another coin and shouts to himself to wave the flag at the entry if the coin comes up heads and at the exit otherwise. Note that if the outcomes of the two coins are equal Bob can wave the flag as required, but if they are different he can not. If the latter event occurs, Bob simply steps out of the labyrinth the same way he entered it, rewinds the video tape and tries again. In each attempt he succeeds with probability $1/2$. Thus, he will eventually get a video tape that looks exactly like the video tape he would get from interacting with Alice. In other words, Bob can not convince Ebba of anything that he could not already convince her of without interacting with Alice. We say that Bob gains zero knowledge from the interaction with Alice.

Using zero-knowledge proofs Goldreich, Micali and Wigderson [74], Ben-Or, Goldwasser, Wigderson [20] and Chaum, Crépeau and Damgård [46] showed that any function can be securely computed without disclosing anything but the output of the function. The simple examples above captures the spirit of how cryptographers construct and reason about protocols. A definition of security of general function evaluation was sketched in [74], and was formally defined in Goldwasser and Levin [75], Micali and Rogaway [106], and Beaver [13]. These security frameworks have been refined since, and many variations have been investigated in numerous papers.

The constructions [74, 20, 46] mentioned above are very general, but they do not give practical protocols. Thus, alongside the development of general theory, researchers have been investigating more practical solutions to specific problems. In this thesis we consider two such problems: mix-nets and hierarchical group signatures.

1.2 Provable Security

At this point the reader may rightfully ask in what sense one proves the security of a complex cryptographic protocol. Although it is often easy to give informal requirements on cryptographic primitives, e.g., that a cryptosystem should hide the encrypted message, it is surprisingly difficult to give a correct and rigorous definition of security.

We must define what a protocol is. A natural model is to assume that each party is represented by a Turing machine. Thus, a protocol is simply a list of

Turing machines.

We must also model an adversary. The adversary is a hypothetical party that tries to extract information or make the protocol diverge from normal execution. We must decide which parties are controlled by the adversary. When a party is controlled by the adversary, we say that it is corrupted. We must also decide when an adversary corrupts a party: is it done before or during the execution of the protocol? The first type of adversary is called a *static* adversary and the second type an *adaptive* adversary. Furthermore, if an adaptive adversary corrupts a party during execution, is it given the history of all computations already carried out by the party, or are any such trace erased during execution? We say that the parties execute without or with *erasures* respectively. Since most modern cryptographic protocols can be broken in theory, given unlimited computational resources, we must also limit the computing power of the adversary. A natural way to do this without imposing to many restrictions on the adversary is to assume that it computes in polynomial time in a special parameter called the *security parameter*. Finally, we assume that the adversary controls the communication between all parties, including the parties it has not corrupted. The adversary sees all communicated messages, it decides when a message is delivered, and it can also introduce fake messages.

To define security we must pin down what exactly we expect from the protocol. This is often done by describing an idealized model of the protocol and is best explained by an example. Consider public key encryption. Alice wishes to secretly send a message to Bob. To satisfy Alice's request Bob generates a private key and a corresponding public key of a cryptosystem, and hands Alice the public key. Alice then encrypts her message using Bob's public key and hands the cryptotext to Bob. Bob finally decrypts the cryptotext using his private key. The ideal model of public key encryption corresponds to a trusted party with which both Alice and Bob can communicate secretly. It waits for a message "public key" from Bob, and then hands a message "Bob's public key" to Alice. Alice then hands the message she wishes to transmit directly to the trusted party. Finally, the trusted party forwards the message to Bob and the length of the transmitted message to the adversary. Note that in the ideal model of public key encryption the adversary can not extract any knowledge of the transmitted message except its length, since we assume that the trusted party communicates secretly with Alice and Bob. We say that a public key cryptosystem is secure if for each adversary in the real protocol there is an adversary in the ideal version that has essentially the same advantage. Obviously, this means that no adversary in the real protocol can extract anything else except the length of the encrypted message.

In Chapter 4 we review a mathematical definition of a general security framework corresponding to the above that was introduced by Canetti [41]. We analyze our mix-nets in this framework. We use a different approach to formalize the security of hierarchical group signatures, but the basic idea is the same.

The current understanding of cryptography only rarely allows a mathematical proof of security for a primitive or protocol not relying on any assumptions. What

is normally done instead is to prove security given that some computational assumption holds. Such a proof is sometimes called a security reduction. An example of a computational assumption is the following. Given a random integer that is a product of two large primes it is hard to find its factorization. In Chapter 3 we formalize several such assumptions. In most security proofs we argue by contradiction. We assume that there exists an adversary in the real protocol that has an essentially greater advantage than every adversary in the ideal model. Then we show that this adversary can be used to contradict a given computational assumption. Thus, if the computational assumption is true we have reached a contradiction and we conclude that there exists no such adversary for the real protocol.

The amount of details given in security reductions differ. It is common to prove statements such as "no adversary running in polynomial time in the size of the security parameter κ can break the scheme X unless it breaks the computational assumption Y". Given such a reduction it is mostly straightforward, albeit time consuming, to derive a proof of a statement such as "no adversary running in time $T(\kappa)$ can break the scheme X unless they break computational assumption Y in time $T'(\kappa)$". The latter type of reduction is sometimes referred to as "exact security" to signify that all parameters are explicit. In this thesis we do not provide exact reductions.

At this point the reader should be skeptic. What good is a proof if it only holds in some model of the adversary and under computational assumptions? If attacking the protocol is a computational problem, why can we not assume that it is hard to attack from the beginning?

We consider some answers to these questions. A short answer is that this is the best the cryptographic community can do at the moment, but we think this answer is too pessimistic. In fact, it is hard to imagine an adversary in the real world that is as powerful as the one modeled above. Furthermore, the assumptions about the standard set of problems are based on long experience and experimental evidence. Any notable progress on any of these problems would be a remarkable and very surprising mathematical result. The computational problem corresponding to attacking a new complex protocol on the other hand has not been investigated by many people. Thus, there exists little or none experimental evidence to suggest that it is hard to attack. In fact there is plenty of evidence that suggest that a new protocol without a careful proof of security under standard assumptions is likely to contain serious flaws.

What should be regarded an acceptable proof of security is under constant debate in the cryptographic community. For an account of this we refer the reader to Koblitz and Menezes [98] or Stinson [139].

1.3 Mix-Nets

The main application of mix-nets is for electronic elections. Therefore, before we proceed we review previous and related work on the problem of constructing an

electronic election scheme.

1.3.1 Electronic Elections

The conventional election procedure has been refined over many years to be robust and to ensure correctness of the result. The cast votes are often recounted by different authorities to verify the result, and even though it may be possible to forge a single vote, it is very hard to forge a sufficient number of votes to change the result notably. Furthermore, in many countries the privacy of the voter is preserved by physical means. The intent is not only to guarantee the privacy of the voter, but to protect the voter against coercing and to counter vote buying.

Many questions arise when we try to replace the conventional election procedure with an electronic protocol. In particular, if the objective is to let voters cast their ballot from their personal computer: How do we identify the voters? How do we prevent multiple voting? Can the privacy of the voter be guaranteed, and for how long? Are there ways to coerce voters, or to convince them to sell their vote? Who verifies that the protocol was executed correctly?

Some of these problems seem impossible to solve completely in practice, e.g. it can not be guaranteed that there is no virus on the home PC. Other problems, such as identifying voters, are possible to solve as standalone problems using standard cryptographic techniques. The challenge is to find practical protocols that solve all of these problems at the same time, and to give convincing arguments of their security in reasonable models.

Although there are exceptions, most constructions for electronic voting in the literature fall into one of two categories. Below we discuss the advantages and disadvantages of these approaches.

The first type of construction is based on the use of a homomorphic cryptosystem. Such cryptosystems have the remarkable property that, if several cryptotexts are multiplied, the result is a cryptotext of the product of the cleartexts of the original cryptotexts. This approach has been studied and refined by several authors [50, 22, 21], leading up to the work of Cramer, Gennaro, and Schoenmakers [52]. They give a well analyzed solution based on the El Gamal [71] cryptosystem. Damgård and Jurik [57], and independently Baudron, Fouque, Pointcheval, Poupard, and Stern [12], generalized this approach to use the Paillier [121] cryptosystem. The main advantage of this approach is that counting the votes can be performed very efficiently. A drawback, however, is that the set of candidates must be fixed in advance, i.e., there can be no write-in votes. This problem is particularly severe if the El Gamal [71] cryptosystem is used, in which case only a small number of candidates can be used. If the Paillier [121] cryptosystem is used the number of candidates can be much larger, but write-in votes can still not be cast. Another drawback of the homomorphic approach is that voters must give a relatively complex proof that their ballot contains a vote for a candidate on the candidate list. Finally, the election laws in some countries or states, e.g. the

United States of America, require that the actual ballots are stored. This prohibits the use of the homomorphic approach.

The second approach was introduced by Chaum [45]. His idea is a straightforward translation of the conventional voting procedure. Each voter encrypts its vote and writes it on a bulletin board. Then a mix-net (or anonymous channel in Chaum's terminology) is used to mix and decrypt the cryptotexts in such a way that it is infeasible to find any correspondence between the input cryptotexts and the output cleartexts. Thus, the mix-net ensures the privacy of the voter. The main advantage of this approach is that arbitrary write-in votes can be allowed. A fixed set of candidates can of course also be used.

Interestingly, ideas from the second approach have been used by Hirt and Sako [84] to counter vote selling in an election protocol of the first type. Kiayias and Yung [95] also propose an interesting way to combine the two approaches.

1.3.2 What is a Mix-Net?

In this section we give an informal introduction to mix-nets and sketch two basic constructions. We give some references along the way, but a more complete account of previous and related work on mix-nets is given in Chapter 5.

Suppose a set of senders $S_1, \ldots, S_N$ each have an input m_i, and want to compute the sorted list $(m_{\pi(1)}, \ldots, m_{\pi(N)})$ of messages, but keep the identity of the sender of any particular message m_i secret. A trusted party T can provide the service required by the senders. First it collects all messages. Then it sorts the inputs and outputs the result. This is illustrated in Figure 1.1.

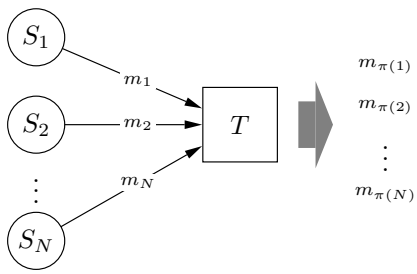

Figure 1.1: The trusted party T receives a message m_i from each sender S_i. Then it outputs the messages but in sorted order $(m_{\pi(1)}, \ldots, m_{\pi(N)})$.

A protocol, i.e., a list of machines $M_1, \ldots, M_k$, that emulates the service of the trusted party as described above is called a *mix-net*, and the parties $M_1, \ldots, M_k$ are referred to as *mix-servers*. Other terms used for the notion of a mix-net include: anonymous channel, mix, or shuffle-network. Similarly a mix-server is called a mix-center or a mixer by some authors.

The usual assumption is that each sender S_i trusts that a certain fraction of the mix-servers $M_1, \ldots, M_k$ are honest. The sender may not want to decide upon any particular parties M_j to trust, but only how many. This type of trust is common in the real world. For example the board of a company consists of a group of individuals. A stockholder does not have to trust each individual to trust the board as a whole.

1.3.3 Constructions

In his paper [45] Chaum suggests the following construction of a mix-net. Each mix-server M_j has a public key pk_j and a private key sk_j of a probabilistic cryptosystem. We write $c = \mathsf{Enc}_{pk}(m)$ for the probabilistic encryption of a message m using the public key pk and $\mathsf{Dec}_{sk}(c) = m$ for the decryption of a cryptotext c using the private key sk. To send a message m_i the sender S_i forms a cryptotext

$$c_{0,i} = \mathsf{Enc}_{pk_1}(\mathsf{Enc}_{pk_2}(\ldots \mathsf{Enc}_{pk_k}(m_i)\ldots))$$

and writes it on a bulletin board. This gives a list

$$L_0 = (c_{0,1}, \ldots, c_{0,N})$$

of cryptotexts from the N senders. Then for $j = 1, \ldots, k$, the jth mix-server M_j decrypts each element $c_{j-1,i}$ by computing $d_{j,i} = \mathsf{Dec}_{sk_j}(c_{j-1,i})$ and forms the list

$$L_j = (c_{j,1}, \ldots, c_{j,N})$$

consisting of the elements $(d_{j,1}, \ldots, d_{j,N})$ in sorted order. By construction the output of M_k is the list of cleartexts, but in sorted order. Note that since each mix-server reorders the cryptotexts the combined reordering is shared by the mix-servers. Thus, if at least one mix-server keeps its part of the reordering secret, essentially no information of the joint reordering is revealed and the anonymity of the senders is ensured.

The cryptosystem used by Chaum is in fact not probabilistic in itself. The encrypted message in each layer of encryption is padded with some random bits. One drawback of the direct implementation of the above method is that the size of the cryptotext $c_{0,i}$ computed by the sender grows linearly with the number of mix-servers k. Another, more important problem is that there seems to be no efficient way to verify that the mix-servers behave honestly. Indeed any of the mix-servers could replace all of the cryptotexts with other cryptotexts of its choice, or refuse to decrypt its input at all. A mix-net is sometimes said to be robust if it outputs the correct result as long as a certain fraction, e.g., the majority of the mix-servers, are honest.

An advantage of Chaum's original scheme is that a sender can verify, by looking at the intermediate lists $L_0, \ldots, L_k$, that its cryptotext is processed correctly by the mix-servers. Furthermore, if any mix-server does not process the cryptotexts properly, the corrupt mix-server can be identified and the sender can easily

prove that the identified mix-server did not follow the protocol. We say that the protocol is *sender verifiable*. Note that sender verifiability does not guarantee the overall correctness or anonymity of a mix-net. Chaum's scheme can be said to be decryption based, since each mix-server decrypts and reorders its input.

Park, Itoh, and Kurosawa [122] show that the problem of linear growth of cryptotexts and proving correctness can be solved by using a homomorphic cryptosystem such as the El Gamal cryptosystem [71] or the Paillier cryptosystem [121]. Such a mix-net is sometimes called length preserving. Suppose we write $\mathsf{Enc}_{pk}(m, r)$ for the probabilistic encryption of a message using randomness r. The cryptosystem is said to be homomorphic if

$$\mathsf{Enc}_{pk}(m_1, r_1) \star \mathsf{Enc}_{pk}(m_2, r_2) = \mathsf{Enc}_{pk}(m_1 m_2, r_1 + r_2) \ .$$

The symbol $\star$ denotes a form of "multiplication" of cryptotexts, and the product $m_1 m_2$ and sum $r_1 + r_2$ are defined over some groups.

Note that if $c = \mathsf{Enc}_{pk}(m, r)$ anybody can, without knowledge of the private key sk, replace the randomness r with fresh randomness by computing $c \star \mathsf{Enc}_{pk}(1, s)$ with random s. This property is known as the re-encryption property. For the two most common cryptosystems, El Gamal [71] and Paillier [121], with this property it is possible to construct a joint public key pk and a corresponding joint private key sk. The joint private key is not known by any individual mix-server. Instead each mix-server is given a secret share, s_j, of the key. Furthermore, there exists a way for the mix-servers to jointly decrypt a cryptotext without revealing their part of the private key. This set-up is called a distributed cryptosystem, since the power to decrypt is distributed among the mix-servers.

Sako and Killian [134] construct a mix-net that is similar to [122] but easier to describe. The sender encrypts its message m_i by simply computing $c_{0,i} = \mathsf{Enc}_{pk}(m_i, r_i)$. Note that the cryptotext no longer grows with the number of mix-servers. Then for $j = 1, \ldots, k$ the jth mix-server M_j computes the cryptotexts $d_{j,i} = c_{j-1,i} E_{pk}(1, r_{j,i})$ and forms the list

$$L_j = (c_{j,1}, \ldots, c_{j,N})$$

consisting of the elements $(d_{j,1}, \ldots, d_{j,N})$ in sorted order. Note that now the output of M_k is no longer the list of cleartexts. Instead the mix-servers jointly decrypt each cryptotext $c_{k,i}$ in the output of M_k using their secret shares s_j to find the cleartexts. We construct a mix-net of this type in Chapter 11. This approach is also illustrated in Figure 1.2.

The advantage of using a homomorphic cryptosystem is that there exists a well defined algebraic structure that can be used to construct an efficient zero-knowledge proof of correct behavior. In other words, a mix-server can prove to the other mix-servers that it formed L_j from L_{j-1} as described above without revealing anything about the randomness $r_{j,i}$ it used. Such a protocol is sometimes called a proof of a shuffle, or a proof of re-encryption-permutation. The first proof of a shuffle we are aware of was given by Sako and Killian [134]. Their construction is based on the

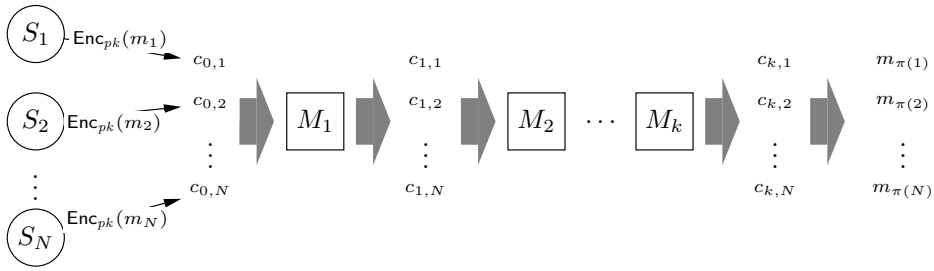

Figure 1.2: The figure illustrates a re-encryption based mix-net. Each sender S_i submits an encrypted message c_{0i}. The cryptotexts $(c_{0,1}, \ldots, c_{0,N})$ are repeatedly re-encrypted and permuted by the mix-servers $M_1, \ldots, M_k$. Finally the mix-servers jointly decrypt the list $(c_{k,1}, \ldots, c_{k,N})$. This gives the list $(m_{\pi(1)}, \ldots, m_{\pi(N)})$ consisting of the cleartexts but in sorted order.

cut-and-choose technique and is impractical. The first efficient constructions of a proof of a shuffle of El Gamal cryptotexts were given by Neff [112] and Furukawa and Sako [70]. Groth [80] generalized Neff's protocol to any homomorphic cryptosystem. In this thesis we give another approach. The idea is that a proof of a shuffle should give a robust mix-net.

In the first type of length preserving mix-net [122] each mix-server partially decrypts, re-encrypts and then permutes the cryptotexts in its input. The advantage of this approach compared to the re-encryption based approach is that less communication is needed among the mix-servers. Recently, efficient proofs of a shuffle have been constructed also for this type of transformation by Neff [113] and Furukawa et al. [69]. An alternative approach is taken by Wikström [147]. His protocol is based on a type of secret sharing scheme, but it is only efficient for a small number of mix-servers.

A drawback of the two main approaches based on a homomorphic cryptosystem is that sender verifiability is lost. In Chapter 8 we show how to construct a mix-net which is sender verifiable and still allows efficient proofs of a shuffle.

The problem of constructing an efficient proof of a shuffle was open for several years. We briefly mention that some authors, e.g. Jakobsson [87, 86], Jakobsson, Juels, and Rivest [91], and Golle et al. [79] propose mix-nets in which global methods are used to ensure robustness, i.e., to avoid costly proofs of a shuffle other methods are used and it is claimed that this guarantees robustness. The constructions in [87, 86, 79] are known to contain serious flaws. The construction in [91] claims to achieve a weak form of anonymity and correctness. As far as we know there are no known attacks against this construction.

1.3.4 On the Need For a Rigorous Treatment

Numerous constructions of mix-nets have been proposed in the literature and have been claimed to be both privacy preserving and robust, e.g. [66, 122, 134, 117, 2, 87, 86]. There is also a large number of successful attacks [59, 108, 126, 125, 107, 146, 5] in the literature. This suggests that a rigorous definition of security and detailed security analyses are needed. To our knowledge the first rigorous definition of security was given recently by Abe and Imai [5], but they do not present any construction that satisfies their definition. The first mix-net construction with a complete security analysis in any model we are aware of was given by Wikström [147]. The definition of an ideal mix-net in this thesis is taken from this paper, but the constructions presented in the thesis are different from that in [147].

In defense of the early works one should keep in mind that the general understanding of the security of protocols and composition of protocols in the cryptographic community has increased dramatically since these works appeared.

1.3.5 Summary of Our Contributions

We give several attacks against mix-nets given by Golle, Zhong, Boneh, Jakobsson, Juels [79], Jakobsson [86], and Jakobsson and Juels [90]. Then we present the first definition of security of a mix-net in the universally composable security framework. We also present two mix-net constructions that are provably secure in the universally composable security framework against adversaries corrupting any minority under standard computational assumptions. The first construction is the first efficient mix-net with a complete security analysis. This construction is based on the El Gamal cryptosystem, but our approach is novel in that each mix-server partially decrypts and permutes its inputs, i.e., no re-encryption is needed. An advantage of our approach is that a sender can verify non-interactively that each mix-server processed its cryptotext correctly, and if this is not the case point out the corrupt mix-server. Our second construction is based on the Paillier cryptosystem and similar in structure to the scheme given by Sako and Killian [134]. This is the first mix-net that is provably secure against an adaptive adversary. We also introduce a new approach to construct an efficient proof of a shuffle.

1.4 Hierarchical Group Signatures

There exists no previous work on hierarchical group signatures, since they are introduced in the thesis, but hierarchical group signatures is a generalization of group signatures which in turn is a generalization of ordinary digital signatures. Therefore, before we proceed we give an informal description of digital signatures and group signatures. We give some references here, but a more detailed and complete account on previous and related work on group signatures is given in Chapter 13.

1.4.1 Digital Signatures

The original concept of digital signatures was discovered by Diffie and Hellman [60], but the first practical implementation was given by Rivest, Shamir, and Adleman [132]. The standard definition of security was introduced by Goldwasser, Micali and Rivest [78].

A digital signature scheme works as follows. A signer generates a private key and a public key and the public key is made public. To sign a message at some later time the signer invokes a signature algorithm on the message and its private key. The result is called a *digital signature* (or simply a *signature*). Anybody holding the public key can verify that the signer, and nobody else, computed the signature by invoking a verification algorithm on the message, the signature, and the public key of the signer. A digital signature scheme is said to be secure if it is infeasible to compute a digital signature that is considered valid by the verification algorithm without access to the private key. Thus, digital signatures is an analogue of ordinary written signatures.

1.4.2 Group Signatures

In this section we give an informal introduction to the group signatures introduced by Chaum and van Heyst [48]. There is a single *group manager* M and several *signers* $S_1, \ldots, S_N$. The signers are also called group members. A signer S_i can compute a signature that reveals nothing about the signer's identity to anybody, except the group manager, except that he is a member of the group. On the other hand the group manager M can, given a signature, always reveal the identity of the signer.

An application of group signatures is anonymous credit cards. The cardholder wishes to preserve his privacy when he pays a merchant for goods, i.e., he is interested in unlinkability of payments. The bank must obviously be able to extract the identity of a cardholder from a payment or at least an identifier for an account, to be able to debit the account. To avoid fraud, the bank, the merchant, and the cardholder all require that a cardholder cannot pay for goods without holding a valid card. To solve the problem using group signatures we let the bank be the group manager and the cardholders be signers. A cardholder signs a transaction and hands it to the merchant. The merchant then hands the signed transaction to the bank, which debits the cardholder and credits the merchant. Since signatures are unlinkable, the merchant learns nothing about the cardholder's identity. The bank on the other hand can always extract the cardholder's identity from a valid signature and debit the correct account.

The most efficient group signature scheme known today that is secure under standard assumptions in the random oracle model is that of Camenisch and Groth [33]. More efficient schemes do exist [27, 34], but they are based on bilinear maps and thus relies on less well-studied assumptions for security.

There are many variations of the notion of group signatures. A group signature scheme is said to have a static group if it is impossible to change the set of signers after they have been given their keys. If it is possible to add and possibly revoke signers the scheme is said to be dynamic.

The first rigorous definition of security for a group signature scheme with static groups were given by Bellare et al. [17]. For dynamic group signatures the first proper definitions were given by Kiayias and Yung [94]. They also proved that a modification of [7] is secure under these definitions. Independently, Bellare et al. [19] extended the definitions of [17] in a similar way, and presented a scheme that is secure under general assumptions.

1.4.3 Constructions

The early constructions of group signatures given by Chaum [48], Chen [49], and Camenisch [32] all suffer from the problem that the size of a signature grows with the number of signers. The first scheme that does not have this deficiency was given by Camenisch and Stadler [38]. Their construction turned out to be insecure, but the general approach is sound.

Before we can explain their approach we need to review an idea of Fiat and Shamir [64]. Suppose that a party distributes a public key pk, but keeps the corresponding private key sk secret. Assume also that the public key is computed using the private key. This is often the case in practice. Then the owner of pk can prove, using a zero-knowledge proof of knowledge, that it knows the private key sk corresponding to pk. A special class of such proofs have the following structure. The prover sends a message α to the verifier. Then the verifier chooses a random bit-string c, a so called challenge, and hands it to the prover. Finally, the prover computes a reply e and hands it to the verifier and the verifier checks the tuple (pk, α, c, e) in some way to decide if it accepts the proof. When this is the case one can replace interaction with the verifier by an invocation of a cryptographic hash function. Thus, instead of asking the verifier to send challenge, the prover generates a bit-string by computing

$$c = H(pk, \alpha)$$

using a cryptographic hash function H. A cryptographic hash function behaves very much like a randomly chosen function, so this should give a result similar to running the interactive protocol with the verifier. Finally, the prover sends the tuple (pk, α, c, e) to the verifier, which checks the proof in the corresponding way, i.e., by evaluating the hash function. The advantage of the Fiat-Shamir scheme is that it reduces interaction to a single message. Furthermore, if the prover includes a message m in its inputs to the hash function, i.e., it computes $c = H(m, pk, \alpha)$, the result is a signature scheme. This follows since anybody can verify the proof and as soon as the message m is altered the output of the hash function is very different from before. Thus, somebody not holding sk is likely to fail to compute a valid proof.

We now consider the approach of Camenisch and Stadler. The group manager M has a private key sk and a public key pk of a probabilistic cryptosystem. Each signer S_i is given an individual private key sk_i. The private key sk_i is a signature of a fixed public message in the sense that given a candidate private key sk_i it can be validated that sk_i is generated in co-operation with M, and no adversary can generate a private key sk_i on its own that validates in this way. To form a signature of a message m the signer computes a cryptotext

$$C = \mathsf{Enc}_{pk}(sk_i)$$

and then proves to the group manager, using a Fiat-Shamir proof, that it formed the cryptotext in this way using a valid private key sk_i. The signer includes the message in its invocation of the one-way hash function as described above. In other words the signer forms the first message α of the proof, computes a challenge $c = H(m, pk, C, \alpha)$, and finally the reply e following the interactive zero-knowledge proof of knowledge. This gives a group signature (α, c, e) of m, that can be verified in the corresponding way. Similarly to the Fiat-Shamir signatures, an adversary is unlikely to be able to compute a new valid proof if the message m is modified in any way or if C is not on the expected form.

To summarize, a signature can not be computed unless the signer holds a private key sk_i and encrypts this private key using the public key of the group manager. The signature does not leak any knowledge on the identity of the signer, since its identity is encrypted and the proof of knowledge is zero-knowledge.

1.4.4 What is a Hierarchical Group Signature Scheme?

The payment card application described above for group signatures is somewhat simplified since normally there are many banks that issue cards of the same brand which are processed through the same payment network. The payment network normally works as an administrator and routes transactions to several independent banks. Thus, the merchant hands a payment to the payment network which hands the payment to the issuing bank. We could apply group signatures here as well by making the payment network act as the group manager. The network would send the extracted identity to the issuing bank. Another option is to set up several independent group signatures schemes, one for each issuer. In the first approach, the payment network learns the identity of the customer, and in the second approach the merchant learns which bank issued the customer's card. A better solution would reveal nothing except what is absolutely necessary to each party. The merchant needs to be convinced that the credit card is valid, the payment network must be able to route the payment to the correct card issuer, and the issuer must be able to determine the identity of the cardholder.

A solution that comes to mind is to use ordinary group signatures with the modification that the customer encrypts his identity with his bank's public key. Then we have the problem of showing to the merchant that this encryption contains

valid information. The customer cannot reveal the public key of the bank to the merchant, making such a proof far from trivial.

In this thesis we introduce and investigate the notion of *hierarchical group signatures*. These can be employed to solve the above problem. When using a hierarchical group signature scheme there is not one single group manager. Instead there are several group managers organized in a tree, i.e., each group manager either manages a group of signers or a group of group managers. This is illustrated in Figure 1.3.

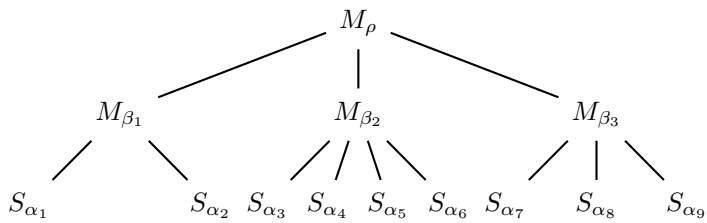

Figure 1.3: A tree of group managers and signers, where $\rho = \{\beta_1, \beta_2, \beta_3\}$, $\beta_1 = \{\alpha_1, \alpha_2\}$, $\beta_2 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6\}$, and $\beta_3 = \{\alpha_7, \alpha_8, \alpha_9\}$.

In the original notion the group manager can always identify the producer of a signature, but nobody else can distinguish between signatures produced by different signers. The corresponding property for hierarchical group signatures is more complicated. When opening a signature from a signer in its subtree, a group manager learns to which of the subtrees directly below it the signer belongs, but nothing else. Signatures from other signers are indistinguishable. Hence a group manager on the level directly above the signers can identify its signers, whereas group managers higher in the hierarchy only learns to which of its immediate subtrees the signer belongs.

When we use hierarchical group signatures to construct anonymous credit cards for the more realistic setting we let the payment network be the root manager that manages a set of group managers, i.e., the issuing banks, and we let the cardholders be signers. The credit card application also demonstrates what kind of responsibility model is likely to be used with a hierarchical group signature scheme. With a valid signature on a transaction, the merchant has a valid demand on the payment network. If the payment network has a signature that can be shown to belong to a certain bank, the network has a valid demand on that bank. Thus, it is in the network's interest to open the signatures it receives from merchants, and it is in the issuing banks' interest to open the signatures they receive from the network.

At a high level the tools we use to construct hierarchical group signatures are similar to those used to construct ordinary group signatures, but the construction is far more complicated.

1.4.5 Summary of Our Contributions

We introduce the notion of hierarchical group signatures. This is a proper generalization of group signatures. We provide definitions for the new notion and construct a scheme that is provably secure given the existence of a family of trapdoor permutations. We also present a construction which is relatively practical, and prove its security in the random oracle model under standard computational assumptions.

1.5 Organization of the Thesis

The thesis consists of three parts. The first part contains mostly definitions and constructions taken from the literature, although we modify some of these. Our results are presented in the second and third parts of the thesis. In this way the definitions and the constructions taken from the literature are presented in a systematic way that allows easy reference, and it is easy to distinguish previous results from our contributions.

The remainder of the first part of the thesis is organized as follows. We conclude this chapter with a list of the publications on which the thesis is based. We introduce notation, primitives, and security definitions in Chapter 2. In Chapter 3 we introduce the computational assumptions on which the security of our protocols rest and the concrete primitives we use. The universally composable security framework in which we analyze our mix-nets is defined in Chapter 4.

The second part of the thesis contains our results on mix-nets, and is divided into chapters as follows. In Chapter 5 we give a more technical description of previous work on mix-nets and introduce notation specific to the second part of the thesis. In Chapter 6 we describe practical attacks against a mix-net of Golle et. al [79]. This serves as motivation for the following chapters. In Chapter 7 we introduce a definition of a secure mix-net. In Chapter 8 we then describe the first efficient and universally composable mix-net. The mix-net is described in a model with access to an ideal zero-knowledge proof of knowledge of the cleartext of an El Gamal cryptotext, and an ideal zero-knowledge proof of knowledge of a witness of correct behavior of a mix-server. In Chapter 9 we describe a novel approach to construct a zero-knowledge proof of knowledge of a witness of a correct shuffle. This is a proof of knowledge in the classical sense, i.e., rewinding is necessary to extract the witness. We also show in Chapter 10 how this can be turned into an ideal zero-knowledge proof of knowledge of a witness of correct behavior of a mix-server. In Chapter 11 we describe the first mix-net that is provably secure against an adaptive adversary.

The third part of the thesis contains our results on hierarchical group signatures, and is divided into chapters as follows. In Chapter 13 we discuss related work on group signatures and define the notion of hierarchical group signatures and its security. We also discuss the difficulties involved in constructing a hierarchical group signature scheme. In Chapter 14 we describe a construction of hierarchical group signatures that is secure under the existence of a trapdoor permutation family.

In Chapter 15 we give a construction that is secure and almost practical under standard computational assumptions. This construction requires a complex zero-knowledge proof of knowledge, which is presented and analyzed in Chapter 16.

1.6 Our Publications

The thesis is based on the results originally presented in the publications below. We have reproduced the entries from the bibliography at the end of the thesis.

[146] D. Wikström. Five practical attacks for "optimistic mixing for exit-polls". In *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 160–174. Springer Verlag, 2003.

Chapter 6 is based on this paper.

[147] D. Wikström. A universally composable mix-net. In *1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 315–335. Springer Verlag, 2004.

The definition of a secure mix-net in Chapter 7 and the secure realization of the proof of knowledge of a cleartext in Chapter 10 are taken from this paper.

[148] D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 273–292. Springer Verlag, 2005. (Full version [149]).

Chapter 8, Chapter 9, and a part of Chapter 10 are based on this paper.

[150] D. Wikström and J. Groth. An adaptively secure mix-net without erasures. submitted manuscript.

Chapter 11 is based on the results in this paper. The contribution of the author of the thesis to the paper is approximately 75%.

[141] M. Trolin and D. Wikström. Hierarchical group signatures. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 446–458. Springer Verlag, 2005. (Full version [140]).

The entire second part of this thesis is based on this paper. The contribution of the author of the thesis to the paper is approximately 50%.

Chapter 2

Notation and Basic Definitions

In this chapter we first introduce notation that is common for all parts of the thesis. Then we introduce definitions for the primitives we use and their respective security properties. Most definitions are taken almost directly from the literature, but the definition of a computationally convincing proof is non-standard.

2.1 Notation and Conventions

We write $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ to denote the natural numbers, the integers, and the reals respectively. We write $[a, b]$ to denote either the set of integers $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$ or the closed subset $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ of real numbers. The intended meaning is hopefully clear from the context. We write $A \times B$ for the Cartesian product of two sets A and B. We denote the set of permutations of N elements by Σ_N. We use $\emptyset$ to denote both the empty set and the empty string. If s is a string $|s|$ denotes its bit-size, if S is a set $|S|$ denotes its cardinality, and if a is a number $|a|$ denotes its absolute value. The intended meaning is always clear from the context. We write $\lceil r \rceil$ to denote the smallest integer $n \geq r$ and we write $\lfloor r \rfloor$ to denote the largest integer $n \leq r$.

We say that an element is chosen "randomly" instead of the more cumbersome "independently and uniformly at random".

We denote the set of all finite binary strings by $\{0,1\}^*$. Sometimes we say that an element is chosen randomly from $\{0,1\}^*$ and interpret this as if a sufficiently long string was chosen randomly. Whenever we do so there exists an explicit bound on the length needed.

We let ϕ denote Euler's ϕ function, and write $\gcd(m, n)$ to denote the greatest common divisor of two integers m and n. We use $\mathbb{Z}_n$ to denote the ring of integers modulo n. Its multiplicative group is denoted by $\mathbb{Z}_n^*$, i.e., the group of elements m such that $\gcd(m, n) = 1$. We use QR_n to denote the subgroup of squares of $\mathbb{Z}_n^*$. The notation G_m is reserved for a cyclic subgroup of order m in $\mathbb{Z}_n^*$. We denote by

$\log_g h$ the logarithm of h in the basis g in a group, e.g. if $g, h \in G_m$ then $g^{\log_g h} = h$. The group considered is hopefully clear from the context.

We write $\log_2 n$ and $\ln n$ to denote the binary and natural logarithms of a number n. We also take the liberty to interpret the result of taking a logarithm as an integer when convenient. In other words we write $\log_2 q$ instead of $\lceil \log_2 q \rceil$ or $\lfloor \log_2 q \rfloor$ which ever is appropriate.

We follow common practice in the cryptographic community and say that a prime p is safe if $(p-1)/2$ is a prime. This another way of saying that $(p-1)/2$ is a Sophie Germain prime.

Throughout the thesis κ denotes the main security parameter, but we also use two additional security parameters κ_c and κ_r extensively. We use κ_c to denote the number of random bits used in challenges in proofs of knowledge. The value of κ_r decides the statistical distance between the distribution of a the view in an execution of a protocol and a simulated view. The exact interpretation of these parameters differ slightly depending on the chapter, so the reader should consider this a convention.

A function $f : \mathbb{N} \to [0, 1]$ is said to be negligible if for each $c > 0$ there exists a $\kappa_0 \in \mathbb{N}$ such that $f(\kappa) < \kappa^{-c}$ for $\kappa > \kappa_0$. We say that a function $f : \mathbb{N} \to [0, 1]$ is non-negligible whenever it is not negligible, and we say that a function f is overwhelming if $1 - f(\kappa)$ is negligible. The additional security parameters are defined such that $2^{-\kappa_c}$ and $2^{-\kappa_r}$ are negligible in κ.

All adversaries in this thesis are modeled as polynomial time Turing machines with non-uniform auxiliary advice string, or equivalently as polynomial size circuit families. We denote the set of such adversaries by PPT^*.

When we say that n is a κ-bit integer, we implicitly mean that it is contained in the interval $[2^{\kappa-1}, 2^\kappa - 1]$.

We say that a distribution ensemble $\mathcal{D} = \{D_\kappa\}_{\kappa \in \mathbb{N}}$ is efficiently sampleable if there exists a probabilistic polynomial time algorithm $T_\mathcal{D}$ that on input 1^κ outputs a random sample distributed according to D_κ.

When we describe experiments we write $y \leftarrow \mathsf{Alg}(x)$ to denote that y is the output of the algorithm Alg when executed on input x. When Alg is a probabilistic algorithm we consider x as sampled with the induced distribution.

Whenever we say that an element is chosen randomly from a set it is possible to generate an element with distribution statistically close to uniform. For example, if q is a prime we may choose a random $2 \log_2 q$ bit string s and output $s \bmod q$ to generate an almost random element in the field $\mathbb{Z}_q$.

2.2 One-Way and Collision-Free Hash Functions

Intuitively, a one-way function is a function that is easy to evaluate, but difficult to invert. This concept was first considered by Diffie and Hellman [60]. A related notion is that of a collision-free function. This means that it is difficult to find two elements that map to the same output.

Below both notions are formalized using collections of functions, but we abuse notation and refer to such a collection as a "one-way hash function" or a "collision-free hash function".

Definition 2.1 (Collection of Functions). A collection of functions (I, F) consists of an infinite index set I and a set of functions $F = \{f_i : D_i \to \{0,1\}^*\}$ with finite domains D_i.

Definition 2.2 (Useful). A collection of functions (I, F) is useful if there exists two probabilistic polynomial time algorithms Gen, Sample, and a deterministic polynomial time algorithm Eval such that

1. The output of $\mathsf{Gen}(1^\kappa)$ is randomly distributed in $I \cap \{0,1\}^\kappa$.
2. The output of $\mathsf{Sample}(i)$, where $i \in I \cap \{0,1\}^\kappa$, is randomly distributed in D_i.
3. If $i \in I \cap \{0,1\}^\kappa$ and $x \in D_i$, then $\mathsf{Eval}(i, x) = f_i(x)$.

When every function $f_i \in F$ is a permutation on D_i we say that (I, F) is a collection of permutations.

Definition 2.3 (One-Way). Let $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval})$ be a useful collection of functions. Define the random variables $I_\kappa = \mathsf{Gen}(1^\kappa)$ and $X_\kappa = \mathsf{Sample}(I_\kappa)$. The function collection is one-way if for every adversary $A \in \mathrm{PPT}^*$ the probability $\Pr[A(I_\kappa, f_{I_\kappa}(X_\kappa)) \in f_{I_\kappa}^{-1}(f_{I_\kappa}(X_\kappa))]$ is negligible in κ.

Definition 2.4 (Collision-Free). Let $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval})$ be a useful collection of functions. Define the random variable $I_\kappa = \mathsf{Gen}(1^\kappa)$. The function collection is collision-free if for every adversary $A \in \mathrm{PPT}^*$ the probability $\Pr[A(I_\kappa) = (x_0, x_1) \wedge x_0 \neq x_1 \wedge f_{I_\kappa}(x_0) = f_{I_\kappa}(x_1))]$ is negligible in κ.

2.3 Trapdoor Permutation Families and Hard-Core Bits

A trapdoor permutation is a permutation that is easy to compute, but difficult to invert, unless one has access to a special trapdoor. Using the trapdoor it is easy to invert the permutation. This notion originates from Diffie and Hellman [60]. An example of a function believed to be a trapdoor permutation is the RSA-function introduced by Rivest, Shamir, and Adleman [132].

Definition 2.5 (Trapdoor Permutation Family). Let Gen and Sample be probabilistic polynomial time algorithms and let Eval and Invert be deterministic polynomial time algorithms. Denote by Gen_l the left half of the output of Gen. Then $\mathcal{TPF} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ is a trapdoor permutation family if the following holds.

1. $(\mathsf{Gen}_l, \mathsf{Sample}, \mathsf{Eval})$ is a one-way collection of permutations such that $D_i = \{0,1\}^\kappa$ for every possible output (i, t) of $\mathsf{Gen}(1^\kappa)$.

2. For every possible output (i, t) of $\mathsf{Gen}(1^\kappa)$ and every $x \in D_i$ we have $\mathsf{Invert}(t, \mathsf{Eval}(i, x)) = x$.

Remark 2.6. The above definition restricts the possible domains to $\{0,1\}^\kappa$. This may be a restriction, but a trapdoor permutation family that fits the above format can be constructed using an idea of Yao from the known candidates, e.g., the RSA-function. We refer the reader to Bellare and Micali [16] for a discussion on this.

Even if it is hard to invert a function it is not necessarily hard to find some of the bits of a pre-image. A hard-core bit is a bit of information of the pre-image that is hard to compute given only the output of the function. This notion was introduced by Blum and Micali [26]. They used it to construct pseudo-random generators which are introduced in the next section. Goldwasser and Micali [76] use this notion to construct a polynomially indistinguishable cryptosystem.

Definition 2.7 (Hard-Core Bit). Let $\mathcal{B} : \{0,1\}^* \to \{0,1\}$ be a function such that there exists a polynomial time algorithm that computes $\mathcal{B}(x)$ on every possible input $x \in \{0,1\}^*$. Let $\mathcal{TPF} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ be a trapdoor permutation family and define the random variables $(I_\kappa, T_\kappa) = \mathsf{Gen}(1^\kappa)$ and $X_\kappa = \mathsf{Sample}(I_\kappa)$. The function $\mathcal{B}$ is a hard-core bit for $\mathcal{TPF}$ if for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[A(I_\kappa, f_{I_\kappa}(X_\kappa)) = \mathcal{B}(X_\kappa)] - 1/2|$ is negligible in κ.

The notion of a hard bit can be defined for any one-way function, but we only use it in conjunction with trapdoor permutation families.

2.4 Pseudo-Random Generators

Even if a random variable is not uniformly distributed it may appear to be so to every polynomially bounded observer. An ensemble of random variables is said to be pseudo-random when this is the case. This notion was first considered by Yao [151].

Definition 2.8 (Pseudo-Random Ensemble). The ensemble $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ of random variables is pseudo-random if there exists a length function $l(\kappa)$ and an ensemble $U = \{U_{l(\kappa)}\}_{\kappa \in \mathbb{N}}$ of random variables uniformly distributed in $\{0,1\}^{l(\kappa)}$ such that for every $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[A(X_\kappa) = 1] - \Pr[A(U_{l(\kappa)}) = 1]|$ is negligible in κ.

We sometimes abuse notation and talk about an ensemble of random variables as a random variable. In most cryptographic activities randomness is needed. Random bits of good quality are often expensive to generate. An alternative is to generate a short seed of truly random bits and then expand this into a longer string of pseudo-random bits. An algorithm that does this is called a pseudo-random generator. This notion was introduced by Blum and Micali [26].

Definition 2.9 (Pseudo-Random Generator). A pseudo-random generator is a deterministic polynomial-time algorithm PRG such that

1. There exists a length function $l : \mathbb{N} \to \mathbb{N}$ such that $l(\kappa) > \kappa$ for all $\kappa \in \mathbb{N}$ and $|\mathsf{PRG}(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.

2. The random variable $\{\mathsf{PRG}(U_\kappa)\}_{\kappa\in\mathbb{N}}$ is pseudo-random.

Håstad, Impagliazzo, Levin, and Luby [82] proves that there exists a pseudo-random generator if there exists a one-way function. There are also more practically oriented constructions such as that given Håstad, Schrift, and Shamir [83] which is secure under the factoring assumption. The strong RSA-assumption introduced in Section 3.8 implies the factoring assumption.

We use a pseudo-random generator in Chapter 9 to reduce the size of the challenge generated by a verifier in a proof of knowledge.

2.5 Public Key Cryptosystems

A public key cryptosystem is used to communicate secretly. The definition below is essentially the one given in Micali, Rackoff, and Sloan [105], but we use different notation.

Definition 2.10 (Public Key Cryptosystem). A public key cryptosystem $\mathcal{CS} = (\mathsf{CSKg}, \mathsf{Enc}, \mathsf{Dec})$ consists of three probabilistic polynomial-time algorithms.

1. A key generation algorithm CSKg that on input 1^κ outputs a public key pk and a private key sk.

2. An encryption algorithm Enc that on input a public key pk and a message m outputs a cryptotext c.

3. A decryption algorithm Dec that on input a private key sk and a cryptotext c outputs a message m.

Furthermore, for each output (sk, pk) of $\mathsf{CSKg}(1^\kappa)$ and each m it must hold that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$.

The last requirement is sometimes relaxed, i.e. $\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m]$ may hold only with overwhelming probability, but the stricter definition is more convenient for our purposes. When we want to make the internal randomness used by Enc explicit we write $\mathsf{Enc}_{pk}(m, r)$ with r in some randomizer space $\mathcal{R}_{(pk,sk)}$ instead of writing $\mathsf{Enc}_{pk}(m)$.

2.5.1 Polynomial Indistinguishability

The first definition of security of public key cryptosystems we use was introduced by Goldwasser and Micali in their seminal paper [76] on probabilistic encryption.

Consider the following experiment running with an adversary A. The adversary is given a public key output by the key generator and outputs two messages m_0 and

m_1. The experiment then encrypts m_b and hands the cryptotext to the adversary. Finally, the adversary must output a guess d of the value of b. If the cryptosystem is polynomially indistinguishable the probability that $d = b$ when $b \in \{0, 1\}$ is random should be close to $1/2$.

Experiment 2.11 (Polynomial Indistinguishability, $\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa)$).

$$\begin{aligned} (pk, sk) &\leftarrow \mathsf{CSKg}(1^\kappa) \\ (m_0, m_1, \text{state}) &\leftarrow A(\mathsf{choose}, pk) \\ c &\leftarrow \mathsf{Enc}_{pk}(m_b) \\ d &\leftarrow A(\mathsf{guess}, \text{state}, c) \end{aligned}$$

The experiment returns d.

Without loss we implicitly assume above that $m_0, m_1 \in \mathcal{M}_{(pk,sk)}$ and that A outputs a single bit. The labels choose and guess are only used to signal to the adversary in which phase of the experiment it is executing. The variable state denotes the internal state the algorithm wishes to remember between the two phases of the experiment. Similar conventions are used for all experiments in the thesis.

Definition 2.12 (Polynomial Indistinguishability). A public key cryptosystem $\mathcal{CS}$ is polynomially indistinguishable if for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS},A}(\kappa) = 1]|$ is negligible in κ.

Micali, Rackoff, and Sloan [105] show that for non-uniform adversaries polynomial indistinguishability is equivalent to semantic security. Thus, we follow common practice in the literature and use both terms interchangeably. Informally, semantic security means that no adversary can compute anything about an encrypted cleartext better than guessing.

The following generalization is standard. Denote by $\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa)$ the experiment above except for the following changes. Let $\mu_1(\kappa)$ and $\mu_2(\kappa)$ be polynomially bounded in κ. The experiment generates a list $((pk_1, sk_1), \ldots, (pk_{\mu_1}, sk_{\mu_1}))$ of key pairs instead (pk, sk). Then the adversary is given $(pk_1, \ldots, pk_{\mu_1})$. The adversary then outputs $m_0 = (m_{0,1,1}, \ldots, m_{0,\mu_1,\mu_2})$ and $m_1 = (m_{1,1,1}, \ldots, m_{1,\mu_1,\mu_2})$. Finally, the encryption oracle computes $c = (\mathsf{Enc}_{pk_i}(m_{b,i,j}))^{\mu_1,\mu_2}_{i=1,j=1}$ instead of a single cryptotext $\mathsf{Enc}_{pk}(m_b)$. The following lemma follows by a straightforward hybrid argument.

Lemma 2.13. *If $\mathcal{CS}$ is polynomially indistinguishable, then for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{ind}-1}_{\mathcal{CS},A}(\kappa) = 1]|$ is negligible in κ.*

2.5.2 Anonymity

In some applications polynomial indistinguishability is not sufficient. The problem is that although the adversary can not learn anything about the encrypted cleartext,

it may be able to tell which public key was used to compute the cryptotext. A cryptosystem that hides the public key used for encryption is called anonymous. This property was discussed by Abadi and Rogaway [1] and studied extensively by Bellare et al. in [14]. It turns out to be essential in our construction of hierarchical group signatures in the third part of the thesis. Anonymity is formalized by an experiment similar to the polynomial indistinguishability experiment.

Experiment 2.14 (Anonymity, $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A}(\kappa)$).

$$\begin{aligned}(pk_0, sk_0) &\leftarrow \mathsf{CSKg}(1^\kappa)\\ (pk_1, sk_1) &\leftarrow \mathsf{CSKg}(1^\kappa)\\ (m, \text{state}) &\leftarrow A(pk_0, pk_1)\\ c &\leftarrow \mathsf{Enc}_{pk_b}(m)\\ d &\leftarrow A(\mathsf{guess}, \text{state}, c)\end{aligned}$$

The experiment returns d.

Note that compared to the definition of polynomial indistinguishability, the roles played by public keys and messages are reversed. One could consider a variant experiment that captures both types of indistinguishability, but we think it is more natural to think of anonymity as an additional property.

Definition 2.15 (Anonymity). A cryptosystem $\mathcal{CS}$ is anonymous if for all $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A}(\kappa) = 1]|$ is negligible in κ.

The property of anonymity is clearly useless if the cryptosystem is not indistinguishable, since it allows the encryption function to be the identity map. Thus, anonymity does not imply indistinguishability. To see that the reverse implication is false, note that if $\mathcal{CS}$ is a polynomially indistinguishable cryptosystem, then so is the cryptosystem where the encryption and decryption functions $c = \mathsf{Enc}_{pk}(m)$ and $\mathsf{Dec}_{sk}(c) = m$ are replaced by $(c, c') = \mathsf{Enc}'_{pk}(m) = (\mathsf{Enc}_{pk}(m), pk)$ and $\mathsf{Dec}'_{sk}(c, c') = \mathsf{Dec}_{sk}(c) = m$ respectively, and this is clearly not anonymous.

The following generalization follows by a similar argument as the generalization of polynomial indistinguishability. Denote by $\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-b}_{\mathcal{CS},A}(\kappa)$ the experiment above except for the following changes. Let $\mu_1(\kappa)$ and $\mu_2(\kappa)$ be polynomially bounded in κ. The experiment generates lists $((pk_{1,b}, sk_{1,b}), \ldots, (pk_{\mu_1,b}, sk_{\mu_1,b}))$ for $b \in \{0,1\}$ instead of (pk_0, sk_0) and (pk_1, sk_1). Then the adversary is given $(pk_{1,b}, \ldots, pk_{\mu_1,b})$ for $b \in \{0,1\}$. The adversary outputs $m = (m_{1,1}, \ldots, m_{\mu_1,\mu_2})$. Finally, the encryption oracle computes $c = (\mathsf{Enc}_{pk_{i,b}}(m_{i,j}))_{i=1,j=1}^{\mu_1,\mu_2}$ instead of a single cryptotext. The lemma below follows by a straightforward hybrid argument.

Lemma 2.16. *If $\mathcal{CS}$ is polynomially indistinguishable, then for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-1}_{\mathcal{CS},A}(\kappa) = 1]|$ is negligible in κ.*

2.5.3 Chosen-Ciphertext Security

When a cryptosystem is used in a protocol a more stringent security property than polynomial indistinguishability is often needed. Experiment 2.11 above should have the same unrewarding result for the adversary even if it can ask the experiment to decrypt arbitrarily many other cryptotexts before and after receiving the challenge cryptotext. This notion of security was developed by Naor and Yung [111], Rackoff and Simon [130], and Dolev, Dwork and Naor [61], and a scheme that has this property is said to be secure against chosen ciphertext attacks (CCA). We remark that there are some variations of this notion and that we formalize what is called CCA2-security. Formally, the adversary is given access to a decryption oracle $\mathsf{Dec}_{sk}(\cdot)$ during the experiment.

Experiment 2.17 (CCA2-Security, $\mathsf{Exp}^{\mathsf{cca2}-b}_{\mathcal{CS},A}(\kappa)$).

$$\begin{aligned}(pk, sk) &\leftarrow \mathsf{CSKg}(1^\kappa)\\ (m_0, m_1, \text{state}) &\leftarrow A^{\mathsf{Dec}_{sk}(\cdot)}(\mathsf{choose}, pk)\\ c &\leftarrow \mathsf{Enc}_{pk}(m_b)\\ d &\leftarrow A^{\mathsf{Dec}_{sk}(\cdot)}(\mathsf{guess}, \text{state}, c)\end{aligned}$$

The experiment returns 0 if $\mathsf{Dec}_{sk}(\cdot)$ was queried on c, and d otherwise.

Definition 2.18 (CCA2-Security). A public key cryptosystem $\mathcal{CS}$ is said to be secure against chosen ciphertext attacks (CCA2-secure) if for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[\mathsf{Exp}^{\mathsf{cca2}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{cca2}-1}_{\mathcal{CS},A}(\kappa) = 1]|$ is negligible in κ.

2.6 Signature Schemes

A signature scheme is the digital equivalent of a written signature. Only the holder of the private key can sign a message, but anybody can use the public key to verify the validity of a signature.

Definition 2.19 (Signature Scheme). A signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ consists of three polynomial-time algorithms

1. A probabilistic key generation algorithm Kg that on input 1^κ outputs a public key pk and a private key sk.
2. A probabilistic signature algorithm Sig that on input a private key sk and a message m outputs a signature s.
3. A deterministic verification algorithm Vf that on input a public key pk, a message m, and a signature s outputs a bit $b \in \{0, 1\}$.

Furthermore, for each output (sk, pk) of $\mathsf{Kg}(1^\kappa)$ and each m it must hold that $\mathsf{Vf}_{pk}(m, \mathsf{Sig}_{sk}(m)) = 1$.

2.6.1 Security Against Chosen Message Attacks

The standard definition of security of a signature scheme was introduced by Goldwasser, Micali and Rivest [78]. A signature scheme is said to be secure against chosen message attacks (CMA) if no adversary can output a message of its own choice and a valid signature on the message, even if it can ask for arbitrarily many signatures of messages of its choice. It is obviously not allowed to output a signature of a message on which it has requested a signature. Formally, the adversary is given access to a signature oracle $\mathsf{Sig}_{sk}(\cdot)$.

Experiment 2.20 (CMA-Security, $\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa)$).

$$\begin{aligned}(pk, sk) &\leftarrow \mathsf{Kg}(1^\kappa)\\ (m, s) &\leftarrow A^{\mathsf{Sig}_{sk}(\cdot)}(\mathsf{guess}, pk)\end{aligned}$$

If $\mathsf{Vf}_{pk}(m, s) = 1$ and $\mathsf{Sig}_{sk}(\cdot)$ was never queried on m return 1, else return 0.

Definition 2.21 (CMA-Security). A signature scheme $\mathcal{SS}$ is CMA-secure if for all adversaries $A \in \mathrm{PPT}^*$ the probability $\Pr[\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = 1]$ is negligible in κ.

2.7 Statistical Closeness

One standard metric on the space of random variables is the statistical distance. We say that two random variables are statistically close if their statistical distance is negligible. This notion of closeness is useful in cryptography.

Definition 2.22 (Statistical Distance). The statistical distance between random variables X_κ and Y_κ is defined by $\Delta(X_\kappa, Y_\kappa) = \frac{1}{2}\sum_\alpha |\Pr[X_\kappa = \alpha] - \Pr[Y_\kappa = \alpha]|$, where the sum is taken over the support of X_κ and Y_κ.

Definition 2.23 (Statistical Closeness). Two random variables $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $\{Y_\kappa\}_{\kappa\in\mathbb{N}}$ are statistically close if $\Delta(X_\kappa, Y_\kappa)$ is negligible in κ.

The following lemma is immediate.

Lemma 2.24. *If two random variables $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $\{Y_\kappa\}_{\kappa\in\mathbb{N}}$ are statistically close then for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[A(X_\kappa) = 1] - \Pr[A(Y_\kappa) = 1]|$ is negligible in κ.*

2.8 Proofs of Knowledge, Proofs, and Zero-Knowledge

A proof of knowledge can be used by one party P, called the prover, to convince another party V, called the verifier, that it knows a witness of some fact about a common object. For example, the prover can prove knowledge of the private key corresponding to a public key of a cryptosystem. A proof on the other hand allows

P to convince V of the validity of a statement about some common object, without necessarily knowing a witness.

Typically, the proofs of knowledge and the proofs used in cryptographic protocols are also zero-knowledge. This means that they disclose no knowledge to the verifier. The concepts of zero-knowledge proofs of knowledge and proofs were introduced by Goldwasser, Micali, and Rackoff [77].

There are many variations of these concepts and different authors use different terms to distinguish between these variations. In informal descriptions and discussions we use the terms "proof of knowledge", "proof", and "zero-knowledge" to denote any protocol that loosely speaking has the above properties. This convention is common in the literature. In formal statements we use the definitions introduced below.

Before we continue we recall the definition of an NP-relation and the complexity class NP.

Definition 2.25 (Polynomially Bounded). A relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is polynomially bounded if there exists a polynomial $p(\cdot)$ such that $|y| \leq p(|x|)$ for all $(x,y) \in R$.

Definition 2.26 (NP-Relation). A relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is an NP-relation if it is polynomially bounded and there exists a deterministic polynomial machine M such that $M(x,y) = R(x,y)$.

Definition 2.27 (Complexity Class NP). A language $L_R \subset \{0,1\}^*$ belongs to NP if there exists an NP-relation R such that $x \in L_R$ if and only if there exists an $y \in \{0,1\}^*$ such that $(x,y) \in R$.

Every relation R considered in this thesis corresponds to a language $L_R \in \text{NP}$ in the sense of the definition. Given two NP-relations R_1 and R_2 we denote by $R_1 \vee R_2$ the relation defined by $((x_1,x_2),w) \in R_1 \vee R_2$ if and only if $(x_1,w) \in R_1$ or $(x_2,w) \in R_2$. Similarly we denote by $R_1 \wedge R_2$ the relation defined by $((x_1,x_2),(w_1,w_2) \in R_1 \vee R_2$ if $(x_1,w_1) \in R_1$ and $(x_2,w_2) \in R_2$.

2.8.1 Computationally Convincing Proofs of Knowledge

The standard definition of a proof of knowledge given by Bellare and Goldreich [15] is too strict for our setting. The standard definition states that there must exist an algorithm, called the knowledge extractor, which for every $x \in R$ and every prover P^* that convinces an honest verifier with non-negligible probability outputs a witness w such that $(x,w) \in R$ in expected polynomial time, using P^* as a blackbox. Several of our protocols do not satisfy this definition, but they satisfy a relaxed definition that is sufficient in many settings.

Damgård and Fujisaki [55] introduce a definition that captures a weaker form of a proof of knowledge. They introduce the notion of a "relation generator" that is invoked before the protocol between the prover and verifier is executed. The relation

generator outputs a relation. Then the prover chooses an instance of the relation, and the protocol is executed with the verifier. Knowledge extraction should then be possible with overwhelming probability over the randomness of the relation generator. A protocol that satisfies this extraction property is called a computationally convincing proof of knowledge.

We use a variation of this definition. We simplify the definition in that we do not mention the knowledge error explicitly, since our reductions are not exact anyway. We also rephrase the definition to allow us to state our results in a more natural way.

We analyze our protocols in the following setting. Let R be an NP-relation. The adversary is given a special joint parameter h chosen from a set H and outputs an instance x. Then the prover and verifier execute the protocol on the joint input x and the special parameter h. The protocol is said to be a computationally convincing proof of knowledge for R with regards to the distribution of h if it holds that if the prover convinces the verifier with non-negligible probability, then a witness w such that $(x, w) \in R$ can be extracted in expected polynomial time with overwhelming probability over the randomness of h and the internal randomness of the prover.

More precisely, we denote by $I_{P^*}(\kappa, h, r_\mathrm{p})$ the instance output by the prover when run on security parameter 1^κ, special joint parameter $h \in H$, and internal randomness r_p. We denote by $\mathrm{view}^V_{P^*}(\kappa, h, r_\mathrm{p}, r_\mathrm{v})$ the view of the verifier when P^* is executed on common input $I_{P^*}(\kappa, h, r_\mathrm{p})$, special input h and random input r_p, and V is executed on common input $I_{P^*}(\kappa, h, r_\mathrm{p})$, special input h and random input r_v. Thus, the view contains the special parameter, all messages exchanged by the prover and verifier, and also the random string of the verifier. It does not contain the random string of the prover. Denote by Acc_V a predicate that on input a view outputs the output of V in the protocol. Finally, define $\delta^V_{P^*}(\kappa, h, r_\mathrm{p}) = \Pr_{r_\mathrm{v}}[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(\kappa, h, r_\mathrm{p}, r_\mathrm{v})) = 1]$. To simplify the exposition we sometimes omit the security parameter from our notation.

Definition 2.28 (Computationally Convincing Proof of Knowledge). A protocol (P, V) is a computationally convincing proof of knowledge for an NP-relation R with regards to the distribution of $h \in H$ if there exists a probabilistic oracle algorithm $\mathcal{X}^{(\cdot)}$ called the knowledge extractor, and a polynomial $p(\kappa)$ such that for all $P^* \in \mathrm{PPT}^*$ the following holds

1. If $\delta^V_{P^*}(\kappa, h, r_\mathrm{p})$ is non-negligible in κ, then $\mathcal{X}^{P^*}$ executes in expected time $O(p(\kappa)/\delta^V_{P^*}(\kappa, h, r_\mathrm{p}))$ on input (h, r_p).

2. For every constant c, if $\Pr_{h, r_\mathrm{p}}[\delta^V_{P^*}(\kappa, h, r_\mathrm{p}) \geq \kappa^{-c}]$ is non-negligible, then

$$\Pr[(I_{P^*}(\kappa, h, r_\mathrm{p}), \mathcal{X}^{P^*}(\kappa, h, r_\mathrm{p})) \in R \mid \delta^V_{P^*}(\kappa, h, r_\mathrm{p}) \geq \kappa^{-c}]$$

   is overwhelming in κ, where the probability is taken over h, r_p and the internal randomness of $\mathcal{X}^{P^*}$.

We can recover a coarse grained version of the standard definition of a proof of knowledge as follows.

Definition 2.29 (Proof of Knowledge). A protocol (P, V) is a proof of knowledge for an NP-relation R if it is a computationally convincing proof with regards to every constant distribution on $h \in H$.

2.8.2 Computationally Convincing Proofs

The standard definition of a proof introduced by Goldwasser, Micali, and Rackoff [77] requires that no adversary can convince the honest verifier of any false statement with probability exceeding 1/3. Another formulation that is more useful in cryptography requires the probability to be negligible. Loosely speaking the two definitions are equivalent, since a protocol that satisfies the first definition can be repeated sequentially to give a protocol that satisfies the second definition. In any case these definitions are too strict for our setting.

We consider the same adversarial model as for computational convincing proofs of knowledge, i.e., the adversary is given a special parameter, outputs an instance, and then executes the protocol with the verifier. In contrast to the standard definition soundness does not hold for all common inputs, only with overwhelming probability for a common input chosen by the adversary. More precisely we use the following definitions.

Definition 2.30 (Computationally Convincing Proof). A protocol (P, V) is a computationally convincing proof for an NP-relation R with regards to the distribution of $h \in H$ if for all provers $P^* \in \mathrm{PPT}^*$ the probability

$$\Pr[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(h, r_\mathrm{p}, r_\mathrm{v})) = 1 \wedge I_{P^*}(h, r_\mathrm{p}) \notin L_R]$$

is negligible in κ.

We can recover the standard definition of a proof as follows.

Definition 2.31 (Proof). A protocol (P, V) is a proof for an NP-relation R if it is a computationally convincing proof with regards to every constant distribution on $h \in H$.

Note that a computationally convincing proof is something different than a computationally sound proof. A computationally sound proof is sound for every input, but only computationally so.

It turns out that every computationally convincing proof of knowledge is also a computationally convincing proof. There may however be computationally convincing proofs that are not computationally convincing proofs of knowledge.

Proposition 2.32 (Soundness). *If (P, V) is a computationally convincing proof of knowledge for an* NP*-language L_R with regards to the distribution of $h \in H$, then it is also a computationally convincing proof for the same parameters.*

Proof. Consider an arbitrary prover P^*. We first prove that for every constant c exists a κ_0 such that

$$\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}, r_{\mathrm{v}}) \geq \kappa^{-c}] < \kappa^{-c} \ . \tag{2.1}$$

If this is not the case there exists a malicious prover P^*, a constant c and an infinite index set $\mathcal{N}$ such that

$$\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \kappa^{-c} \ ,$$

for $\kappa \in \mathcal{N}$. This implies that $\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \mid \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \kappa^{-c}$ and $\Pr[\delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \kappa^{-c}$. Since the protocol is a proof of knowledge we conclude that

$$\Pr[(I_{P^*}(h, r_{\mathrm{p}}), \mathcal{X}_{P^*}(h, r_{\mathrm{p}})) \in R \mid \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \kappa^{-c}]$$

is overwhelming. The union bound implies that

$$\Pr[(I_{P^*}(h, r_{\mathrm{p}}), \mathcal{X}_{P^*}(h, r_{\mathrm{p}})) \in R \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \mid \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \ ,$$

which gives $\Pr[(I_{P^*}(h, r_{\mathrm{p}}), \mathcal{X}_{P^*}(h, r_{\mathrm{p}})) \in R \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R] \geq \frac{1}{2\kappa^{2c}} > 0$. This is obviously a contradiction, since $I_{P^*}(h, r_{\mathrm{p}})$ is either an element in L_R or it is not.

Next we prove the statement in the proposition using Equation (2.1). Suppose that the statement in the proposition is false. Then there exists a malicious prover P^*, a constant c and an infinite index set $\mathcal{N}$ such that

$$\Pr[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(h, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R] \geq \kappa^{-c}$$

for $\kappa \in \mathcal{N}$. We have

$$\begin{aligned}
\frac{1}{\kappa^c} &\leq \Pr[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(h, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R] \\
&= \Pr[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(h, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \frac{1}{2\kappa^c}] \\
&\quad + \Pr[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(h, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}) < \frac{1}{2\kappa^c}] \\
&\leq \Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \frac{1}{2\kappa^c}] + \frac{1}{2\kappa^c} \ .
\end{aligned}$$

From this we conclude that $\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_R \wedge \delta^V_{P^*}(h, r_{\mathrm{p}}) \geq \frac{1}{2\kappa^c}] \geq \frac{1}{2\kappa^c}$. This contradicts Equation (2.1), and the proposition holds. □

2.8.3 Honest Verifier Statistical Zero-Knowledge

A protocol is zero-knowledge if a verifier's view of the protocol can be simulated in a way that is indistinguishable from the verifiers view of a real execution of the protocol. The concept of zero-knowledge is fundamental in modern cryptography.

It was introduced by Goldwasser, Micali, and Rackoff [77]. There are many flavors of this concept and we only formalize the one we use in this thesis. In all our applications the verifier is honest. Thus, we only consider honest verifier zero-knowledge. Informally, this means that we only require that the view of an honest verifier can be simulated. On the other hand all our protocols are statistical zero-knowledge. Thus, by indistinguishability of views we mean that their distributions are statistically close. The strong type of indistinguishability simplifies our security proofs considerably.

Denote by $\text{hview}_P^V(h, x, w)$ the view of the verifier when the protocol (P, V) is executed on special input h, common input x, and the prover is given a witness w as auxiliary input. In other words we consider the view of an honest verifier when executing the protocol with the honest prover. If we want to make the randomness of the verifier explicit we write $\text{hview}_P^V(h, x, w, c)$.

Definition 2.33 (Honest Verifier Statistical Zero-Knowledge). A protocol (P, V) is honest verifier statistical zero-knowledge if there exists a probabilistic polynomial time algorithm S, called the simulator, such that for each special parameter $h \in H$ and each common input x such that $x \in L_R$, the distributions of $\text{hview}_P^V(h, x, w)$ and $S(h, x)$ are statistically close in κ. If the distributions are identical, we say that the protocol is honest verifier perfect zero-knowledge.

2.8.4 Completeness

Let R be an NP-relation. The completeness of a protocol (P, V) is the probability that an honest verifier outputs 1 after interacting with an honest prover. Denote by $\langle P(h, x, w), V(h, x)\rangle$ the output of V on an interaction with the honest prover P on special input $h \in H$ and a common input x, where P is also given a witness w such that $(x, w) \in R$.

Definition 2.34 (Completeness). A computationally convincing proof of knowledge (P, V) has completeness p if for all special parameters $h \in H$ and all $(x, w) \in R$ we have $\Pr[\langle P(h, x, w), V(h, x)\rangle = 1] \geq p$ where the probability is taken over the internal randomness of P and V. If $p = 1$ we say that the protocol has perfect completeness.

2.8.5 Sigma-Protocols

We consider the set of protocols between a prover P and a verifier V that have three rounds: P sends a message α to V, V sends a challenge c to P, and P sends a reply d to V. Furthermore, suppose that c is randomly chosen in some set C. We call such protocols C-three-move protocols. Note that the view of an honest verifier V in a C-three-move protocol can be written (x, α, c, d), where x is the common input, α is the first message sent by P, c is the random challenge of V, and d is final message sent by P.

Definition 2.35 (Special Honest Verifier Statistical Zero-Knowledge). Let (P, V) be a C-three-move protocol for a language L. We say that (P, V) is special honest verifier statistical zero-knowledge if there exists a probabilistic polynomial time algorithm S, called the simulator, such that for each $(x, w) \in R$ and $c \in C$, the distributions of $S(\kappa, x, c)$ and $\text{hview}_P^V(\kappa, x, w, c)$ are statistically close in κ.

The term *special* is used since the simulator S is not allowed to pick the challenge c itself, but must be able to compute a valid view when given c together with x as input.

Suppose the challenge $c = (c_1, \ldots, c_k)$ is randomly chosen from a product set $C_1 \times \cdots \times C_k$ for some constant k and that $1/|C_i|$ is negligible for $i = 1, \ldots, k$ where κ is the security parameter. Then the following slight generalization of special soundness makes sense. We get the standard definition of special-sound if $k = 1$.

Definition 2.36 (Special Soundness). Let $C = C_1 \times \cdots \times C_k$. A C-three-move protocol (P, V) for a relation R is C-special-sound if there exists a deterministic polynomial time algorithm that given two accepting views (x, α, c, d) and (x, α, c', d') with $c_i \neq c'_i$ for $i = 1, \ldots, k$, outputs a witness w such that $(x, w) \in R$.

We use a generalized definition of Σ-protocol along the lines suggested by Cramer, Damgård, and Schoenmakers [51].

Definition 2.37 (Σ-Protocol). Let $C = C_1 \times \cdots \times C_k$. A C-Σ-protocol is a C-three-move protocol (P, V) that is statistical special honest verifier zero-knowledge, C-special-sound, and has overwhelming completeness.

Composition of Sigma-Protocols

There are two natural ways to compose Σ-protocols. Consider a C_1-Σ-protocol π_1 and C_2-Σ-protocol π_2. It is of course possible to run both protocols at the same time, i.e., the messages in each round are concatenated and sent as a single message, and the resulting verifier accepts if both verifiers accepts. We call this parallel composition. If $C_1 = C_2$ we assume that a single challenge is used for both protocols. The following observations follow straightforwardly.

Observation 2.38. Let (P_i, V_i) be a C-Σ-protocol for a language L_i for $i = 1, \ldots, l$, where l is polynomially bounded. Then the parallel composition (P, V) of the protocols where a single challenge in C is used for all protocols is a C-Σ-protocol for the language $L_1 \wedge \cdots \wedge L_{l(\kappa)}$.

Observation 2.39. Let (P_i, V_i) be a C_i-Σ-protocol for a language L_i for $i = 1, \ldots, l$, where l is polynomially bounded. Then the parallel composition (P, V) of the protocols is a $C_1 \times \cdots \times C_l$-$\Sigma$-protocol for the language $L_1 \wedge \cdots \wedge L_{l(\kappa)}$.

Special Sound Protocols are Proofs of Knowledge

Lemma 2.40. *Let $l(\kappa)$ be polynomially bounded and let (P, V) be a $C_1 \times \cdots \times C_l$-special-sound protocol, with $1/|C_i|$ negligible for $i = 1, \ldots, l$, for an* NP*-language L. Then (P, V) is a proof of knowledge.*

Proof. Note that the random input r_{p} of P^* defines the common input $I_{P^*}(\kappa, r_{\mathrm{p}})$, and also the first message α of the prover in the protocol. Consider an extractor $\mathcal{X}^{P^*}$ defined as follows. The extractor repeatedly chooses $r_{\mathrm{v}} \in \{0,1\}^*$ randomly and completes the execution of the protocol with P^* by executing V using r_{v} as random input. This gives a tuple $(I_{P^*}(\kappa, r_{\mathrm{p}}), \alpha, c, d)$. The extractor $\mathcal{X}^{P^*}$ continues until a tuple is found such that

$$\mathrm{Acc}_V(I_{P^*}(\kappa, r_{\mathrm{p}}), \alpha, c, d) = 1 \ .$$

Then the extractor repeatedly chooses $r_{\mathrm{v}}' \in \{0,1\}^*$ randomly and completes the execution of the protocol with P^* by executing V using r_{v}' as random input. This gives a tuple $(I_{P^*}(\kappa, r_{\mathrm{p}}), \alpha, c', d')$. The extractor $\mathcal{X}^{P^*}$ continues until a tuple is found such that

$$\mathrm{Acc}_V(I_{P^*}(\kappa, r_{\mathrm{p}}), \alpha, c', d') = 1 \quad \text{and} \quad c_i' \neq c_i \ \text{ for } i = 1, \ldots, l.$$

Suppose now that $\delta^V_{P^*}(\kappa, r_{\mathrm{p}})$ is non-negligible. In each iteration of the first loop the probability that a tuple is suitable is $\delta^V_{P^*}(\kappa, r_{\mathrm{p}})$. Recall that $c' = (c_1', \ldots, c_l')$ are randomly chosen in $C_1 \times \cdots \times C_l$ for a polynomially bounded l and that $|C_i| \geq 2^\kappa$. Thus, the probability that $c_i = c_i'$ is $1/|C_i|$ and the union bound implies that the the probability that $c_i' = c_i$ for some $i = 1, \ldots, l$ is at most $l(\kappa)2^{-\kappa}$ which is negligible. Another application of the union bound then implies that the probability that a suitable second tuple is found is at least $\delta^V_{P^*}(\kappa, r_{\mathrm{p}})/2$ in each iteration. This implies that the expected number of invocations of P^* is $O(1/\delta^V_{P^*}(\kappa, r_{\mathrm{p}}))$. Thus, the expected execution time of the extractor satisfies the first requirement in Definition 2.29.

It follows immediately from special soundness that the output of the extractor is a valid witness of the fact that $I_{P^*}(\kappa, r_{\mathrm{p}}) \in L$. Thus, the second requirement in Definition 2.29 is satisfied. □

2.9 Non-Interactive Zero-Knowledge Proofs

A non-interactive zero-knowledge proof allows a prover to convince a verifier of a statement by sending a single message, given that both the prover and verifier have access to a public random string. Furthermore, in this process the prover leaks no knowledge to the verifier. We do not construct any explicit non-interactive zero-knowledge proof in the thesis. We only use this notion in a blackbox fashion.

Non-interactive zero-knowledge proofs (NIZK) were introduced by Blum, Feldman, and Micali [25]. Several works have refined and extended the notion in various

ways. We employ the definition of adaptive zero-knowledge for NIZK introduced by Feige, Lapidot, and Shamir [62] and we use the notion of simulation soundness introduced by Sahai [133]. The notion of simulation soundness is strengthened by De Santis et al. [135].

Definition 2.41 (NIPS). A triple $(p(\kappa), P, V)$ is said to be an efficient adaptive non-interactive proof system (NIPS) for a language $L \in \mathrm{NP}$ with witness relation R if $p(\kappa)$ is a polynomial and P and V are probabilistic polynomial time algorithms such that

1. $(x, w) \in R$ and $\xi \in \{0,1\}^{p(\kappa)}$ implies $V(x, P(x, w, \xi), \xi) = 1$.
2. For all computable functions A,
   $\Pr_{\xi \in \{0,1\}^{p(\kappa)}}[A(\xi) = (x, \pi) \wedge x \notin L \wedge V(x, \pi, \xi) = 1]$ is negligible in κ.

The string ξ plays the role of the public random bit-string in the definition. We suppress p in our notation of a NIPS and simply write (P, V). Loosely speaking a non-interactive zero-knowledge proof system is a NIPS, which is also zero-knowledge, but there are several flavors of zero-knowledge. We need a NIPS which is adaptive zero-knowledge (for a single statement) in the sense of Feige, Lapidot, and Shamir [62].

The following experiment formalizes an experiment where the adversary is given access to an oracle that either computes proofs following the protocol, or by invoking the zero-knowledge simulator, depending on if a bit b equal 0 or 1. Thus, the adversary chooses adaptively for which instances it wishes to see a proof of membership in the language. Finally, the adversary outputs a guess of the value of b. If the protocol is zero-knowledge the probability of a correct guess should be negligibly close to $1/2$ when b is chosen randomly.

We only require that proofs of valid statements can be simulated when the adversary knows the witness. Thus, the adversary is given access to oracles defined as follows. Denote by S' the machine that given $(x, w, \xi, \mathrm{simstate})$ as input outputs $S(x, \xi, \mathrm{simstate})$ if $(x, w) \in L$ and $\perp$ otherwise. Denote by P' the machine that given (x, w) as input outputs $P(x, w, \xi)$ if $(x, w) \in L$ and $\perp$ otherwise.

Experiment 2.42 (Adaptive Zero-Knowledge, $\mathsf{Exp}^{\mathsf{adind}-b}_{(P,V,S),A}(\kappa)$).

$$
\begin{array}{l}
\text{If} \quad (b = 0) \quad \text{do} \\
\qquad \xi \leftarrow \{0,1\}^{p(\kappa)} \\
\qquad d \leftarrow A^{P'(\cdot,\cdot,\xi)}(\xi) \\
\text{Else If} \quad (b = 1) \quad \text{do} \\
\qquad (\xi, \mathrm{simstate}) \leftarrow S(1^\kappa) \\
\qquad d \leftarrow A^{S'(\cdot,\cdot,\xi,\mathrm{simstate})}(\xi) \\
\text{End If}
\end{array}
$$

The experiment outputs d.

Definition 2.43 (Adaptive Zero-Knowledge). A NIPS (P, V) is adaptive zero-knowledge (NIZK) if there exists a polynomial time algorithm S such that for all $A \in \text{PPT}^*$ the probability $|\Pr[\mathsf{Exp}^{\mathsf{adind}-0}_{(P,V,S),A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{adind}-1}_{(P,V,S),A}(\kappa) = 1]|$ is negligible in κ.

We include S in our notation whenever it plays a role in the analysis, and omit it otherwise.

In cryptographic proofs one often performs hypothetic experiments where the adversary is run with simulated NIZKs. If the experiment simulates NIZKs to the adversary, the adversary could potentially gain the power to compute valid proofs of false statements. For a simulation sound NIZK this is not possible. Soundness holds even if the public string ξ is generated by the simulator and the adversary is allowed to see simulated proofs using this simulated ξ.

Experiment 2.44 (Simulation Soundness, $\mathsf{Exp}^{\mathsf{sims}}_{(P,V,S),A}(\kappa)$).

$$\begin{aligned}(\xi, \text{simstate}) &\leftarrow S(1^\kappa)\\ (x, \pi) &\leftarrow A^{S(\cdot,\xi,\text{simstate})}(\xi)\end{aligned}$$

Let Q be the set of proofs returned by the $S(\cdot, \xi, \text{simstate})$ oracle. Return 1 if $\pi \notin Q$, $x \notin L$, and $V(x, \pi, \xi) = 1$, and 0 otherwise.

Definition 2.45 (Simulation Soundness). A NIZK (P, V) with polynomial time simulator S for a language L is unbounded simulation sound if for all $A \in \text{PPT}^*$ the probability $\mathsf{Exp}^{\mathsf{sims}}_{(P,V,S),A}(\kappa)$ is negligible in κ.

In this thesis we abbreviate "adaptive zero-knowledge unbounded simulation sound efficient adaptive non-interactive proof" by NIZK.

2.10 The Random Oracle Model

In Section 2.2 we formalize two possible requirements on functions (or rather collections of functions). Broadly speaking, these requirements capture the fact that a function is unpredictable in some specific way. The most unpredictable function one can imagine is a randomly chosen function.

Sometimes it is not possible, or not known, how to prove the security of a cryptographic construct under complexity assumptions. This is often the case for efficient constructions. In such circumstances it is common to analyze the security in the random oracle model. This means that one, or several, of the hash functions used in the construction are modeled as randomly chosen functions. The security analysis is then carried out in this model. When the protocol is deployed the random oracles are replaced by some functions that are believed to be highly unpredictable such as the SHA-family [119].

The random oracle hypothesis, first explicitly stated in a paper by Bellare and Rogaway [18], says that if a construction is secure in the random oracle model, and

the function used when the protocol is deployed is "highly unpredictable" and chosen "independently of the protocol", then the protocol is secure even when the random function is replaced by the unpredictable function. This is not a mathematical statement. In fact, if it is turned into one it can be shown that the hypothesis is false [43] if interpreted literally. Thus, a protocol that is analyzed in the random oracle model can at best be heuristically secure.

On the other hand, all known counterexamples such as [43] are contrived. This is why many people trust the random oracle model even if it strictly speaking is false. In particular many people believe that constructing a signature scheme by applying the Fiat-Shamir heuristic to an identification scheme implies that the resulting signature scheme is in fact secure. We described the Fiat-Shamir heuristic informally in Section 1.4.3 and give a more formal description below. For an interesting discussion on these issues and provable security in general we refer the reader to Koblitz and Menezes [98] and Stinson [139].

2.10.1 Zero-Knowledge Proofs of Knowledge in the Random Oracle Model

One common use of the random oracle model is to prove the security of signature schemes constructed using the Fiat-Shamir heuristic [64].

This idea can be explained as follows. Let R be an NP-relation and suppose that one party holds a witness w of some joint input x such that $(x, w) \in R$. Let (P, V) be a C-Σ-protocol for the language L_R. Recall that such a protocol proceeds as follows. The prover computes a first message α and sends it to the verifier. Then the verifier chooses a challenge $c \in C$ randomly and sends it to the prover. Finally, the prover sends a reply d and the verifier verifies the triple (α, c, d). Fiat and Shamir's idea is to replace the challenge c with the output of a "cryptographic hash function" H. In other words, the prover computes α, but then instead of waiting for a challenge from the verifier it computes $c = H(x, \alpha)$. Finally, it computes d as usual. This gives the prover a triple (α, c, d) that it can send to the verifier. The verifier checks the triple by verifying the triple (α, c, d) as before and that $c = H(x, \alpha)$. Thus, the protocol is now non-interactive.

Note that although H may be "highly unpredictable" the output c is not independently chosen from α. Thus, the soundness of the C-Σ-protocol does not imply that the non-interactive version is sound. Furthermore, it is no longer zero-knowledge, but if we replace H by a randomly chosen function O, a so called random oracle, both soundness and zero-knowledge holds. It is assumed that the random oracle is available to both the verifier and the prover. We write $P^{\mathsf{O}(\cdot)}$ to denote the prover that computes c as $c = \mathsf{O}(x, \alpha)$ using a random oracle O. We write $V^{\mathsf{O}(\cdot)}$ for the verifier that verifies the triple (α, c, d) as is done in the original C-Σ-protocol, but also that $c = \mathsf{O}(x, \alpha)$.

Suppose that the prover wishes to sign a message m. To do that it computes $(\alpha, c, d) = P^{\mathsf{O}(m,\cdot)}$, i.e., it includes the message to be signed as a prefix to its random oracle. The signature is verified by checking that $V^{\mathsf{O}(m,\cdot)}(\alpha, c, d) = 1$. Thus, the

triple (α, c, d) may be viewed as a signature of m computed by the party holding a witness w such that $(x, w) \in R$. For this to make any sense it must of course be infeasible to find a witness w such that $(x, w) \in R$ given only x.

In Chapter 15 we analyze a hierarchical group signature scheme in the random oracle model, and use the above notation.

Chapter 3

Cryptographic Assumptions and Concrete Primitives

In this chapter we introduce some assumptions and concrete primitives we need to achieve our results. The reader should at least browse this chapter, since we modify some primitives slightly and introduce additional notation.

3.1 The Goldwasser-Micali Cryptosystem

Goldwasser and Micali [76] construct a public key cryptosystem based on the existence of non-approximable trapdoor predicates. This concept is captured in modern terminology as a hardcore bit of a family of trapdoor permutations.

The cryptosystem $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}} = (\mathsf{Kg}^{\mathsf{gm}}, E, D)$ of Goldwasser and Micali [76] using the family of trapdoor permutations $\mathcal{TPF}$ and hardcore bit $\mathcal{B}$ can be defined as follows. The key generator $\mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$ simply runs the permutation generator $(pk, sk) = \mathsf{Gen}(1^\kappa)$ of $\mathcal{TPF}$. To compute a cryptotext $E_{pk}(m)$ of a bit $m \in \{0,1\}$, a sample $r = \mathsf{Sample}(pk)$ is computed, and $(\mathsf{Eval}(pk, sk), \mathcal{B}(r) \oplus m)$ is the cryptotext. To decrypt a cryptotext (c, c'), we compute $D_{sk}(c, c') = \mathcal{B}(\mathsf{Invert}(t, c)) \oplus c'$. Goldwasser and Micali essentially show the following theorem.

Theorem 3.1. *If $\mathcal{TPF}$ is a trapdoor permutation family with hard-core bit $\mathcal{B}$, then $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ is polynomially indistinguishable.*

Goldreich and Levin [73] show how to construct a family of trapdoor permutations $\mathcal{TPF}$ with a hard-core bit $\mathcal{B}$ from any family of trapdoor permutations. Thus, we may take $\mathcal{B}$ above to be the Goldreich-Levin predicate.

To encrypt a bit-string the encryption function is invoked with a fresh randomly chosen r for each bit in the natural way. From Lemma 2.13 we know that this is as secure as the original scheme.

3.2 Assumptions On the Distribution of the Primes

In the cryptographic literature a prime p is said to be safe if $p = 2q+1$ with q prime. This is another way of saying that q is a Sophie Germain prime. For several reasons safe primes are particularly useful in the construction of cryptographic primitives. If we require that q also is a safe prime we end up with a chain of primes called a Cunningham chain.

Definition 3.2 (Cunningham Chain). A sequence $q_0, \ldots, q_{k-1}$ of primes is called a Cunningham Chain[1] of length k if $q_i = 2q_{i+1} + 1$ for $i = 0, \ldots, k-2$.

In the third part of this thesis the importance of such primes is illustrated. Before we start using Cunningham chains for cryptography we are obliged to ask if they exist at all and if they can be found efficiently. Unfortunately, there exists no proof that there are infinitely many Cunningham chains of any length, not even of length 2 which correspond to the Sophie Germain primes. One can apply a heuristic argument and assume that a randomly chosen integer n is prime with probability roughly $1/\ln n$. If we also assume that the event that $(n-1)/2$ is prime is independent of the event that n is prime for every prime a randomly chosen prime should give a Cunningham chain of length k with probability close to $1/\ln^k n$.

The assumption about independence is clearly false, but the heuristic argument is still very plausible in our setting and agrees with computational experiments. In the thesis we use Cunningham chains of length four. In practice it is not hard to find such chains for primes of the size used in current cryptography (cf. [128], [129]). Young and Yung [152] have also published some heuristic tricks for finding length-3 Cunningham chains. In the thesis we also use primes on the form $apq + 1$, where p and q are safe primes. Such primes also exist under similar plausible heuristic assumptions.

We make the following assumption.

Definition 3.3 (Distribution of Primes Assumption). The distribution of primes (DP) assumption states that

1. For each constant k exists constants c and κ_0 such that a random κ-bit prime q_0 defines a k-Cunningham chain $q_0, \ldots, q_{k-1}$ with probability at least κ^{-c} for $\kappa > \kappa_0$.

2. There exists constants c and κ_0 such that if p and q are random safe κ-bit primes, the probability that there exists a prime $apq + 1$ with a $\log_2 \kappa$-bit integer a is at least κ^{-c} for $\kappa > \kappa_0$.

We denote by $\mathsf{CunnGen}_k$ a polynomial-time algorithm that on input 1^κ outputs a Cunningham chain $q_0, \ldots, q_{k-1}$ of length k with overwhelming probability.

We remark that assumptions similar to the above are implicit in several papers in the cryptographic literature.

[1] This is a chain of the second kind. A chain of the first kind satisfies $q_i = 2q_{i+1} - 1$.

3.3 The Discrete Logarithm Assumption

The discrete logarithm assumption for some cyclic group G_n of order n with generator g says that given a random element $h \in G_n$ it is infeasible to compute the discrete logarithm $\log_g h$ of h in the basis g. Note that we by G_n denote a particular representation of a group. Thus, strictly speaking the discrete logarithm assumption is assumed to hold with regards to a specific representation. An interesting survey on the discrete logarithm assumption can be found in Odlyzko [118].

It is widely believed that solving discrete logarithms in a subgroup G_n of a multiplicative group $\mathbb{Z}_p^*$ modulo a prime p is hard if $|G_n|$ is a product of large distinct primes. Another example is to define G_n to be an elliptic curve group of order n. An introduction to elliptic curve based cryptography can be found in [23, 97].

It is possible to formalize the discrete logarithm assumption in general terms, but this is not the focus of this thesis. Thus, we define the assumption in a specific group. There seems to be no consensus on a formal definition of a "standard discrete logarithm assumption" in a subgroup of the multiplicative group modulo a prime. Thus, we take the liberty to simply call the specific assumption we define below "the discrete logarithm assumption" without any special qualifier.

Definition 3.4 (Discrete Logarithm Assumption). The discrete logarithm (DL) assumption states that the DP-assumption is true and the following.

1. Let $q_0, \ldots, q_k$ be a random length k Cunningham chain for a constant k, where q_0 is a κ-bit prime and let G_{q_i} be the unique subgroup of $\mathbb{Z}_{q_{i-1}}^*$ of order q_i for $i = 1, \ldots, k$. Let $g_i, h_i \in G_{q_i}$ be random elements. Then for $i = 1, \ldots, k$ and all adversaries $A \in \text{PPT}^*$ the probability $\Pr[A(q_i, g_i, h_i) = \log_{g_i} h_i]$ is negligible in κ.

2. Let p and q be random κ-bit safe primes such that $P = apq + 1$ is prime for some $\log_2 \kappa$-bit integer a. Let G_{pq} be the unique subgroup of $\mathbb{Z}_P^*$ of order pq, let $g, h \in G_{pq}$ be random elements. Then for all adversaries $A \in \text{PPT}^*$ the probability $\Pr[A(P, a, g, h) = \log_g h]$ is negligible in κ.

In each case the probability is taken over the random choice of prime, the random choice of h and the internal randomness of A.

In our security analyses we often ignore the inputs q_i and (P, a) to the adversary when they are clear from the context.

Before we prove the lemma we introduce a relation. We define R_{DL} to consist of the pairs $((g, h), x)$ such that $h = g^x$, where g and h are understood to belong to a group G_{q_i} or G_n generated as described in the assumption.

Lemma 3.5 (Non-trivial Representation). *Suppose that the DL-assumption holds, and let q_i be defined as in Definition 3.4. Choose $g_{i,1}, \ldots, g_{i,N} \in G_{q_i}$ randomly. Then for all $A \in \text{PPT}^*$, $\Pr[A(q_i, g_{i,1}, \ldots, g_{i,N}) = (\eta_1, \ldots, \eta_N) \neq 0 \wedge \prod_{j=1}^{N} g_{i,j}^{\eta_j} = 1]$ is negligible in κ.*

Proof. If the lemma is false there exists an adversary A, a constant c, and an infinite index set $\mathcal{N}$ such that the probability is greater than or equal to $1/\kappa^c$ for $\kappa \in \mathcal{N}$ and some $1 \leq i \leq k$. For simplicity we drop the i subscripts in the proof.

Consider the adversary A' defined as follows. It takes input (q, g, h) and chooses $j \in \{1, \ldots, N\}$ randomly. Then it sets $g_j = h$, and chooses $e_l \in \mathbb{Z}_q$ randomly and defines $g_l = g^{e_l}$ for $l \neq j$. Then it computes $(\eta_l)_{l=1}^N = A(G_q, g, (g_l)_{l=1}^N)$ and outputs $\frac{1}{-\eta_j} \sum_{l \neq j} e_l \eta_l$ if $\eta_j \neq 0$ and $(\eta_l)_{l=1}^N$ gives a non-trivial representation and $\perp$ otherwise.

If $\eta_j \neq 0$ we have $\prod_{l \neq j} g_l^{\eta_l} = h^{-\eta_j}$ and it follows that the output is the logarithm of h. The probability that $\eta_j \neq 0$ conditioned on that A outputs a non-trivial representation is at least $1/N$, since j is chosen uniformly at random and the distribution of $g_1, \ldots, g_N$ is independent of j. Thus, from independence follows that A' outputs $\log_g h$ with probability at least $\frac{1}{N\kappa^c}$, which contradicts the DL-assumption. □

3.4 The Chaum-van Heijst-Pfitzmann Hash Function

We employ the hash function $\mathcal{CHP} = (\mathsf{CHPg}, D^{\mathsf{CHP}}, H^{\mathsf{CHP}})$ introduced by Chaum, van Heijst, and Pfitzmann [47]. The function sampling algorithm CHPg takes as input the representation of a prime q such that $2q + 1$ is prime and $\delta \in \mathbb{N}$ and outputs a list $(q, h_1, \ldots, h_\delta)$ where $(h_1, \ldots, h_\delta) \in G_q^\delta$ are randomly chosen elements. The domain sampling algorithm D^{CHP} takes as input (q, δ) and outputs a random element in $\mathbb{Z}_q^\delta$. The evaluation algorithm H^{CHP} takes input $(q, h_1, \ldots, h_\delta)$ and $(z_1, \ldots, z_\delta)$ and outputs $\prod_{l=1}^\delta h_l^{z_l}$. The following proposition is given in [47], but it is also an immediate consequence of Lemma 3.5 above.

Proposition 3.6. *Let k be a constant and define $\mathsf{CHPg}_{k,i}$ to be the algorithm that on input $(1^\kappa, \delta)$ computes $(q_0, \ldots, q_{k-1}) = \mathsf{CunnGen}_k(1^\kappa)$ and outputs $\mathsf{CHPg}(q_i, \delta)$.*

Then the collection of functions $(\mathsf{CHPg}_{k,i}, D^{\mathsf{CHP}}, H^{\mathsf{CHP}})$ is one-way and collision-free under the DL-assumption.

We abuse notation and use H^{CHP} to denote the map computed by H^{CHP} on input $(h_1, \ldots, h_\delta)$ and also to denote the list $(h_1, \ldots, h_\delta)$. Thus, we think of H^{CHP} as a function defined by $H^{\mathsf{CHP}}(z_1, \ldots, z_\delta) = \prod_{l=1}^\delta h_l^{z_l}$ and represented by $(h_1, \ldots, h_\delta)$.

3.5 The Decision Diffie-Hellman Assumption

The Decision Diffie-Hellman (DDH) assumption states that it is infeasible to distinguish $(g^\alpha, g^\beta, g^{\alpha\beta})$ from $(g^\alpha, g^\beta, g^\gamma)$ when $\alpha, \beta, \gamma \in \mathbb{Z}_q$ are randomly chosen and g is the generator of a group G_q. This is a decisional version of the Diffie-Hellman assumption [60] which states that it is infeasible to compute $g^{\alpha\beta}$ given only g^α and g^β for randomly chosen $\alpha, \beta \in \mathbb{Z}_q$. The DDH-assumption is equivalent to the security of the El Gamal cryptosystem, which is introduced in the next section. Currently

the best method to solve the decision Diffie-Hellman problem is to solve the discrete logarithm problem, but no proof of equivalence between the two problems is known.

Definition 3.7 (Decision Diffie-Hellman Assumption). The decision Diffie-Hellman assumption states that the DP-assumption is true and the following. Let $q_0, \ldots, q_k$ be a random length k Cunningham chain for a constant k, where q_0 is a κ-bit prime and let G_{q_i} be the unique subgroup of $\mathbb{Z}^*_{q_{i-1}}$ of order q_i for $i = 1, \ldots, k$. Let $g_i \in G_{q_i}$ and $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}_{q_i}$ be random elements. Then for $i = 1, \ldots, k$ and all adversaries $A \in \mathrm{PPT}^*$ the absolute value

$$|\Pr[A(q_i, g_i, g_i^{\alpha_i}, g_i^{\beta_i}, g_i^{\alpha_i\beta_i}) = 1] - \Pr[A(q_i, g_i, g_i^{\alpha_i}, g_i^{\beta_i}, g_i^{\gamma_i}) = 1]|$$

is negligible in κ.

In our security analyses we often omit the inputs q_i and g_i when they are clear from the context. In the second part of the thesis we also use a variant of the DDH-problem captured below.

Lemma 3.8 (Variant DDH-Assumption). *Suppose that the DDH-assumption is true, and let q_i be defined as in Definition 3.7. Let $\alpha_i, \beta_i, \beta_i', \gamma_i, \gamma_i' \in \mathbb{Z}_{q_i}$ be randomly chosen. Then for $i = 1, \ldots, k$ and all adversaries $A \in \mathrm{PPT}^*$ the absolute value*

$$|\Pr[A(g_i^{\alpha_i}, g_i^{\beta_i}, g_i^{\alpha_i\beta_i}, g_i^{\beta_i'}, g_i^{\alpha_i\beta_i'}) = 1] - \Pr[A(g_i^{\alpha_i}, g_i^{\beta_i}, g_i^{\gamma_i}, g_i^{\beta_i'}, g_i^{\gamma_i'}) = 1]|$$

is negligible in κ.

Proof. We drop the subscript i, since the proof for each i is essentially identical. Suppose that the lemma is false. Then there exists an adversary $A \in \mathrm{PPT}^*$, a constant $c > 0$, and an infinite index set $\mathcal{N}$ such that

$$|\Pr[A(g^\alpha, g^\beta, g^\gamma, g^{\beta'}, g^{\gamma'}) = 1] - \Pr[A(g^\alpha, g^\beta, g^{\alpha\beta}, g^{\beta'}, g^{\alpha\beta'}) = 1]| \geq \frac{1}{\kappa^c} \ .$$

This implies that one of the following inequalities hold

$$\begin{aligned} |\Pr[A(g^\alpha, g^\beta, g^\gamma, g^{\beta'}, g^{\gamma'}) = 1] - \Pr[A(g^\alpha, g^\beta, g^\gamma, g^{\beta'}, g^{\alpha\beta'}) = 1]| &\geq \frac{1}{2\kappa^c} \\ |\Pr[A(g^\alpha, g^\beta, g^\gamma, g^{\beta'}, g^{\alpha\beta'}) = 1] - \Pr[A(g^\alpha, g^\beta, g^{\alpha\beta}, g^{\beta'}, g^{\alpha\beta'}) = 1]| &\geq \frac{1}{2\kappa^c} \ . \end{aligned}$$

The former is impossible, since given a triple (u, v, w), the tuple $(u, g^\beta, g^\gamma, v, w)$ for random $\beta, \gamma \in \mathbb{Z}_q$ is identically distributed to the input to A in the right or left probability in the first equation depending on if (u, v, w) is a DDH-triple or not. The latter is impossible, since given a triple (u, v, w), the tuple $(u, v, w, g^{\beta'}, u^{\beta'})$, for a random $\beta' \in \mathbb{Z}_q$, is identically distributed to the input to A to the left or right probability in the second equation depending on if (u, v, w) is a random triple or if it is a DDH-triple. Thus, the lemma is true. □

3.6 The El Gamal Cryptosystem

The El Gamal [71] public key cryptosystem can be defined in any cyclic group, but we consider only groups G_q as defined above, i.e., $p = 2q + 1$ and G_q is the unique subgroup of order q in $\mathbb{Z}_p^*$.

We write $\mathsf{CSKg}^{\mathsf{elg}}$ for the key generation algorithm that takes as input a prime q such that $2q + 1$ is prime and a generator g in G_q. It then chooses a random private key $x \in \mathbb{Z}_q$, computes a public key (g, y), where $y = g^x$, and outputs $(q, (g, y), x)$. The encryption and decryption algorithm below are given q as part of their input, but this is omitted throughout the thesis to simplify notation. When invoked on a public key (g, y) and message $m \in G_q$ the encryption algorithm chooses $r \in \mathbb{Z}_q$ randomly and outputs $(g^r, y^r m)$. We denote this $E_{(g,y)}(m, r) = (g^r, y^r m)$, or $E_y(m, r) = (g^r, y^r m)$ when g is fixed. Sometimes we also write $E_{(g,y)}(m)$, when we do not care about the random input. To decrypt a cryptotext (u, v) using the private key x the decryption algorithm outputs vu^{-x}. We denote this $D_x(u, v) = vu^{-x}$.

Note that the cleartext m must be contained in G_q. Thus, to encrypt an arbitrary bit-string there must exist an efficient algorithm that encodes an arbitrary fixed-size bit-string as an element in G_q. There must of course also exist an efficient way to recover the message from the encoding. The group we use is in fact equal to the subgroup of squares in $\mathbb{Z}_p^*$, or differently phrased the quadratic residues modulo p. To encode a bit-string $s \in \{0, 1\}^{\kappa - t - 1}$ into an element in G_q with $\log_2 q = \kappa$ we repeatedly choose $r \in [0, 2^t - 1]$ randomly and check if $2^{\kappa - t} r + s$ is a quadratic residue. Checking for quadratic residuosity can be done efficiently [102]. It is of course easy to decode an element as a bit-string. In practice the encoding works well, since heuristically we expect that the probability that $2^{\kappa - t} r + s$ is a quadratic residuosity should be roughly $1/2$ for each s.

We are not aware of any encoding that can be analyzed rigorously, so strictly speaking the message space of the El Gamal cryptosystem in G_q can not be taken to be the set $\{0, 1\}^{\kappa - t}$ for some small t. A similar problem is encountered if G_q is taken to be an elliptic curve. This is not a problem in practice and we ignore this issue in the remainder of the thesis.

An interesting property of the El Gamal cryptosystem is that it is homomorphic. This means that if $(u_0, v_0) = E_{(g,y)}(m_0, r_0)$ and $(u_1, v_1) = E_{(g,y)}(m_1, r_1)$ then $(u_0 u_1, v_0 v_1) = E_{(g,y)}(m_0 m_1, r_0 + r_1)$. This is a straightforward consequence of the definition, but it implies that a cryptotext (u, v) can be re-encrypted by computing (ug^r, vy^r). All El Gamal based mix-nets in the literature are based on this observation, but in the second part of this thesis we present an alternative.

The following proposition is almost immediate. A proof is given in Tsiounis and Yung [142].

Proposition 3.9. *Let k be a constant and define $\mathsf{CSKg}^{\mathsf{elg}}_{k,i}$ to be the algorithm that on input 1^κ computes $(q_0, \ldots, q_{k-1}) = \mathsf{CunnGen}_k(1^\kappa)$, chooses $g_i \in G_{q_i}$ randomly and outputs $\mathsf{CSKg}^{\mathsf{elg}}(q_i, g_i)$.*

Then the El Gamal cryptosystem $(\mathsf{CSKg}^{\mathsf{elg}}_{k,i}, E, D)$ *is polynomially indistinguishable under the DDH-assumption.*

3.7 The Cramer-Shoup Cryptosystem

Cramer and Shoup [53] introduced the first practical CCA2-secure cryptosystem based on a standard assumptions. The cryptosystem $\mathcal{CS}^{\mathsf{cs}} = (\mathsf{Kg}^{\mathsf{cs}}, E^{\mathsf{cs}}, D^{\mathsf{cs}})$, is defined as follows.

Let (I, F), with $F = \{f_i\}_{i\in I}$, be a collision-free collection of functions with corresponding algorithms $\mathcal{CF} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval})$. Assume that $D_i \supset G_q \times G_q$ and that $f_i(D_i) \subset \mathbb{Z}_q$ for all $i \in I \cap \{0,1\}^{\log_2 q}$. An example of this is the Chaum-van Heyst-Pfitzmann function in Section 3.4 with suitable modified security parameter.

The key generation algorithm $\mathsf{Kg}^{\mathsf{cs}}_{\mathcal{CF}}$ takes as input a prime q such that $2q+1$ is prime. It generates random $g_1, g_2 \in G_q$ and $x_1, x_2, y_1, y_2, z \in \mathbb{Z}_q$ and computes $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. Then it computes $i = \mathsf{Gen}(\kappa)$ and outputs $((i, q, g_1, g_2, c, d, h), (i, q, x_1, x_2, y_1, y_2, z))$. Encryption of a message $m \in G_q$ using the public key $Y = (i, q, g_1, g_2, c, d, h)$ and randomness $r \in \mathbb{Z}_q$ is given by

$$E^{\mathsf{cs}}_Y(m, r) = (u, \mu, v, \nu) = (g_1^r, g_2^r, h^r m, c^r d^{r\mathsf{Eval}(i,u,\mu,v)}) \ .$$

Note that (u, v) is an El Gamal cryptotext of the message m using the El Gamal public key (g_1, h), so decryption of a cryptotext (u, μ, v, ν) using the private key $X = (i, q, x_1, x_2, y_1, y_2, z)$ is given by $D^{\mathsf{cs}}_X(u, \mu, v, \nu) = D_z(u, v) = m$ for valid cryptotexts. A cryptotext is considered valid if the predicate

$$T^{\mathsf{cs}}_X(u, \mu, v, \nu) = (u^{x_1 + x_2 \mathsf{Eval}(i,u,\mu,v)} \mu^{y_1 + y_2 \mathsf{Eval}(i,u,\mu,v)} = \nu)$$

is satisfied. An invalid cryptotext decrypts to $\perp$. Throughout the thesis we abuse notation and omit i and q from the public and private keys when they are clear from the context.

Theorem 3.10. *Let k be a constant and define $\mathsf{Kg}^{\mathsf{cs}}_{k,i,\mathcal{CF}}$ to be the algorithm that on input 1^κ computes $(q_0, \ldots, q_{k-1}) = \mathsf{CunnGen}_k(1^\kappa)$ and outputs $\mathsf{Kg}^{\mathsf{cs}}_{\mathcal{CF}}(q_i)$.*

The Cramer-Shoup cryptosystem $(\mathsf{Kg}^{\mathsf{cs}}_{k,i,\mathcal{CF}}, E^{\mathsf{cs}}, D^{\mathsf{cs}})$ is CCA2-secure under the DDH-assumption if $\mathcal{CF}$ is collision-free.

A proof is given in [53]. Note that Proposition 3.6, and the obvious fact that the first statement of the DL-assumption is true if the DDH-assumption is true, imply that the cryptosystem is secure under the DDH-assumption if we instantiate the collision-free hash function $\mathcal{CF}$ with the Chaum-van Heijst-Pfitzmann hash function $\mathcal{CHP}$.

3.8 The Strong RSA-Assumption

The strong RSA-assumption says that it is infeasible to compute any non-trivial root of a random element in $\mathrm{QR}_{\mathbf{N}}$ where $\mathbf{N}$ is an RSA-modulus, even if allowed

to select which root to compute. This assumption was first considered by Barić and Pfitzmann [11] and differs from the standard RSA-assumption in that the root to compute is not predetermined. Currently the fastest known method to solve this problem is to factor $\mathbf{N}$, but it is not known if the strong RSA-assumption is equivalent to the factoring assumption.

Definition 3.11 (Strong RSA-Assumption). Let $\mathbf{p}$ and $\mathbf{q}$ be randomly chosen $\kappa/2$-bit primes, define $\mathbf{N} = \mathbf{pq}$, and let $\mathbf{g} \in \mathrm{QR}_{\mathbf{N}}$ be randomly chosen. The strong RSA-assumption states that for all adversaries $A \in \mathrm{PPT}^*$ the probability $\Pr[A(\mathbf{N}, \mathbf{g}) = (\mathbf{b}, e) \wedge e \neq \pm 1 \wedge \mathbf{b}^e = \mathbf{g} \bmod \mathbf{N}]$ is negligible in κ.

It is easy to see that if the DP-assumption and and the strong RSA-assumption are true, then the strong RSA-assumption is still true if $\mathbf{p}$ and $\mathbf{q}$ are randomly chosen safe primes such that there exists a $\log_2 \kappa$-bit integer a such that $a\mathbf{pq} + 1$ is prime, since this happens with non-negligible probability for random primes. This is the setting considered in this thesis.

To simplify some of the proofs in the thesis we prove a useful lemma that give alternative ways to view the strong RSA-assumption. The proof of the lemma below follows the proof in Damgård and Fujisaki [55], but our lemma is slightly stronger. In their analysis it is essential that the bit-size of η_0 is smaller than $\kappa/2$. We show that this restriction is not necessary.

Lemma 3.12 (Variants of Strong RSA-Assumption). *Assume the strong RSA-assumption. Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be randomly chosen* $\kappa/2$*-bit safe primes, define* $\mathbf{N} = \mathbf{pq}$*, and let* $\mathbf{g}, \mathbf{h} \in \mathrm{QR}_{\mathbf{N}}$ *be random. Then for all adversaries* $A \in \mathrm{PPT}^*$ *the probabilities*

$$\begin{aligned}
&\Pr[A(\mathbf{N}, \mathbf{g}, \mathbf{h}) = (\mathbf{b}, \eta_0, \eta_1, \eta_2) \wedge \eta_0 \neq 0 \wedge (\eta_0 \nmid \eta_1 \vee \eta_0 \nmid \eta_2) \\
&\quad \wedge \mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{h}^{\eta_2} \bmod \mathbf{N}] \\
&\Pr[A(\mathbf{N}, \mathbf{g}, \mathbf{h}) = (\mathbf{b}, \eta_1, \eta_2) \wedge (\eta_1, \eta_2) \neq (0, 0) \wedge \mathbf{g}^{\eta_1} = \mathbf{h}^{\eta_2} \bmod \mathbf{N}]
\end{aligned}$$

are negligible in κ*.*

Before we prove the lemma we introduce a relation. We define R_{SRSA} to consist of the pairs $((\mathbf{N}, \mathbf{g}, \mathbf{h}), (\mathbf{b}, \eta_0, \eta_1, \eta_2))$ such that either $\eta_0 \nmid \eta_1$ or $\eta_0 \nmid \eta_2$ and $\mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{h}^{\eta_2}$, or $\eta_0 = 0$ and $(\eta_1, \eta_2) \neq (0, 0)$ and $\mathbf{g}^{\eta_1} = \mathbf{h}^{\eta_2}$, or $\eta_0 \mid \mathbf{N}$ and $|\eta_0| < \mathbf{N}$.

Proof of Lemma 3.12. Denote by extgcd the extended Euclidean algorithm, i.e., given input (η_0, η_1) it outputs a tuple of integers (f, a, b), where $f = \gcd(\eta_0, \eta_1)$ and $f = a\eta_0 + b\eta_1$.

Suppose that there exists an adversary $A \in \mathrm{PPT}^*$, a constant c, and an infinite index set $\mathcal{N}$ such that

$$\begin{aligned}
&\Pr[A(\mathbf{N}, \mathbf{g}, \mathbf{h}) = (\mathbf{b}, \eta_0, \eta_1, \eta_2) \wedge \eta_0 \neq 0 \wedge (\eta_0 \nmid \eta_1 \vee \eta_0 \nmid \eta_2) \\
&\quad \wedge \mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{h}^{\eta_2} \bmod \mathbf{N}]
\end{aligned}$$

for $\kappa \in \mathcal{N}$. Consider the adversary A' to the strong RSA-experiment defined as follows. Denote by κ_r an additional security parameter large enough to make $2^{-\kappa_r}$ negligible. The adversary A' accepts $(\mathbf{N}, \mathbf{g})$ as input, chooses $e \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly and defines $\mathbf{h} = \mathbf{g}^e \bmod \mathbf{N}$. Then it computes $(\mathbf{b}, \eta_0, \eta_1, \eta_2) = A(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(f, a, b) = \text{extgcd}(\eta_0, \eta_1 + e\eta_2)$. Finally, it outputs $(\mathbf{g}^a\mathbf{b}^b, \eta_0/f)$. Note that

$$(\mathbf{g}^a\mathbf{b}^b)^{\eta_0/f} = (\mathbf{g}^{a\eta_0}\mathbf{g}^{b(\eta_1+e\eta_2)})^{1/f} = \mathbf{g}^{(a\eta_0+b(\eta_1+e\eta_2))/f} = \mathbf{g} \ .$$

Thus, we must argue that $f \neq \pm\eta_0$ with non-negligible probability.

We analyze the output conditioned on the event that the output of A has the property that $\eta_0 \neq 0$ and that η_0 does not divide both η_1 and η_2 and that $\mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{h}^{\eta_2}$. We argue that for any fixed $(\mathbf{h}, \eta_0, \eta_1, \eta_2)$ the probability that $\eta_0 \nmid (\eta_1 + e\eta_2)$ is at least $1/4$ over the random choice of e, conditioned on $\mathbf{h} = \mathbf{g}^e$.

To start with we note that if $\eta_0 \mid (\eta_1 + e\eta_2)$ and $\eta_0 \mid \eta_2$ then clearly $\eta_0 \mid \eta_1$ as well, which is a contradiction. Thus, if $\eta_0 \mid \eta_2$, the probability that $\eta_0 \nmid (\eta_1 + e\eta_2)$ is one. Consider now the case where $\eta_0 \nmid \eta_2$.

Define $\mathbf{p}' = (\mathbf{p} - 1)/2$, $\mathbf{q}' = (\mathbf{q} - 1)/2$, and $t = \mathbf{p}'\mathbf{q}'$. We argue that there exists a prime r such that $r \mid \eta_0$ and $\gcd(r, t) = 1$. If this is not the case we may assume that η_0 is on the form $\pm(\mathbf{p}')^a(\mathbf{q}')^b$ for some natural numbers a and b. If $a, b > 0$, then we have $\mathbf{g}^{\eta_0+1} = \mathbf{g}$ and we could have defined A to simply output $(\mathbf{g}, \eta_0 + 1)$. If $a = 0$ (or $b = 0$), then we could have defined A to simply take the lth root of η_0 for a polynomially bounded number of l, check for primality, and thus extract $\mathbf{p}'$ (or $\mathbf{q}'$). Finally, it could compute $t = \mathbf{p}'(\mathbf{N}/(2\mathbf{p}' + 1) - 1)/2$ and output $(\mathbf{g}, t)$. We conclude that there exists a prime r such that $r \mid \eta_0$ and $\gcd(r, t) = 1$ with overwhelming probability.

Let r^i such that $r^i \mid \eta_0$ but $r^i \nmid \eta_2$. It follows from the Chinese remainder theorem that

$$\Pr_e[\eta_0 \mid (\eta_1 + e\eta_2) \ \mid \ \mathbf{h} = \mathbf{g}^e] \leq \Pr[\eta_1 + e\eta_2 = 0 \bmod r^i \ \mid \ \mathbf{h} = \mathbf{g}^e] \ .$$

We write $e = e't + (e \bmod t)$. Since $\gcd(t, r^i) = 1$ we know that t is a generator in $\mathbb{Z}_{r^i}$. This implies that

$$\begin{aligned} &\Pr_e[\eta_1 + e\eta_2 = 0 \bmod r^i \mid \mathbf{h} = \mathbf{g}^e] \\ &= \Pr_e[\eta_1 + (e \bmod t)\eta_2 + e't = 0 \bmod r^i \mid \mathbf{h} = \mathbf{g}^e] \ . \end{aligned}$$

Note that $\eta_1 + (e \bmod t)\eta_2$ is constant for a fixed $(\mathbf{h}, \eta_0, \eta_1, \eta_2)$, but since t is a generator in $\mathbb{Z}_{r^i}$ the probability that $e't$ takes on any given value is at most $1/r^i + 1/\kappa_r \leq 3/4$. It follows that A' outputs an RSA-root with probability at least $\frac{1}{4}\kappa^{-c}$.

Suppose that there exists an adversary $A \in \text{PPT}^*$, a constant c, and an infinite index set $\mathcal{N}$ such that

$$\Pr[A(\mathbf{N}, \mathbf{g}, \mathbf{h}) = (\eta_1, \eta_2) \wedge (\eta_1, \eta_2) \neq (0, 0) \wedge \mathbf{g}^{\eta_1} = \mathbf{h}^{\eta_2} \bmod \mathbf{N}]$$

for $\kappa \in \mathcal{N}$. Consider the adversary A' defined as follows. On input $(\mathbf{N}, \mathbf{g})$ it chooses $e \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly and defines $\mathbf{h} = \mathbf{g}^e \bmod \mathbf{N}$. Then it chooses $d \in \{0, 1\}$ randomly and defines

$$(\mathbf{g}', \mathbf{h}') = (\mathbf{g}^d \mathbf{h}^{1-d}, \mathbf{g}^{1-d} \mathbf{h}^d) \ ,$$

and computes $(\eta_1, \eta_2) = A(\mathbf{N}, \mathbf{g}', \mathbf{h}')$. If $\eta_1 = \pm\eta_2$ it outputs $(\mathbf{g}, \eta_1 + 1)$. Otherwise it computes $(f, a, b) = \text{extgcd}(\eta_1, \eta_2)$ and outputs

$$((\mathbf{g}')^b (\mathbf{h}')^a, (d\eta_1 + (1-d)\eta_2)/f) \ .$$

If $\eta_1 = \pm\eta_2 \neq 0$ and $(\mathbf{g}')^{\eta_1} = (\mathbf{h}')^{\eta_2}$ then $(\mathbf{h}'/\mathbf{g}')^{\eta_1} = 1$ or $(\mathbf{h}'\mathbf{g}')^{\eta_1} = 1$. Note that the probability that $\mathbf{g}'/\mathbf{h}'$ or $\mathbf{g}'\mathbf{h}'$ does not generate $\text{QR}_\mathbf{N}$ is negligible. This means that η_1 is a multiple of $t = (\mathbf{p} - 1)(\mathbf{q} - 1)/4$, and we have $\mathbf{g}^{\eta_1 + 1} = \mathbf{g}$. We conclude that the probability that $\eta_1 = \pm\eta_2 \neq 0$ and $(\mathbf{g}')^{\eta_1} = (\mathbf{h}')^{\eta_2}$ is negligible.

Suppose that $\eta_1 \neq \pm\eta_2$ and $(\mathbf{g}')^{\eta_1} = (\mathbf{h}')^{\eta_2}$. Then we have

$$((\mathbf{g}')^b (\mathbf{h}')^a)^{(d\eta_2 + (1-d)\eta_1)/f} = \begin{cases} (\mathbf{h}^b \mathbf{g}^a)^{\eta_1/f} = \mathbf{g}^{(a\eta_1 + b\eta_2)/f} = \mathbf{g} & \text{if } d = 0 \\ (\mathbf{g}^b \mathbf{h}^a)^{\eta_2/f} = \mathbf{g}^{(a\eta_1 + b\eta_2)/f} = \mathbf{g} & \text{if } d = 1 \end{cases} \ .$$

We observe that the distributions of the conditional random variables $(\mathbf{N}, \mathbf{g}', \mathbf{h}' \mid d = 0)$ and $(\mathbf{N}, \mathbf{g}', \mathbf{h}' \mid d = 1)$ are statistically close. Since d is randomly chosen we conclude that the probability that $(d\eta_1 + (1-d)\eta_2)/f$ equals one is at most $3/4$. Thus, A' outputs a non-trivial RSA-root with probability at least $\frac{1}{4}\kappa^{-c}$. This is a contradiction, so the second probability in the lemma is negligible. □

3.8.1 A Simplifying Convention

Consider any computation involving an RSA-modulus $\mathbf{N}$ where the inverse of an element $a \in \mathbb{Z}_\mathbf{N}$ must be computed. In principle, it could happen that a is not a unit in $\mathbb{Z}_\mathbf{N}$. However, if such an element is encountered with non-negligible probability in a computation where the factorization of $\mathbf{N}$ is not known, we have of course found one of the factors of $\mathbf{N}$ and the strong RSA-assumption is broken.

To simplify the exposition of the protocols and their analysis we assume, without loss, that all elements in $\mathbb{Z}_\mathbf{N}$ that appear in the simulations in the security analyses always can be inverted.

3.9 The Shamir Hash Function

Consider the collection of functions $(I_{\mathsf{Sh}}, F_{\mathsf{Sh}})$ defined as follows. Let I_{Sh} be the set of pairs $(\mathbf{N}, \mathbf{g})$, where $\mathbf{N}$ is a product of two safe primes of the same bit-size and $\mathbf{g}$ is a generator of $\text{QR}_\mathbf{N}$, and define $F_{\mathsf{Sh}} = \{f_{(\mathbf{N},\mathbf{g})} : D_{(\mathbf{N},\mathbf{g})} \to \{0,1\}^*\}$ by setting $D_{(\mathbf{N},\mathbf{g})} = \{0,1\}^{4\kappa}$ and $f_{(\mathbf{N},\mathbf{g})}(x) = \mathbf{g}^x \bmod \mathbf{N}$, where $\log_2 \mathbf{N} = \kappa$. The three algorithms $(\mathsf{Shg}, D^{\mathsf{Sh}}, H^{\mathsf{Sh}})$ required by the definition of a polynomial collection of functions are defined in the obvious way.

The idea to use this construction as a collision-free hash function was proposed by Shamir. To simplify the exposition we omit $\mathbf{N}$ and $\mathbf{g}$ from our notation when they are clear from the context.

Lemma 3.13. *The function collection $\mathcal{SH} = (\mathsf{Shg}, D^{\mathsf{Sh}}, H^{\mathsf{Sh}})$ is collision-free under the strong RSA-assumption.*

Proof. Suppose that there exists an adversary $A \in \mathrm{PPT}^*$ such that

$$\Pr[A(\mathbf{N}, \mathbf{g}) = (x_1, x_2) \wedge x_1 \neq x_2 \wedge \mathbf{g}^{x_1} = \mathbf{g}^{x_2}]$$

is non-negligible. Then we can define A' to be the adversary that on input $(\mathbf{N}, \mathbf{g})$ computes $(x_1, x_2) = A(\mathbf{N}, \mathbf{g})$ and defines $(\mathbf{b}, \eta)$ equal to $(\mathbf{g}, x_1 - x_2 + 1)$ or $(\mathbf{g}, x_2 - x_1 + 1)$ depending on if $x_1 > x_2$ or $x_1 < x_2$ respectively. Finally, A' outputs $(\mathbf{b}, \eta)$. Note that this implies that $\eta \neq \pm 1$ and $\mathbf{b}^\eta = 1$ with non-negligible probability and A' breaks the strong RSA-assumption. □

Remark 3.14. The Shamir hash function is in fact secure under the factoring assumption, but in our application we need the strong RSA-assumption anyway. Thus, there is little point in introducing another assumption and proving a stronger result.

3.10 The Cramer-Shoup Signature Scheme

Cramer and Shoup [54] introduce a signature scheme based on the strong RSA-assumption. We describe a slightly modified scheme. The Cramer-Shoup signature scheme $\mathcal{SS}^{\mathsf{cs}}_{\mathcal{CF}_1,\mathcal{CF}_2} = (\mathsf{SSKg}^{\mathsf{cs}}_{\mathsf{Gen}_1,\mathsf{Gen}_2}, \mathsf{Sig}^{\mathsf{cs}}_{\mathsf{Eval}_1,\mathsf{Eval}_2}, \mathsf{Vf}^{\mathsf{cs}}_{\mathsf{Eval}_1,\mathsf{Eval}_2})$ is defined as follows, where $\mathcal{CF}_1 = (\mathsf{Gen}_1, \mathsf{Sample}_1, \mathsf{Eval}_1)$ and $\mathcal{CF}_2 = (\mathsf{Gen}_2, \mathsf{Sample}_2, \mathsf{Eval}_2)$ are collision-free collections of functions.

On input 1^κ the key generation algorithm $\mathsf{SSKg}^{\mathsf{cs}}_{\mathsf{Gen}_1,\mathsf{Gen}_2}$ first chooses two $\kappa/2$-bit safe primes $\mathbf{p}$ and $\mathbf{q}$ randomly such that there exists a $\log_2 \kappa$-bit integer a such that $a\mathbf{pq} + 1$ is prime and defines $\mathbf{N} = \mathbf{pq}$, $\mathbf{p}' = (\mathbf{p} - 1)/2$, and $\mathbf{q}' = (\mathbf{q} - 1)/2$. Then it chooses $\mathbf{h}, \mathbf{z} \in \mathrm{QR}_{\mathbf{N}}$ and a $(\kappa + 1)$-bit prime e' such that $e' = 1 \bmod 4$ randomly. Finally, it computes $i_1 = \mathsf{Gen}_1(1^\kappa)$ and $i_2 = \mathsf{Gen}_2(1^\kappa)$ and outputs $((i_1, i_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e'), (i_1, i_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}'))$. We also assume that it on input $(1^\kappa, i_1)$ uses this value of i_1 instead of generating it.

The signature algorithm $\mathsf{Sig}^{\mathsf{cs}}_{\mathsf{Eval}_1,\mathsf{Eval}_2}$ takes as input a message m and a private key $(i_1, i_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}')$ and outputs a signature $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$ computed as follows. The algorithm chooses a random $(\kappa + 1)$-bit prime e such that $e = 3 \bmod 4$ and a random $\boldsymbol{\sigma}' \in \mathrm{QR}_{\mathbf{N}}$ and computes

$$\mathbf{z}' = (\boldsymbol{\sigma}')^{e'} \mathbf{h}^{-\mathsf{Eval}_1(i_1, m)} \quad , \quad \text{and} \quad \boldsymbol{\sigma} = \left(\mathbf{z}\mathbf{h}^{\mathsf{Eval}_2(i_2, \mathbf{z}')}\right)^{1/e} \quad .$$

The verification algorithm $\mathsf{Vf}^{\mathsf{cs}}_{\mathsf{Eval}_1,\mathsf{Eval}_2}$ takes as input a message m, candidate signature $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$, and $(i_1, i_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e')$ and verifies the signature as follows. It verifies

that $e \neq e'$ and that it is an odd integer with at least $(\frac{3}{2}\kappa - 4)$ and at most $(\kappa + 1)$ bits. Then it computes $\mathbf{z}' = (\boldsymbol{\sigma}')^{e'} \mathbf{h}^{-\mathsf{Eval}_1(i_1, m)}$ and verifies that $\mathbf{z} = \boldsymbol{\sigma}^e \mathbf{h}^{-\mathsf{Eval}_2(i_2, \mathbf{z}')}$. If so it outputs 1 and otherwise 0.

We have modified the original scheme slightly by making e' always equal to 1 modulo 4 and the primes e generated at signing equal to 3 modulo 4. This makes it easier to prove later in zero-knowledge that $e \neq e'$. In the original scheme $\mathcal{CF}_1$ and $\mathcal{CF}_2$ are equal, but in our setting they will be different. This does not affect the security proof in any way.

Also in our description the exponent e is longer than the modulus, but in the original description e is shorter than $\mathbf{p}'$ and $\mathbf{q}'$. Below we show that the security proof still holds.

Theorem 3.15. *The signature scheme $\mathcal{SS}^{\mathsf{cs}}_{\mathcal{CF}_1, \mathcal{CF}_2}$ is CMA-secure under the strong RSA-assumption and assuming that $\mathcal{CF}_1$ and $\mathcal{CF}_2$ are collision-free.*

Proof. We assume familiarity with [54]. When the length of the exponent is between $\kappa + 1$ bits and $\frac{3}{2}\kappa - 4$ bits, the proof of [54] holds except for how a Type III forger is used to break the strong RSA-assumption, where a Type III forger is defined as a forger that outputs a signature (using our notation) $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma'})$ such that $e \neq e_i$ for all signatures e_i it has seen previously with non-negligible probability.

Let us recall their simulator. It accepts an RSA-modulus $\mathbf{N}$ and a generator $\mathbf{g} \in \mathrm{QR}_{\mathbf{N}}$ as input and chooses a polynomial number of primes e_i distributed as the prime chosen in the computation of a signature, and defines $\mathbf{h} = \mathbf{g}^{2e' \prod_i e_i}$. These primes are used in the simulation of the signature oracle. Then it chooses $a \in [0, 2^{\kappa + \kappa_r} - 1]$ and computes $\mathbf{z} = \mathbf{h}^a$. It is shown in [54] that this allows the adversary to simulate the signature oracle perfectly. If the Type III forger outputs a valid signature $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$ such that for $e \neq e_i$ for all i it is concluded in [54] that

$$\boldsymbol{\sigma}^e = \mathbf{z}\mathbf{h}^{\mathsf{Eval}_2(i_2, \mathbf{z}')} \quad . \tag{3.1}$$

It is then shown that when $e < 2^{\kappa/2}$ this allows extraction of a non-trivial RSA-root of $\mathbf{g}$. The problem with our variant of the scheme is that we do not guarantee that $e < 2^{\kappa/2}$, and this turns out to be essential in our construction of the hierarchical group signature scheme in the third part of the thesis.

Fortunately, we can still extract a non-trivial root as follows. We modify the definition of the simulator such that it accepts $(\mathbf{N}, \mathbf{g}, \mathbf{z})$ as input, i.e., the simulator no longer generate $\mathbf{z}$. If the output of the forger is a valid signature $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$ such that $e \neq e_i$ for all i, then the simulator outputs $(\boldsymbol{\sigma}, e, 1, \mathsf{Eval}_2(i_2, \mathbf{z}'))$ that satisfies Equation (3.1). We conclude from Lemma 3.12 that there exists no Type III adversary. □

3.11 The Composite Residuosity Class Assumptions

Assumptions about the hardness of computing and deciding composite residuosity classes were first considered by Paillier [121] to prove the security of the Paillier cryptosystem introduced in the next section.

Let p and q be primes with the same number of bits and define $\mathsf{N} = \mathsf{pq}$ and $\mathsf{g} = \mathsf{N} + 1$. Define also $\mathcal{E}_\mathsf{g} : \mathbb{Z}_\mathsf{N} \times \mathbb{Z}_\mathsf{N}^* \to \mathbb{Z}_{\mathsf{N}^2}^*$, $\mathcal{E}_\mathsf{g} : (m, r) \mapsto \mathsf{g}^m r^\mathsf{N} \bmod \mathsf{N}^2$. Then Lemma 3 in [121] states that $\mathcal{E}_\mathsf{g}$ is a bijection.

Definition 3.16 (Residue Class). The N-th residue class $[u]_\mathsf{g}$ of u with respect to g is the unique $m \in \mathbb{Z}_\mathsf{N}$ such that there exists an $r \in \mathbb{Z}_\mathsf{N}^*$ such that $\mathcal{E}_\mathsf{g}(m, r) = u$.

Definition 3.17 (Composite Residuosity Assumption). Let p and q be randomly chosen $\kappa/2$-bit primes and define $\mathsf{N} = \mathsf{pq}$. Let $u \in \mathbb{Z}_{\mathsf{N}^2}^*$ be randomly chosen. The composite residuosity (CR) assumption states that for all adversaries A the probability $\Pr[A(\mathsf{N}, u) = [u]_\mathsf{g}]$ is negligible in κ.

Definition 3.18 (Decisional Composite Residuosity Assumption). Let p and q be randomly chosen $\kappa/2$-bit primes and define $\mathsf{N} = \mathsf{pq}$. Let $u \in \mathbb{Z}_{\mathsf{N}^2}^*$ be randomly chosen. The decisional composite residuosity (DCR) assumption states that for all adversaries $A \in \mathrm{PPT}^*$ the absolute value $|\Pr[A(\mathsf{N}, u) = 1] - \Pr[A(\mathsf{N}, u^\mathsf{N} \bmod \mathsf{N}^2) = 1]|$ is negligible in κ.

Under the DP-assumption we may assume that p and q are safe primes. This is the setting considered in this thesis.

3.12 The Paillier Cryptosystem

Paillier [121] introduces a cryptosystem based on the DCR-assumption that is additively homomorphic. The cryptosystem has some important technical properties that makes it suitable to construct protocols secure against an adaptive adversary.

The scheme is described as follows. On input 1^κ the key generator $\mathsf{CSKG}^\mathsf{pai}$ chooses two $\kappa/2$-bit safe primes p and q randomly and defines the public key $\mathsf{N} = \mathsf{pq}$. For notational convenience we define $\mathsf{g} = \mathsf{N} + 1$ and $\mathsf{f} = (\mathsf{p} - 1)(\mathsf{q} - 1)/4$. Then it chooses a private key d under the restriction $\mathsf{d} = 0 \bmod \mathsf{f}$ and $\mathsf{d} = 1 \bmod \mathsf{N}$ and outputs (N, d).

The cryptosystem is based on the algebraic properties of N and g. Recall that $\mathbb{Z}_{\mathsf{N}^2}^* = G_\mathsf{N} \times G_\mathsf{f} \times V$, where G_N and G_f are groups of order N and f, and V is the Klein 4-group. Furthermore, g is a generator of G_N and each element in $G_\mathsf{f} \times V$ can be written $r^\mathsf{N} \bmod \mathsf{N}^2$ with $r \in \mathbb{Z}_\mathsf{N}^*$. Note that $\mathsf{g}^m = 1 + m\mathsf{N} \bmod \mathsf{N}^2$. We define $L(u) = (u - 1)/\mathsf{N}$ and have $L(\mathsf{g}^m) = m$.

To encrypt a message $m \in \mathbb{Z}_\mathsf{N}$ a random $r \in \mathbb{Z}_\mathsf{N}^*$ is chosen and the cryptotext is defined by $u = E_\mathsf{N}(m, r) = \mathsf{g}^m r^{2\mathsf{N}} \bmod \mathsf{N}^2$. To decrypt a cryptotext u the decryptor computes $D_\mathsf{d}(u) = L(u^\mathsf{d} \bmod \mathsf{N}^2)$. To see why this works it suffices to note that every cryptotext $u \in G_\mathsf{N} \times G_\mathsf{f}$ and when $\mathsf{d} = 0 \bmod \mathsf{f}$ and $\mathsf{d} = 1 \bmod \mathsf{N}$ we have $u^\mathsf{d} \bmod \mathsf{N}^2 = \mathsf{g}^m$. In other words exponentiation by d collapses the group G_f, but keeps G_N fixed.

We remark that in the original version of the cryptosystem, encryption is defined as $\mathsf{g}^m r^\mathsf{N} \bmod \mathsf{N}^2$, i.e., we exponentiate with $2\mathsf{N}$ instead of N. It is easy to see that this does not decrease the security of the scheme. Thus, the following proposition follows from Paillier [121].

Proposition 3.19. *The Paillier cryptosystem is polynomially indistinguishable under the DCR-assumption.*

An interesting property of the scheme is that given a Paillier cryptotext $K_1 = E_{\mathsf{N}}(1, R_1) = \mathsf{g}R_1^{2\mathsf{N}} \bmod \mathsf{N}^2$ of 1 an alternative way to encrypt a message m is to compute $E_{K_1,\mathsf{N}}(m, r) = K_1^m r^{2\mathsf{N}} \bmod \mathsf{N}^2$. This property is essential in the construction of the adaptively secure mix-net in Chapter 11.

Let g_f be a generator of G_f. Then there exists an $2\mathsf{N}$th root r_f of g_f and an alternative way to encrypt a message is as $E_{\mathsf{g}_\mathsf{f},\mathsf{N}}(m, s) = \mathsf{g}^m \mathsf{g}_\mathsf{f}^s = \mathsf{g}^m (r_\mathsf{f}^s)^{2\mathsf{N}} \bmod \mathsf{N}^2$, where s is chosen randomly in $[0, 2^{\kappa+\kappa_r} - 1]$. Here κ_r is an additional security parameter that is large enough to make $2^{-\kappa_r}$ negligible. This implies that the distribution of the new type of cryptotext is statistically close to the distribution of the type we define above and κ_r decides the statistical distance.

The cryptosystem is homomorphic, i.e., $E_{\mathsf{N}}(m_1)E_{\mathsf{N}}(m_2) = E_{\mathsf{N}}(m_1 + m_2)$. As a consequence it is possible to re-encrypt a cryptotext u using randomness $s \in \mathbb{Z}_{\mathsf{N}}^*$ by computing $uE_{\mathsf{N}}(0, s) = E_{\mathsf{N}}(m, rs)$, or alternatively using randomness $s \in [0, 2^{\kappa+\kappa_r} - 1]$ as $uE_{\mathsf{g}_\mathsf{f},\mathsf{N}}(0, s) = E_{\mathsf{N}}(m, r\mathsf{g}_\mathsf{f}^{s/2\mathsf{N}})$. The distributions of the resulting cryptotexts are again statistically close. We use the latter type of re-encryption.

We remark that in Section 3.6, where we define the El Gamal cryptosystem, we use a very similar notation for encryption and decryption. This should not lead to any misunderstandings, since we never use both cryptosystems in the same context.

Chapter 4

The Universally Composable Security Framework

Canetti [41], and independently Pfitzmann and Waidner [127] propose security frameworks for reactive processes. We use the former universal composability (UC) framework. Both frameworks have composition theorems, and are based on older definitional work. The initial ideal-model based definitional approach for secure function evaluation is informally proposed by Goldreich, Micali, and Wigderson in [74]. The first formalizations appear in Goldwasser and Levin [75], Micali and Rogaway [106], and Beaver [13]. Canetti [40] presents the first definition of security that is preserved under composition. See [40, 41] for an excellent background on these definitions.

In this chapter we give a short review of the universally composable security framework. This framework is very general, quite complex, and hard to describe both accurately and concisely. We choose a simplified approach. For a general in depth discussion, intuition, and more details we refer the reader to Canetti [41].

Canetti assumes the existence of an "operating system" that takes care of the creation of subprotocols when needed. This is necessary to handle protocols with a large number of possible trees of calls to subprotocols, but for our purposes we may assume that all subprotocols are instantiated already at the start of the protocol.

Canetti models an asynchronous communication network, where the adversary has the power to delete, modify, and insert any messages of his choice. To do this he is forced to give details for exactly what the adversary is allowed to do. This becomes quite complex. We instead factor out all aspects of the communication network into a separate concrete "communication model"-machine. The real, ideal, and hybrid models are then defined solely by how certain machines are linked. The adversary is defined as any interactive polynomial time Turing machine, and how the adversary can interact with other machines follows implicitly from the definitions of the real and ideal communication models.

Since each protocol or subprotocol communicate through its own copy of the

"communication model", and all protocols are instantiated at the start of the protocol we need not bother with session ID:s. Such ID:s would clearly be needed if our protocols would be rewritten in the more general original security framework, but it is notationally convenient to avoid them.

4.1 Interactive Turing Machines

Following Goldreich [72] and Canetti [41] we define the parties to be probabilistic polynomial time interactive Turing machines with a history tape. More precisely, each machine has a read-only input tape, a write-once output tape, a read-only security parameter tape, a read-only random tape, and a history tape. It executes in polynomial time in the size of the input on the security parameter tape. At the start of the execution the security parameter tape contains the security parameter of the protocol, the random tape contains a sequence of randomly chosen bits, and the history tape is empty. Whenever a state transition takes place in a machine, the new state is recorded on the history tape. A protocol is said to be without erasures if no party ever deletes or overwrites any information written to the history tape. All our results are given in a model without erasures, but we discuss informally a protocol with erasures in Chapter 11. We denote the set of interactive Turing machines by ITM, and we denote the set of finite sequences of interactive Turing machines by ITM*.

We also assume that we may connect any pair of machines by a "link" to allow them to communicate. Such a link is more or less equivalent to the notion of a link introduced by Goldreich [72]. But the original notion of a link has the problem that it requires a machine to have a pair of communication tapes for each link, which is problematic when the number of potential links is unbounded. This is a purely definitional problem of no importance and we trust the reader to fill in the details of this in any way he or she chooses. Thus, the following is meaningful.

Definition 4.1 (ITM-Graph). An ITM-graph is a set $V = \{P_1, \ldots, P_t\} \subset \text{ITM}$ with a set of links E such that (V, E) is a connected graph, and no P_i is linked to any machine outside V. Let ITMG be the set of ITM-graphs.

During the execution of an ITM-graph, at most one party, called the active party, change its state in each step. A party can write to exactly one link l in each activation and then activates the party to which it is linked with l.

4.2 The Real Model

The real communication model models an asynchronous communication network, in which the adversary can read, delete, modify, and insert any message of its choice.

Definition 4.2 (Real Communication Model). A real communication model $\mathcal{C}$ is a machine with a link l_{P_i}, to P_i for $i = 1, \ldots, k$, and a link $l_{\mathcal{A}}$ to a real adversary $\mathcal{A}$. Its program is defined as follows.

- If m is read on l_s, where $s \in \{P_1, \ldots, P_k\}$, then (s, m) is written on l_A and $\mathcal{A}$ is activated.
- If (r, m) is read on $l_{\mathcal{A}}$, where $r \in \{P_1, \ldots, P_k\}$, then m is written on l_r, and r is activated.

The real model is supposed to capture the properties of the real world. The parties may interact over the real communication model.

Definition 4.3 (Real Model). The real model is defined to be a map $\mathcal{R} : \text{ITM}^* \to \text{ITMG}$, where $\mathcal{R} : (\mathcal{A}, P_1, \ldots, P_k) \mapsto (V, E)$ is given by:

$$V = \{\mathcal{C}, \mathcal{A}, P_1, \ldots, P_k\} \ , \quad \text{and} \quad E = \{(\mathcal{A}, \mathcal{C})\} \cup \bigcup_{i=1}^{k} \{(P_i, \mathcal{C})\} \ .$$

We formalize the environment in which a protocol is executed as a machine $\mathcal{Z}$. The environment provide the actual data used by the parties in the protocol. Let $(V, E) = \mathcal{R}(\mathcal{A}, P_1, \ldots, P_k)$. Then we write $\mathcal{Z}(\mathcal{R}(\mathcal{A}, P_1, \ldots, P_k))$ for the ITM-graph (V', E') defined by $V' = V \cup \{\mathcal{Z}\}$, and $E' = E \cup \{(\mathcal{Z}, \mathcal{A})\} \cup \bigcup_{i=1}^{k} \{(\mathcal{Z}, P_i)\}$. The setup is illustrated in Figure 4.1.

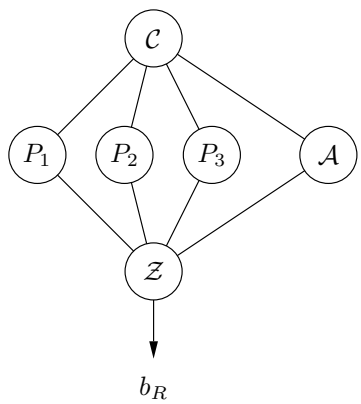

Figure 4.1: The real model with a protocol (P_1, P_2, P_3), communication model $\mathcal{C}$, real adversary $\mathcal{A}$, and environment $\mathcal{Z}$. The environment outputs a single bit b_R.

4.3 The Ideal Model

The ideal model formalizes protocol execution in an ideal world where there is an ideal functionality, i.e., a trusted party that performs some service to the parties. The trusted party is simply an interactive Turing machine executing a program, and it communicates with the parties through the ideal communication model.

The ideal communication model below captures the fact that the adversary may decide if and when it would like to deliver a message from the ideal functionality to a party, but it cannot read the contents of the communication between parties and the ideal functionality.

Definition 4.4 (Ideal Communication Model). An ideal communication model $\mathcal{C}_\mathcal{I}$ is a machine with a link l_{P_i}, to P_i for $i = 1, \ldots, k$, and links $l_\mathcal{F}$, and $l_\mathcal{S}$ to an ideal functionality $\mathcal{F}$ and an ideal adversary $\mathcal{S}$ respectively. Its program is defined as follows.

- If a message m is read on l_s, where $s \in \{P_1, \ldots, P_k\}$, then (s, m) is written on $l_\mathcal{F}$ and $\mathcal{F}$ is activated.
- If a message (s, m) written on $l_\mathcal{F}$ is returned unaltered, m is written on l_s.

  If not, any string read from $l_\mathcal{F}$ is interpreted as a list $((r_1, m_1), \ldots, (r_t, m_t))$, where $r_i \in \{\mathcal{S}, P_1, \ldots, P_k\}$. For each m_i a random string $\tau_i \in \{0,1\}^\kappa$ distinct from $1, \ldots, k$, and $\mathcal{F}$ is chosen, and (r_i, m_i) is stored under τ_i. Then $((r_1, |m_1|, \tau_1), \ldots, (r_t, |m_t|, \tau_t))$, where $|m_i|$ is the bit-length of m_i, is written to $l_\mathcal{S}$ and $\mathcal{S}$ is activated.
- Any string read from $l_\mathcal{S}$ is interpreted as a pair (b, τ), where $b \in \{0,1\}$ and τ is an arbitrary string. If $b = 1$ and (r_i, m_i) is stored in the database under the index τ, m_i is written on l_{r_i} and r_i is activated. Otherwise $(\mathcal{S}, \tau)$ is written to $l_\mathcal{F}$ and $\mathcal{F}$ is activated.

Without loss we assume that the ideal adversary $\mathcal{S}$ always requests the messages m_i for which it is given $(\mathcal{S}, |m_i|, \tau_i)$. Thus, we sometimes say that $\mathcal{S}$ receives a list $((\mathcal{S}, m), \{(r_i, |m_i|, \tau_i)\})$ when in reality $(\mathcal{S}, m)$ is $(\mathcal{S}, |m|, \tau)$ for some τ. We also simply say that $\mathcal{S}$ hands $(\mathcal{F}, m)$ to $\mathcal{C}_\mathcal{I}$, when we should say that it hands $(0, \mathcal{F}, m)$.

A *dummy party* is a machine that given two links writes any message from one of the links on the other. There may be many copies of the dummy party. Following Canetti we write $\tilde{P}$, for dummy parties. We write $\tilde{P}^*$ for the set of finite sequences of dummy machines.

The ideal model below captures the setup one wishes to realize, i.e., the environment may interact with the ideal functionality $\mathcal{F}$, except that the adversary $\mathcal{S}$ has some control over how the communication model behaves.

Definition 4.5 (Ideal Model). The ideal model is defined to be a map $\mathcal{I} : \mathrm{ITM}^2 \times \tilde{P}^* \to \mathrm{ITMG}$, where $\mathcal{I} : (\mathcal{F}, \mathcal{S}, \tilde{P}_1, \ldots, \tilde{P}_k) \mapsto (V, E)$ is given by:

$$V = \{\mathcal{C}_\mathcal{I}, \mathcal{F}, \mathcal{S}, \tilde{P}_1, \ldots, \tilde{P}_k\} \ , \quad \text{and} \quad E = \{(\mathcal{S}, \mathcal{C}_\mathcal{I}), (\mathcal{C}_\mathcal{I}, \mathcal{F})\} \cup \bigcup_{i=1}^{k} \{(\tilde{P}_i, \mathcal{C}_\mathcal{I})\} \ .$$

If $\tilde{\pi} = (\tilde{P}_1, \ldots, \tilde{P}_k)$, we write $\mathcal{I}(\mathcal{S}, \tilde{\pi}^\mathcal{F})$ instead of $\mathcal{I}(\mathcal{F}, \mathcal{S}, \tilde{P}_1, \ldots, \tilde{P}_k)$ to simplify notation.

Let $(V, E) = \mathcal{I}(\mathcal{F}, \mathcal{S}, \tilde{P}_1, \ldots, \tilde{P}_k)$. Similarly, to the real model we consider an environment $\mathcal{Z}$ and write $\mathcal{Z}(\mathcal{I}(\mathcal{F}, \mathcal{S}, \tilde{P}_1, \ldots, \tilde{P}_k))$ for the ITM-graph (V', E') defined by $V' = V \cup \{\mathcal{Z}\}$, and $E' = E \cup \{(\mathcal{Z}, \mathcal{S})\} \cup \bigcup_{i=1}^{k} \{(\mathcal{Z}, \tilde{P}_i)\}$. The setup is illustrated in Figure 4.2. For simplicity we say that an ideal functionality "ignores an input" when it returns the input to the sender immediately. We also assume that inputs on forms that are not explicitly considered as inputs are ignored. The same convention is used for protocols. This simplifies the description of both ideal functionalities and protocols.

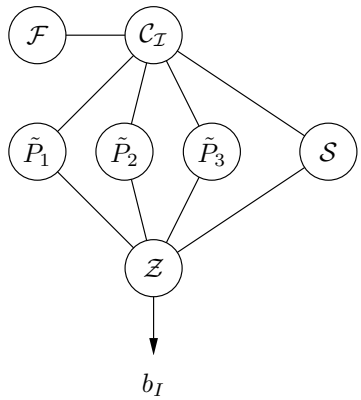

Figure 4.2: The ideal model with dummy parties $(\tilde{P}_1, \tilde{P}_2, \tilde{P}_3)$, ideal communication model $\mathcal{C}_{\mathcal{I}}$, ideal functionality $\mathcal{F}$, ideal adversary $\mathcal{S}$, and environment $\mathcal{Z}$. The environment outputs a single bit b_I.

4.4 The Hybrid Model

A hybrid model is a mix between a number of ideal and real models, and captures the execution of a real world protocol with access to some ideal functionalities. It is also a tool to modularize security proofs. It may be viewed as if we "glue" a number of ideal and real models onto an original real model.

Definition 4.6 (Hybrid Model). Suppose that we are given $(V, E) = \mathcal{R}(\mathcal{A}, \pi)$, $\pi = (P_1, \ldots, P_k)$. Let $(V_j, E_j) = \mathcal{I}(\mathcal{S}_j, \tilde{\pi}_j^{\mathcal{F}_j})$, $\tilde{\pi}_j = (\tilde{P}_{j,1}, \ldots, \tilde{P}_{j,k})$ for $j = 1, \ldots, t$, and $(V_j, E_j) = \mathcal{R}(\mathcal{A}_j, \pi_j)$, $\pi_j = (P_{j,1}, \ldots, P_{j,k})$ for $j = t+1, \ldots, s$.

We denote by $\mathcal{H}(\mathcal{A}^{(\mathcal{S}_1, \ldots, \mathcal{S}_t, \mathcal{A}_{t+1}, \ldots, A_s)}, \pi^{(\tilde{\pi}_1^{\mathcal{F}_1}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t}, \pi_{t+1}, \ldots, \pi_s)})$ the hybrid model

defined as the ITM-graph (V', E'), where

$$\begin{aligned} V' &= V \cup \bigcup_{j=1}^{s} V_j \ , \quad \text{and} \\ E' &= E \cup \bigcup_{j=1}^{s} E_j \cup \bigcup_{j=1}^{t} \left(\{(\mathcal{S}_j, \mathcal{A})\} \cup \bigcup_{i=1}^{k} \{(P_i, \tilde{P}_{j,i})\} \right) \\ &\quad \cup \bigcup_{j=t+1}^{s} \left(\{(\mathcal{A}_j, \mathcal{A})\} \cup \bigcup_{i=1}^{k} \{(P_i, P_{j,i})\} \right) \ . \end{aligned}$$

Similarly as above we consider an environment $\mathcal{Z}$ and write

$$\mathcal{Z}(\mathcal{H}(\mathcal{A}^{(\mathcal{S}_1,\ldots,\mathcal{S}_t,\mathcal{A}_{t+1},\ldots,A_s)}, \pi^{(\tilde{\pi}_1^{\mathcal{F}_1},\ldots,\tilde{\pi}_t^{\mathcal{F}_t},\pi_{t+1},\ldots,\pi_s)}))$$

to denote the ITM-graph (V'', E'') defined by $V'' = V' \cup \{\mathcal{Z}\}$, and $E'' = E' \cup \{(\mathcal{Z}, \mathcal{A})\} \cup \bigcup_{i=1}^{k} \{(\mathcal{Z}, P_i)\}$. Note that all real subprotocols π_j, for $j = t+1, \ldots, s$, above may be integrated into the original real protocol π. Thus, a hybrid model with no ideal functionalities involved is equivalent to a real model, except that it may use several communication models. One may either augment the definition of a real model to allow this, or only use communication models with the property that two communication models can be simulated using a single communication model. The real communication model above, Definition 4.2, has this property. A hybrid model is illustrated in Figure 4.1.

The concept of hybrid models is generalized in the natural way, e.g. we write $\mathcal{H}(\mathcal{A}^{(A_1^{\mathcal{S}_{11}}, A_2^{\mathcal{S}_{21}})}, \pi^{(\pi_1^{\tilde{\pi}_{11}^{\mathcal{F}}}, \pi_2^{\tilde{\pi}_{21}^{\mathcal{F}}})})$ for a hybrid model that executes two subprotocols, where each subprotocol has access to a separate copy of the ideal functionality $\mathcal{F}$. Some care needs to be taken when defining the adversary for such models. If an adversary corrupts a party, it is reasonable in our applications that it automatically corrupts all its sub-parties that are involved in subprotocols. We remark that this rule may be relaxed in other settings, e.g. it may be assumed that some subprotocol is executed on a smartcard that can not be corrupted.

Remark 4.7. In protocols in the real world it is often the case that one party waits for some input from another party. When we model such a party or such an ideal functionality we sometimes say that it "waits" until some event occurs or for some input. Since at most one party is active, this can not be interpreted literally. Instead, we assume that the party or ideal functionality checks if the event occurred and if not activates the adversary, e.g. by sending the empty message to itself. When the party is activated again execution starts at the point it was left off.

4.5 Corruption of Parties

Intuitively, a party is corrupted if it is under the control of the adversary, and adversary should be able to corrupt parties in each of the three models described

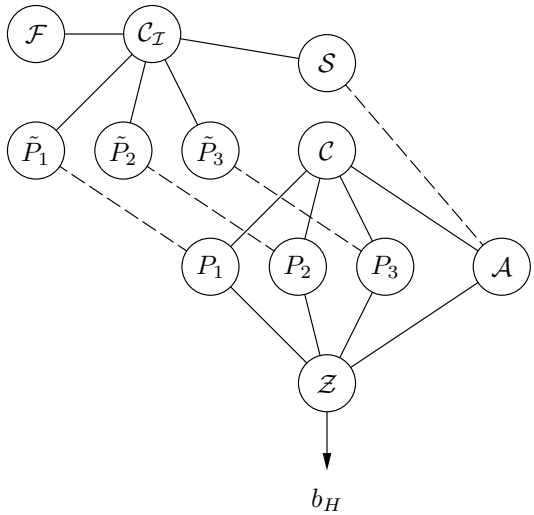

Figure 4.3: The hybrid model with a protocol (P_1, P_2, P_3), communication model $\mathcal{C}$, real adversary $\mathcal{A}$, dummy parties $(\tilde{P}_1, \tilde{P}_2, \tilde{P}_3)$, ideal functionality $\mathcal{F}$, ideal subadversary $\mathcal{S}$, and environment $\mathcal{Z}$. The environment outputs a single bit b_H.

above. We describe one possible formalization of this.

We assume that the execution of any model starts with the unique adversary linked to the environment activated, and formalize corruptions by requiring a certain behavior of communication models, real parties, and dummy parties.

If in the hybrid model a party is corrupted, then so should all its subparties and super parties as well. Furthermore, their history tapes and the control over their future execution should be given to the adversary.

Denote by $\mathit{links}(P_i)$ the set of links of a party P_i. We consider first the problem of collecting the history tapes of the corrupted party and its superparties and subparties.

When a party P_i receives a message (`corrupt`) on a link $l_s \neq l_{\mathcal{Z}}$ it does the following. It initializes a string h_{history} with the contents of its history tape. On each link $l_t \in \mathit{links}(P_i) \setminus \{l_s, l_{\mathcal{C}}, l_{\mathcal{Z}}\}$, in some canonical order, it writes (`corrupt`), waits until a string $(\texttt{history}, h)$ is returned on l_t, and appends h to h_{history}. Then it writes $(\texttt{history}, h_{\text{history}})$ on the link l_s. We also assume that a party P_i never writes (`corrupt`) or $(\texttt{history}, h)$ for any h on any link if it has not received (`corrupt`) on a link $l_s \neq l_{\mathcal{Z}}$. This prohibits the environment or a party to corrupt other parties or itself.

Finally, from this point on it behaves as follows. If P_i receives a message on the form $(\texttt{RemoteSend}, s, m)$ on the link $l_s \neq l_{\mathcal{Z}}$ it writes m on the link l_s. If a message m on any other form is received on a link l_s it writes $(\mathcal{A}, s, m)$ on the link to $\mathcal{C}$.

This gives the adversary the control over the corrupted party's links. To avoid that $\mathcal{Z}$ or some other party controls the links of another party, we also assume that no party ever writes $(\texttt{RemoteSend}, s, m)$ with any s and m on any link unless it has already received $(\texttt{Corrupt})$ on a link $l_s \neq l_{\mathcal{Z}}$.

The requirement on the ideal model follows by replacing P_i with $\tilde{P}_i$, $\mathcal{A}$ with $\mathcal{S}$, and $\mathcal{C}$ with $\mathcal{C}_{\mathcal{I}}$ above, except that $(\texttt{corrupt})$ is written also on $l_{\mathcal{C}_{\mathcal{I}}}$. The reason we require interaction with the ideal communication model is that each ideal functionality should be informed each time one of its dummy parties is corrupted. We ensure this as follows. We require that each copy $\mathcal{C}_{\mathcal{I}}$ of the ideal communication model on input $(\texttt{corrupt})$ on a link $l_s \neq l_{\mathcal{S}}$ writes $(\texttt{corrupt}, s)$ on the link $l_{\mathcal{F}}$ to its ideal functionality. Then it waits until $(\texttt{corrupt}, s)$ is returned from $\mathcal{F}$ and then writes $(\texttt{history}, \emptyset)$ on l_s. If $\mathcal{F}$ writes any other message on $l_{\mathcal{F}}$ the message is simply returned to $\mathcal{F}$ on $l_{\mathcal{F}}$. Note that the ideal functionality is informed, but it is not allowed to communicate with any other party during the activation.

If all parties in a protocol are corrupted no protocol is secure in any interesting way. Thus, the power of the adversary is limited in terms of how many parties it may corrupt and when it they are corrupted.

We say that an adversary is static if it never receives a message on the form $(P_i, \texttt{history}, h)$ with $h \neq \emptyset$ from $\mathcal{C}$ or correspondingly for a dummy party $\tilde{P}_i$ and $\mathcal{C}_{\mathcal{I}}$. This means that every corrupted party is corrupted before it is activated. When a party is not static it is called adaptive, since it may choose which parties to corrupt during the execution of the protocol.

Throughout the thesis we abuse notation and simply say that a party is corrupted, instead of saying that the adversary hands a corrupt message to the communication model.

4.6 The Definition of Security

We write $\mathcal{Z}_z$ to denote that the environment $\mathcal{Z}$ takes auxiliary input z. The following definition is somewhat sloppy in that we have not defined the notion of $\mathcal{M}$-adversaries rigorously. We trust the reader to resolve this, and assume that $\mathcal{M}$ is some class of adversaries.

Definition 4.8 (Secure Realization). Let $\mathcal{F}$ be an ideal functionality. Let $\pi = (P_1, \ldots, P_k)$, and let $\tilde{\pi}_j = (\tilde{P}_{j,i}, \ldots, \tilde{P}_{j,i})$ be the corresponding dummy parties for $\mathcal{F}_j$, for $j = 1, \ldots, t$.

Then $\pi^{(\tilde{\pi}_1^{\mathcal{F}_1}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})}$ securely realizes $\tilde{\pi}^{\mathcal{F}}$ with regards to $\mathcal{M}$-adversaries if for all $\mathcal{M}$-adversaries $\mathcal{A}^{(\mathcal{S}_1, \ldots, \mathcal{S}_t)}$ there exists a $\mathcal{M}$-adversary $\mathcal{S}$ such that for all environments $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$

$$|\Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}})) = 1] - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}^{(\mathcal{S}_1, \ldots, \mathcal{S}_t)}, \pi^{(\tilde{\pi}_1^{\mathcal{F}_1}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})})) = 1]|$$

is negligible in the security parameter κ. Since the dummy parties merely play the role of handles for the links and keepers of the history of the messages that were

transmitted on the link we also say that π realizes $\mathcal{F}$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_t)$-hybrid model.

4.7 The Composition Theorem

Canetti [41] proves a powerful composition theorem. Loosely speaking it says that if a protocol π securely realizes some functionality $\mathcal{F}$, then the protocol can be used instead of the ideal functionality regardless of how the functionality $\mathcal{F}$ is employed.

The general composition theorem can handle polynomially many instances of a constant number of ideal functionalities, but we only need the following weaker special case.

Theorem 4.9 (Composition Theorem). *Suppose that $\pi^{(\tilde{\pi}_1^{\mathcal{F}_1}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})}$ securely realizes $\tilde{\pi}^{\mathcal{F}}$, and that $\pi_i^{(\tilde{\pi}_{i1}^{\mathcal{F}_{i1}}, \ldots, \tilde{\pi}_{it_i}^{\mathcal{F}_{it_i}})}$ securely realizes $\tilde{\pi}_i^{\mathcal{F}_i}$, for $i = 1, \ldots, l$, with regards to $\mathcal{M}$-adversaries, and assume that l is a constant.*

Then $\pi^{(\pi_1^{(\tilde{\pi}_{11}^{\mathcal{F}_{11}}, \ldots, \tilde{\pi}_{1t_1}^{\mathcal{F}_{1t_1}})}, \ldots, \pi_l^{(\tilde{\pi}_{l1}^{\mathcal{F}_{l1}}, \ldots, \tilde{\pi}_{lt_l}^{\mathcal{F}_{lt_l}})}, \tilde{\pi}_{l+1}^{\mathcal{F}_{l+1}}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})}$ securely realizes $\tilde{\pi}^{\mathcal{F}}$ with regards to $\mathcal{M}$-adversaries.

Note that the hybrid protocol can be transformed into a hybrid protocol in the $(\mathcal{F}_{11}, \ldots, \mathcal{F}_{1t_1}, \ldots, \mathcal{F}_{l1}, \ldots, \mathcal{F}_{lt_l}, \mathcal{F}_{l+1}, \ldots, \mathcal{F}_t)$-hybrid model by contracting all the real subparties linked to each party linked to the environment. Thus, the theorem is in fact very general.

Proof of Theorem 4.9. For completeness we give a simple proof of the theorem. A proof of the more general statement can be found in Canetti [41].

If the theorem is false there exists an $\mathcal{M}$-adversary

$$\mathcal{A}_{\mathrm{H}} = \mathcal{A}^{(\mathcal{A}_1^{(\mathcal{S}_{11}, \ldots, \mathcal{S}_{1t_1})}, \ldots, \mathcal{A}_l^{(\mathcal{S}_{l1}, \ldots, \mathcal{S}_{lt_l})}, \mathcal{S}_{l+1}, \ldots, \mathcal{S}_t)}$$

such that for each ideal adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$ such that

$$\begin{aligned} |\Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}_{\mathrm{H}}, \pi^{(\pi_1^{(\tilde{\pi}_{11}^{\mathcal{F}_{11}}, \ldots, \tilde{\pi}_{1t_1}^{\mathcal{F}_{1t_1}})}, \ldots, \pi_l^{(\tilde{\pi}_{l1}^{\mathcal{F}_{l1}}, \ldots, \tilde{\pi}_{lt_l}^{\mathcal{F}_{lt_l}})}, \tilde{\pi}_{l+1}^{\mathcal{F}_{l+1}}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})})) = 1] \\ - \Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}})) = 1]| \end{aligned}$$

is non-negligible, which implies that

$$\begin{aligned} |\Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}_{\mathrm{H}}, \pi^{(\pi_1^{(\tilde{\pi}_{11}^{\mathcal{F}_{11}}, \ldots, \tilde{\pi}_{1t_1}^{\mathcal{F}_{1t_1}})}, \ldots, \pi_l^{(\tilde{\pi}_{l1}^{\mathcal{F}_{l1}}, \ldots, \tilde{\pi}_{lt_l}^{\mathcal{F}_{lt_l}})}, \tilde{\pi}_{l+1}^{\mathcal{F}_{l+1}}, \ldots, \tilde{\pi}_t^{\mathcal{F}_t})})) = 1] \\ - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}_{\mathrm{H}}, \pi^{\tilde{\pi}_1^{\mathcal{F}_1} \ldots, \tilde{\pi}_t^{\mathcal{F}_t})})) = 1]| \end{aligned}$$

is non-negligible, which by a standard hybrid argument implies that there exists a fixed $0 < i \leq l$ such that

$$|\Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}_{\mathrm{H}}, \pi^{(\pi_1^{(\tilde{\pi}_{11}^{\mathcal{F}_{11}},\ldots,\tilde{\pi}_{1t_1}^{\mathcal{F}_{1t_1}})},\ldots,\pi_i^{(\tilde{\pi}_{i1}^{\mathcal{F}_{i1}},\ldots,\tilde{\pi}_{it_i}^{\mathcal{F}_{it_i}})},\tilde{\pi}_{i+1}^{\mathcal{F}_{i+1}},\ldots,\tilde{\pi}_t^{\mathcal{F}_t})})) = 1]$$
$$- \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}_{\mathrm{H}}, \pi^{(\pi_1^{(\tilde{\pi}_{11}^{\mathcal{F}_{11}},\ldots,\tilde{\pi}_{1t_1}^{\mathcal{F}_{1t_1}})},\ldots,\pi_{i-1}^{(\tilde{\pi}_{i-1,1}^{\mathcal{F}_{i-1,1}},\ldots,\tilde{\pi}_{i-1,t_{i-1}}^{\mathcal{F}_{i-1t_{i-1}}})},\tilde{\pi}_i^{\mathcal{F}_i},\ldots,\tilde{\pi}_t^{\mathcal{F}_t})})) = 1]|$$

is non-negligible.

If we denote by $\mathcal{Z}'$ the environment that simulates all machines in the latter model except $\mathcal{F}_i$, $\mathcal{S}_i$, $\tilde{\pi}_i$, and their corresponding ideal communication model, then we conclude that there exists a $\mathcal{M}$-adversary $\mathcal{A}_i^{(\mathcal{S}_{i1},\ldots,\mathcal{S}_{it_i})}$ such that for each $\mathcal{M}$-adversary $\mathcal{S}_i$ there exists an environment $\mathcal{Z}'$ with auxiliary input $z = \{z_\kappa\}_{\kappa\in\mathbb{N}}$ such that

$$|\Pr[\mathcal{Z}'_z(\mathcal{H}(\mathcal{A}_i^{(\mathcal{S}_{i1},\ldots,\mathcal{S}_{it_i})}, \pi_i^{(\tilde{\pi}_{i1}^{\mathcal{F}_{i1}},\ldots,\tilde{\pi}_{it_i}^{\mathcal{F}_{it_i}})})) = 1] - \Pr[\mathcal{Z}'_z(\mathcal{I}(\mathcal{S}_i, \tilde{\pi}_i^{\mathcal{F}_i})) = 1]|$$

is non-negligible. This contradicts the security of the hybrid protocol $\pi_i^{(\tilde{\pi}_{i1}^{\mathcal{F}_{i1}},\ldots,\tilde{\pi}_{it_i}^{\mathcal{F}_{it_i}})}$ and the theorem follows. □

Remark 4.10. It is interesting to note that the key idea of the composition theorem can be presented in such a simple way compared to the complex proof of the more general statement.

Part II

Mix-Nets

Chapter 5

Preliminaries

In this chapter we first give a more detailed account of previous and related work on mix-nets. Then we introduce additional notation and introduce some ideal functionalities used only in the second part of the thesis.

5.1 Previous and Related Work On Mix-Nets

Although electronic elections perhaps is the most spectacular application of mix-nets, Chaum proposes the notion as a general means for a group of senders to send messages without revealing their identity, and then presents electronic elections as an interesting application.

Chaum's original "anonymous channel" [45] enables a sender to securely send mail to a receiver anonymously, and also to securely receive mail from this recipient without revealing the sender's identity. This type of application is also considered by Ogata, Kurosawa, Sako, and Takatani [120]. Several authors have refined Chaum's idea and used it to construct electronic election schemes, e.g. Fujioka, Okamoto, and Ohta [66], Park, Itoh, and Kurosawa [122], Sako and Killian [134], and Niemi and Renvall [117]. In all such constructions the mix-net is used to ensure the anonymity of the voters. Less common applications of mix-nets are to ensure privacy in the construction of electronic cash systems as proposed by Jakobsson and M'Raihi [92], or for general function evaluation as proposed by Jakobsson and Juels [89]. Thus, a mix-net is a useful primitive in constructing cryptographic protocols, and of particular importance in the construction of electronic election systems.

Abe gives an efficient construction of a general mix-net [2], and argues about its properties. Jakobsson has written (partly with Juels) a number of more general papers on the topic of mixing [87, 86, 88] also focusing on efficiency, of which the first appeared at the same time as Abe's construction.

An alternative approach to achieve robustness was proposed by Jakobsson and Juels [90] and further refined by Golle et al. [79]. They give an optimistic protocol that is more efficient than protocols in which each mix-server proves in zero-

knowledge that it performs its computations correctly. Another approach is to relax the requirement on correctness such that a mix-server could conceivably corrupt the output on a small scale, but not on a larger scale. This idea was proposed by Jakobsson, Juels and Rivest [91].

There are two known efficient approaches to proving knowledge of a witness of the correctness of the transformation computed by a mix-server. These are introduced by Neff [112, 113] and Furukawa et al. [70, 69, 68] respectively. Groth [80] generalizes Neff's protocol to form an abstract protocol for any homomorphic cryptosystem. In Chapter 9 we give an alternative approach originally presented in [149, 148]. In independent work, Peng, Boyd, and Dawson [124] gave yet another approach, but they do not prove that their construction is a proof of knowledge, which seems to be an essential property.

There has also been a surprisingly large number of successful attacks on mix-nets. Desmedt and Kurosawa [59] describe an attack on a protocol by Jakobsson [87] based on the fact that mix-servers are not forced to use the output of the previous mix-server in the mixing-chain as input. Mitomo and Kurosawa [108] exhibit a weakness in another protocol by Jakobsson [86]. Pfitzmann has given some general attacks on mix-nets [126, 125], based on the malleability of cryptosystems. Michels and Horster give additional attacks in [107].

In Chapter 6 we describe some attacks, taken from Wikström [146], against a protocol by Golle et al. [79]. We also give attacks against the protocols by Jakobsson [86] and Jakobsson and Juels [90]. Abe and Imai [5] have independently found related attacks.

Several informal and ad hoc definitions of security are proposed in the literature. The first formal definition is given by Abe and Imai [5], but as far as we know there exists no protocol that is shown to satisfy this definition. Nguyen, Safavi-Naini and Kurosawa [114, 115] translate [70] to the Paillier cryptosystem and provide non-standard definitions of security of a proof of a shuffle. They do not provide a definition of security of a mix-net.

The first mix-net with a complete security proof with regards to a rigorous definition is given in Wikström [147]. The definitions in Chapter 7 and the zero-knowledge proof of knowledge of the cleartext of an El Gamal cryptotext in Chapter 10 are taken from this paper. The construction in [147] is only efficient as long as the number of mix-servers is small. This deficiency is eliminated in different ways in our constructions in Chapter 8 and Chapter 11. These constructions are taken from Wikström [148] and Wikström and Groth [150] respectively. We remark that independently of us, Camenisch and Mityagin [37] claim to reach similar results as those in [148].

5.2 Additional Notation

Throughout the second part of the thesis $S_1, \ldots, S_N$ denote senders and $M_1, \ldots, M_k$ mix-servers. All parties are modeled as interactive Turing machines with a history

tape. This means that each machine records its new state on a special history tape each time a state transition takes place. Thus, when a machine is corrupted its entire execution history is available to the adversary unless the machine erases the history tape. We abuse notation and use S_i and M_j to denote both the machines themselves and their identity. We denote by Sort the algorithm that given a list of strings as input outputs the same set of strings in lexicographical order. We denote by $k' = \lceil (k+1)/2 \rceil$ the number of mix-servers needed for majority.

We assume that all protocols always verify that their input is on the required form. This is an important part of any cryptographic protocol, since the security claims only holds under this assumption. For an explicit example of what can go wrong if this rule is not taken seriously we refer the reader to the third attack in Chapter 6. There is a simple trick to avoid verifications during mixing if the mix-net is employed in some particular groups. This trick is described in detail in Wikström [145]. We use this trick in Chapter 11, but we do not discuss it explicitly.

5.3 Some Ideal Functionalities

In this section we introduce some ideal functionalities that we assume to exist throughout the second part of the thesis. In this thesis we do not study how to securely realize these functionalities in the UC-framework. This means that presently one must invoke general methods to securely realize these functionalities. Canetti, Lindell, Ostrovsky, and Sahai [44] prove that essentially any functionality can be securely realized in a hybrid model with an ideal authenticated broadcast channel in the common random string model. Their constructions are based on the work Goldreich, Micali, and Wigderson [74]. The common random string model assumes that all parties have access to a randomly generated string. However, there are efficient secure realizations in other security models than the UC-framework that are a natural starting point to securely realize the functionalities below. We point to some interesting work below each ideal functionality.

All mix-net constructions given in the literature assume the existence of an authenticated bulletin board, but this assumption is rarely formalized. Below we define this notion as an ideal functionality. Recall from the previous chapter that $\mathcal{C}_{\mathcal{I}}$ denotes the communication model.

Functionality 5.1 (Bulletin Board). The ideal bulletin board functionality, $\mathcal{F}_{\mathrm{BB}}$, running with parties $P_1, \ldots, P_n$ and ideal adversary $\mathcal{S}$ proceeds as follows. $\mathcal{F}_{\mathrm{BB}}$ holds two databases D_1 and D_2 indexed on integers. Initialize two counters $c_1 = 0$ and $c_2 = 0$.

- Upon receiving $(P_i, \texttt{Write}, m_i)$, $m_i \in \{0,1\}^*$, from $\mathcal{C}_{\mathcal{I}}$, store (P_i, m_i) in D_2 by the index c_2 in the database, set $c_2 \leftarrow c_2 + 1$, and hand $(\mathcal{S}, \texttt{Input}, c_2, P_i, m_i)$ to $\mathcal{C}_{\mathcal{I}}$.

- Upon receiving $(\mathcal{S}, \texttt{AcceptInput}, c)$ from $\mathcal{C}_\mathcal{I}$ check if a tuple (P_i, m_i) is stored in the database D_2 under c. If so, then store (P_i, m_i) in D_1 under the index c_1, set $c_1 \leftarrow c_1 + 1$, and hand $(\mathcal{S}, \texttt{AcceptInput}, c)$ to $\mathcal{C}_\mathcal{I}$.
- Upon receiving $(P_j, \texttt{Read}, c)$ from $\mathcal{C}_\mathcal{I}$ check if a tuple (P_i, m_i) is stored in the database D_1 under c. If so hand $((\mathcal{S}, P_j, \texttt{Read}, c, P_i, m), (P_j, \texttt{Read}, c, P_i, m_i))$ to $\mathcal{C}_\mathcal{I}$. If not, hand $((\mathcal{S}, P_j, \texttt{NoRead}, c), (P_j, \texttt{NoRead}, c))$ to $\mathcal{C}_\mathcal{I}$.

This definition differs from that in [147] in that the adversary is given the power to decide if and when a message submitted for publication on the bulletin board actually appears there.

We sometimes ignore the index c and simply say that a machine waits until a message on a certain form appears on the bulletin board. Whenever we do so it is assumed that the waiting party polls the bulletin board, so a message "appears" when it is successfully received from the bulletin board (recall that the adversary may block the delivery of a message that is contained in the database D_1).

When protocols are executed with multiple verifiers it is useful to have an ideal coin-flipping functionality.

Functionality 5.2 (Coin-Flipping). The ideal Coin-Flipping functionality, $\mathcal{F}_{\text{CF}}$, with mix-servers $M_1, \ldots, M_k$, and adversary $\mathcal{S}$ proceeds as follows. Set $J_{\kappa,} = \emptyset$ for all κ.

- On receipt of $(M_j, \texttt{GenerateCoins}, \kappa)$ from $\mathcal{C}_\mathcal{I}$, set $J_\kappa \leftarrow J_\kappa \cup \{j\}$. If $|J_\kappa| = k$, then set $J_\kappa \leftarrow \emptyset$ choose $c \in \{0,1\}^\kappa$ randomly and hand $((\mathcal{S}, \texttt{Coins}, c), \{(M_j, \texttt{Coins}, c)\}_{j=1}^k)$ to $\mathcal{C}_\mathcal{I}$.

We sometimes use the coin-flipping functionality as if it outputs elements in specific groups. Whenever we do this there is a standard way to transform a random bit-string of suitable length into an element in the group. For example, if we compute in the subgroup G_q of squares modulo a prime $2q + 1$, a random element in $\mathbb{Z}_q$ can be transformed into a random element in G_q by simply squaring s modulo $2q + 1$.

We also need an ideal functionality that generates an RSA-modulus $\mathbf{N}$ and random elements $\mathbf{g}, \mathbf{h} \in \text{QR}_\mathbf{N}$.

Functionality 5.3 (RSA Common Reference String). The ideal RSA Common Reference String, $\mathcal{F}_{\text{RSA}}$, with mix-servers $M_1, \ldots, M_k$ and ideal adversary $\mathcal{S}$ proceeds as follows. Generate two random $\kappa_\mathbf{N}/2$-bit primes $\mathbf{p}$ and $\mathbf{q}$ such that $(\mathbf{p} - 1)/2$ and $(\mathbf{q} - 1)/2$ are prime and compute $\mathbf{N} = \mathbf{pq}$. Then choose $\mathbf{g}$ and $\mathbf{h}$ randomly in $\text{QR}_\mathbf{N}$. Finally, hand $((\mathcal{S}, \texttt{RSA}, \mathbf{N}, \mathbf{g}, \mathbf{h}), \{(M_j, \texttt{RSA}, \mathbf{N}, \mathbf{g}, \mathbf{h})\}_{j=1}^k)$ to $\mathcal{C}_\mathcal{I}$.

There are special purpose protocols [28, 65] for generating a joint RSA-modulus, but these are not analyzed in the UC-framework, so for technical reasons we cannot apply these directly.

In several places in the thesis we need an ideal zero-knowledge proof of knowledge functionality that can be parameterized by any NP-relation R.

Functionality 5.4 (Zero-Knowledge Proof of Knowledge). Let L_R be a language given by a binary relation R. The ideal zero-knowledge proof of knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^{R}$ of a witness w to an element $x \in L_R$, running with provers $P_1, \ldots, P_k$, verifiers $V_1, \ldots, V_l$ and ideal adversary $\mathcal{S}$ proceeds as follows

- Upon receipt of $(P_i, \texttt{Prover}, x, w)$ from $\mathcal{C}_{\mathcal{I}}$, store w under the tag (P_i, x) and ignore further messages from P_i. Let I be the set of indices j such that (P_i, x) is stored under the tag V_j. Delete these entries and hand $((\mathcal{S}, P_i, \texttt{Prover}, x, R(x, w)), \{(V_j, \texttt{Verifier}, P_i, x, R(x, w))\}_{j \in I})$ to $\mathcal{C}_{\mathcal{I}}$.
- Upon receipt of $(V_j, \texttt{Question}, P_i, x)$ from $\mathcal{C}_{\mathcal{I}}$ such that nothing is stored under the tag V_j, check if there is a witness w stored under the tag (P_i, x). If so, hand $((\mathcal{S}, V_j, \texttt{Verifier}, P_i, x, R(x, w)), (V_j, \texttt{Verifier}, P_i, x, R(x, w)))$ to $\mathcal{C}_{\mathcal{I}}$. If not, store (P_i, x) under the tag V_j.

Remark 5.5. Note that each prover is allowed to prove at most one statement, and that a verifier only is allowed to verify one proof at a time. This is to simplify our analysis.

Chapter 6

Some Practical Attacks On Mix-Nets

In this chapter we motivate a rigorous treatment of mix-nets. We do this by describing several independent practical attacks against the mix-net proposed by Golle, Zhong, Boneh, Jakobsson, and Juels [79] at Asiacrypt 2002. Our attacks illustrate that a protocol without a rigorous definition of security and a careful and detailed security proof should not be trusted, even if it is proposed by professional cryptographers in one of the major conferences.

The first attack breaks the privacy of any given sender *without corrupting any mix-server*. This attack is related to an attack of Pfitzmann [126, 125]. The second attack is similar to the attack of Desmedt and Kurosawa [59] and breaks the privacy of *all* senders. It requires that all senders are honest and that the last mix-server is corrupted. The third attack may be viewed as a novel combination of the ideas of Anderson [6], Lim and Lee [99] and Pfitzmann [126, 125]. Johan Håstad inspired us to find this attack. The attack breaks the privacy of any given sender and requires that the first and last mix-servers are corrupted. This attack breaks also Jakobsson [86], including the fixed version of Mitomo and Kurosawa [108]. The fourth attack breaks the robustness in a novel way. It requires corruption of some senders and the first mix-server. This attack breaks also the hybrid mix-net of Jakobsson and Juels [90].

The attacks in this paper are taken from Wikström [146]. In independent work Abe and Imai [5] report an attack similar to our first attack. Their paper predates [146], but not the publicly available technical report [144] referenced in [5]. We remark that Abe and Imai also give an attack against the protocol of Jakobsson and Juels [90] that is unrelated to our attacks.

6.1 A Review of "Optimistic Mixing for Exit-Polls"

We present a short review of the relevant parts of the protocol of Golle et al. [79]. The description given here is as close as possible to the original, but we avoid details

irrelevant to our attacks and change some notation to simplify the exposition of the attacks. For details we refer the reader to [79].

6.1.1 Parties and Setup

The protocol assumes the existence of a bulletin board. The parties of the protocol are N *senders*, and a relatively small number of *mix-servers*, $M_1, \ldots, M_k$. Each sender encrypts its message, and writes it on the bulletin board. The mix-servers then execute the mix-net protocol.

The protocol employs an El Gamal [71] cryptosystem in a subgroup G_q of prime order q of the multiplicative group modulo a prime p, i.e. $\mathbb{Z}_p^*$. The El Gamal cryptosystem is described in Section 3.6.

In the setup stage each mix-server M_j is somehow given a random $x_j \in \mathbb{Z}_q$, and $y_l = g^{x_l}$ for $l \neq j$. The value x_j is also shared with the other mix-servers using a threshold verifiable secret sharing scheme. Golle et al. [79] discuss different variants for sharing keys, but we choose to present a simple variant, since it has no impact on our attacks. If any mix-server M_j is deemed to be cheating the other mix-servers can verifiably reconstruct its private key x_j. The mix-servers can also compute $y = \prod_{j=1}^{k} y_j$, which gives a joint public key (g, y), with secret corresponding private key $x = \sum_{j=1}^{k} x_j$.

A practical advantage of the mix-net is that it can be used to execute several mix-sessions using the same set of keys, i.e. the El Gamal keys are not changed between mix-sessions. To be able to do this the proofs of knowledge below are bound to a mix-session identifier id that is unique to the current mix-session.

6.1.2 Sending a Message to the Mix-Net

A typical honest sender, Alice, computes the following to send a message m to the mix-net:

$$\begin{aligned} (u, v) = E_{(g,y)}(m), \qquad & w = h(u, v), \text{ and} \\ \alpha = [E_{(g,y)}(u), E_{(g,y)}(v), E_{(g,y)}(w)] \quad = \quad & [(\mu_1, \mu_2), (\nu_1, \nu_2), (\omega_1, \omega_2)] \ , \end{aligned}$$

where $h : \{0,1\}^* \to G_q$ is a hash function modeled by a random oracle.

Then Alice computes a zero-knowledge proof of knowledge $\pi_{\mathrm{id}}(u, v, w)$, in the random oracle model of u, v and w, which depends on the current mix-session identifier id. Finally, Alice writes $(\alpha, \pi_{\mathrm{id}}(u, v, w))$ on the bulletin board. We reserve the notation above for the tuple of Alice and use it in the attacks below.

6.1.3 Execution of the Mix-Net

First the mix-servers remove any duplicate inputs to the mix-net and discard input tuples that contain components not in the subgroup G_q. The mix-servers then discard all input tuples where the proof of knowledge is not valid for the current

mix-session. Let $L_0 = \{[(a_{0,i}, b_{0,i}), (c_{0,i}, d_{0,i}), (e_{0,i}, f_{0,i})]\}_{i=1}^{N}$ be the resulting list of triples of El Gamal pairs. The mixing then proceeds in the following stages.

First Stage: Re-Randomization and Mixing

This step proceeds as in all re-randomization mix-nets based on El Gamal. One by one, the mix-servers $M_1, \ldots, M_k$ randomize all the inputs and their order. (Note that the components of triples are not separated from each other during the re-randomization.) In addition, each mix-net must give a proof that the product of the plaintexts of all its inputs equals the product of the plaintexts of all its outputs. The protocol proceeds as follows.

1. Each mix-server M_j reads from the bulletin board the list L_{j-1} output by the previous mix-server.
2. The mix-server then chooses $r_{ji}, s_{ji}, t_{ji} \in \mathbb{Z}_q$, for $i = 1, \ldots, N$, randomly and computes the re-randomized list

   $$\{[(g^{r_{ji}} a_{j-1,i}, y^{r_{ji}} b_{j-1,i}), (g^{s_{ji}} c_{j-1,i}, y^{s_{ji}} d_{j-1,i}), (g^{t_{ji}} e_{j-1,i}, y^{t_{ji}} f_{j-1,i})]\}_{i=1}^{N}$$

   of triples. The above list of triples is then randomly permuted, and the resulting list: $L_j = \{[(a_{j,i}, b_{j,i}), (c_{j,i}, d_{j,i}), (e_{j,i}, f_{j,i})]\}_{i=1}^{N}$ is written on the bulletin board.
3. Define $a_j = \prod_{i=1}^{N} a_{j,i}$, and define b_j, c_j, d_j, e_j, and f_j correspondingly. The mix-server proves in zero-knowledge that $\log_g a_j/a_{j-1} = \log_y b_j/b_{j-1}$, $\log_g c_j/c_{j-1} = \log_y d_j/d_{j-1}$, and $\log_g e_j/e_{j-1} = \log_y f_j/f_{j-1}$. This implies that $D_x(a_j, b_j) = D_x(a_{j-1}, b_{j-1})$, and similarly for the pairs (c_j, d_j) and (e_j, f_j), i.e. the component-wise product of the inner triples remains unchanged by the mix-server.

Remark 6.1. Since $\log_y b_j/b_{j-1} = \log_g a_j/a_{j-1} = \sum_{i=1}^{N} r_{ji}$, and M_j knows the latter sum, the proof in Step 3) can be implemented by a zero-knowledge proof of knowledge in the random oracle model, and similarly for the pairs (c_j, d_j), and (e_j, f_j).

Second Stage: Decryption of the Inputs

1. A quorum of mix-servers jointly decrypt each triple of cryptotexts in L_k to produce a list L on the form $L = \{(u_i, v_i, w_i)\}_{i=1}^{N}$. Since the method used to do this is irrelevant to our attacks, we do not present it here.
2. All triples for which $w_i = h(u_i, v_i)$ are called *valid*.
3. Invalid triples are investigated according to the procedure described below. If the investigation proves that all invalid triples are *benign* (only senders cheated), we proceed to Step 4. Otherwise, the decryption is aborted and we continue with the back-up mixing.

4. A quorum of mix-servers jointly decrypt the cryptotexts (u_i, v_i) for all valid triples. This successfully concludes the mixing. The final output is defined as the set of plaintexts corresponding to valid triples.

Special Step: Investigation of Invalid Triples

The mix-servers must reveal the path of each invalid triple through the various permutations. For each invalid triple, starting from the last server, each server reveals which of its inputs corresponds to this triple, and how it re-randomized this triple. One of two things may happen:

- **Benign case (only senders cheated):** if the mix-servers successfully produce all such paths, the invalid triples are known to have been submitted by users. The decryption resumes after the invalid triples have been discarded.

- **Serious case (one or more servers cheated):** if one or more servers fail to recreate the paths of invalid triples, these mix-servers are accused of cheating and replaced, and the mix-net terminates producing no output. In this case the inputs are handed over to the back-up mixing procedure below.

Back-Up Mixing

The *outer-layer* encryption of the inputs posted to the mix-net is decrypted by a quorum of mix-servers. The resulting list of *inner-layer* cryptotexts becomes the input to a standard re-encryption mix-net based on El Gamal (using, for example Neff's scheme described in [112]). Then the output of the standard mix-net is given as output by the mix-net.

Remark 6.2. It is infeasible to find two lists $\{(u_i, v_i)\}_{i=1}^N \neq \{(u'_i, v'_i)\}_{i=1}^N$ such that $\prod_{i=1}^N h(u_i, v_i) = \prod_{i=1}^N h(u'_i, v'_i)$, if the product is interpreted in a group where the discrete logarithm problem is hard. This is stated as a theorem by Wagner [143], who credits Wei Dai with this observation, and appears as a lemma in Golle et al. [79].

During the re-encryption and mixing stage each mix-server proves in zero-knowledge that it leaves the component-wise product $(\prod u_i, \prod v_i, \prod w_i)$ of the inner triples (u_i, v_i, w_i) unchanged, but individual triples may still be corrupted. Then invalid triples are traced back. This leaves only valid inner triples in the output and the proofs of knowledge of each server are used to conclude that the component-wise product of these valid inner triples was left unchanged by the mix-net. Golle et al. [79] then refer to the lemma and conclude that the set of valid triples in the output is identical to the set of valid triples hidden in the double encrypted input to the mix-net.

Unfortunately, this intuitively appealing construction is flawed as we explain in Section 6.5. Furthermore, our second attack in Section 6.3 shows that it is possible to behave maliciously without changing the set of inner triples.

The goal of the adversary is to break the privacy of our typical honest sender Alice and to alter the output without detection. Each of our attacks illustrates a separate weakness of the protocol.

6.2 First Attack: Honest Mix-Servers

We show that the adversary can break the privacy of the typical sender Alice. All that is required is that it can send two messages to the mix-net which is a natural assumption in most scenarios. In the following we use the notation for the cryptotext of Alice introduced in Section 6.1.2. The attack is illustrated in Figure 6.1, and the details are as follows. Recall that $[(\mu_1, \mu_2), (\nu_1, \nu_2), (\omega_1, \omega_2)]$ is the tuple sent by Alice. The adversary does the following:

1. It chooses δ and γ randomly in $\mathbb{Z}_q$, and computes:

$$w_\delta = h(\mu_1^\delta, \mu_2^\delta), \qquad \alpha_\delta = (E_{(g,y)}(\mu_1^\delta), E_{(g,y)}(\mu_2^\delta), E_{(g,y)}(w_\delta)) \ , \text{ and}$$
$$w_\gamma = h(\mu_1^\gamma, \mu_2^\gamma), \qquad \alpha_\gamma = (E_{(g,y)}(\mu_1^\gamma), E_{(g,y)}(\mu_2^\gamma), E_{(g,y)}(w_\gamma)) \ .$$

   Then it computes the corresponding proofs of knowledge $\pi_{\mathrm{id}}(\mu_1^\delta, \mu_2^\delta, w_\delta)$ and $\pi_{\mathrm{id}}(\mu_1^\gamma, \mu_2^\gamma, w_\gamma)$. This gives the adversary two perfectly valid input tuples $(\alpha_\delta, \pi_{\mathrm{id}}(\mu_1^\delta, \mu_2^\delta, w_\delta))$, $(\alpha_\gamma, \pi_{\mathrm{id}}(\mu_1^\gamma, \mu_2^\gamma, w_\gamma))$, that it sends to the bulletin board (possibly by corrupting two senders).

2. It waits until the mix-net has successfully completed its execution. During the execution of the mix-net the mix-servers first jointly decrypt the "outer layer" of the double encrypted messages. After benign tuples have been removed the result is a list of valid triples

$$((u_1, v_1, w_1), \ldots, (u_N, v_N, w_N)). \tag{6.1}$$

   The final output of the mix-net is the result of decrypting each inner El Gamal pair (u_i, v_i) and results in a list of cleartext messages $(m_1, \ldots, m_N)$.

3. It computes the list

$$(m'_1, \ldots, m'_N) = (m_1^{\delta/\gamma}, \ldots, m_N^{\delta/\gamma}) \ ,$$

   and then finds a pair (i, j) such that $m_i = m'_j$. From this it concludes that with very high probability $m_j = u^\gamma$. Then it computes $z = m_j^{1/\gamma}$, and finds a triple (u_l, v_l, w_l) in the list (6.1) such that $z = u_l$. Finally it concludes that with very high probability m_l was the message sent by Alice to the mix-net.

Remark 6.3. At additional computational cost it suffices for the adversary to send 2 messages to break the privacy of t senders. Suppose for example that the adversary wants to break the privacy also of Bob and Camilla.

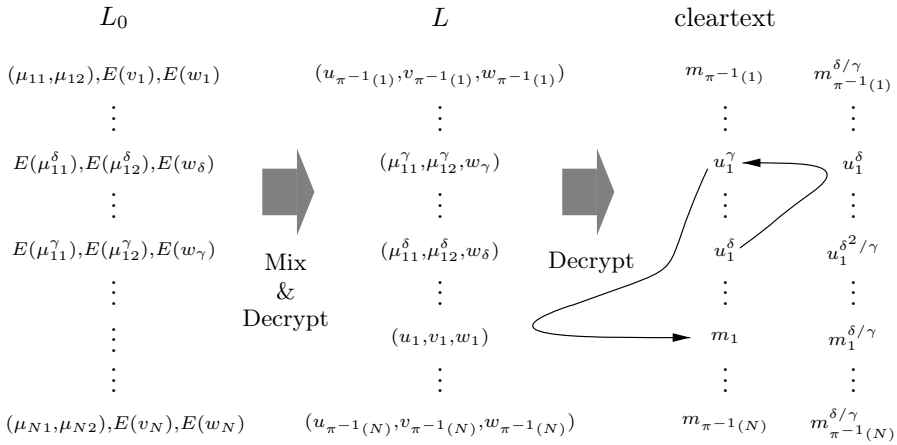

Figure 6.1: The figure illustrates the first attack. The leftmost column L_0 is the list of tuples with valid proofs of knowledge of the inner triple, the middle column L is the output of the outer decryption stage, and the rightmost columns are the cleartext and the exponentiated cleartexts respectively. The adversary follows the arrows to identify the cleartext m_1 belonging to Alice.

We assume that Bob sent m' encrypted as

$$(u', v') = E_{(g,y)}(m'), \quad w' = h(u', v'), \quad \text{and}$$
$$\alpha' = [E_{(g,y)}(u'), E_{(g,y)}(v'), E_{(g,y)}(w')] = [(\mu_1', \mu_2'), (\nu_1', \nu_2'), (\omega_1', \omega_2')] \ ,$$

and that Camilla sent m'' encrypted as

$$(u'', v'') = E_{(g,y)}(m''), \quad w'' = h(u'', v''), \quad \text{and}$$
$$\alpha'' = [E_{(g,y)}(u''), E_{(g,y)}(v''), E_{(g,y)}(w'')] = [(\mu_1'', \mu_2''), (\nu_1'', \nu_2''), (\omega_1'', \omega_2'')] \ ,$$

The first step of the attack is unchanged except that the adversary replaces (μ_1, μ_2) by $(\mu_1^{\eta}(\mu_1')^{\eta'}\mu_1'', \mu_2^{\eta}(\mu_2')^{\eta'}\mu_2'')$, where $\eta, \eta' \in \mathbb{Z}_q$ are randomly chosen. The adversary proceeds with the attack as before until it has computed z.

At this point in the original attack the adversary identifies an inner triple (u_l, v_l, w_l) such that $z = u_l$ and concludes that m_l was the message sent by Alice.

In the generalized attack the adversary instead identifies three triples (u_l, v_l, w_l), $(u_{l'}, v_{l'}, w_{l'})$ and $(u_{l''}, v_{l''}, w_{l''})$ such that $z = u_l^{\eta} u_{l'}^{\eta'} u_{l''}$. Then it concludes that m_l was the message sent by Alice, $m_{l'}$ the message sent by Bob, and $m_{l''}$ the message sent by Camilla.

The approach is generalized to higher dimensions in the natural way to break the privacy of t senders.

6.2.1 Why the Attack is Possible

The attack exploits two different flaws of the protocol. The first is that the sender of a message, e.g. Alice, proves only knowledge of the inner El Gamal pair (u, v) and the hash value $w = h(u, v)$, and not knowledge of the message m. The second flaw is that identical El Gamal keys are used for both the inner and outer El Gamal system.

Anybody can compute a single encrypted message $(\mu_1^\delta, \mu_2^\delta) = (g^{r\delta}, y^{r\delta}u^\delta) = E_{(g,y)}(u^\delta, r\delta)$ of a power u^δ of the first component u of the *inner* El Gamal pair (u, v) of the triple α sent by Alice. Anybody can also compute a proof of knowledge of $(\mu_1^\delta, \mu_2^\delta)$ and $w_\delta = h(\mu_1^\delta, \mu_2^\delta)$ (and similarly for $(\mu_1^\gamma, \mu_2^\gamma)$ and ω_γ).

The first flaw allows the adversary to input triples of El Gamal pairs with such proofs of knowledge to the mix-net. The second flaw allows the adversary to use the mix-net to decrypt $(\mu_1^\delta, \mu_2^\delta)$, and thus get its hands on u^δ (and similarly for u^γ). The adversary can then identify (u, v) as the inner El Gamal pair of Alice and break her privacy.

6.3 Second Attack: Honest Senders and One Corrupt Mix-Server

In this section we assume that all senders and all mix-servers, except the last mix-server M_k, are honest. The last mix-server M_k is corrupted by the adversary and performs the attack. The attack breaks the privacy.

To simplify our notation we write $L_0 = \{\alpha_i\}_{i=1}^N$ for the input list, where we define $\alpha_i = [(a_{0,i}, b_{0,i}), (c_{0,i}, d_{0,i}), (e_{0,i}, f_{0,i})]$ to be the tuple sent by sender S_i. Instead of following the protocol, M_k proceeds as follows in the first stage.

1. It computes the following tuples

$$\begin{aligned}(a', b', \ldots, f') &= (a_{k-1}/a_0, b_{k-1}/b_0, \ldots, f_{k-1}/f_0), \quad \text{and} \\ \alpha_1' &= [(a'a_{0,1}, b'b_{0,1}), (c'c_{0,1}, d'd_{0,1}), (e'e_{0,1}, f'f_{0,1})] \ .\end{aligned}$$

2. Then it forms the list

$$L_{k-1}' = \{\alpha_1', \alpha_2, \ldots, \alpha_N\} \ ,$$

   i.e. a copy of L_0 with the first tuple α_1 replaced by α_1'.

3. When M_k is supposed to re-randomize and permute the output L_{k-1} of M_{k-1} it instead simulates the actions of an honest mix-server on the corrupted input L_{k-1}'. The output list written to the bulletin board by the simulated mix-server is denoted L_k.

4. It waits until the inner layer has been decrypted and uses its knowledge of the permutation that relates L_k to L_0 to break the privacy of all senders.

We show that the attack goes undetected, i.e. the mix-servers decrypt the inner layer. This implies that the attack succeeds.

Firstly, consider the proof of knowledge that mix-server M_k produces during the re-encryption and mixing stage. Define

$$a'_{k-1} = (a' a_{0,1}) \prod_{i=2}^{N} a_{0,i} \ ,$$

and similarly for for b'_{k-1}, c'_{k-1}, d'_{k-1}, e'_{k-1}, and f'_{k-1}. In Step 3 above, the simulated honest mix-server outputs proofs of knowledge of the following equalities of logarithms

$$\begin{aligned} \log_g a_k/a'_{k-1} &= \log_y b_k/b'_{k-1} \ , \\ \log_g c_k/c'_{k-1} &= \log_y d_k/d'_{k-1} \ , \text{ and} \\ \log_g e_k/e'_{k-1} &= \log_y f_k/f'_{k-1} \ . \end{aligned}$$

By construction we have that

$$a'_{k-1} = (a' a_{0,1}) \prod_{i=2}^{N} a_{0,i} = a' \prod_{i=1}^{N} a_{0,i} = \frac{a_{k-1}}{a_0} a_0 = a_{k-1} \ ,$$

and similarly for b_{k-1}, c_{k-1}, d_{k-1}, e_{k-1}, and f_{k-1}. This implies that the proof of knowledge produced by M_k is deemed valid by the other mix-servers.

Secondly, consider the investigation of invalid inner triples. Tracing back invalid triples is difficult to M_k, since it does not know the re-encryption exponents and the permutation relating L_{k-1} and L_k. We show that there are no invalid inner triples. Suppose we define the sums

$$r = \sum_{j=1}^{k-1} \sum_{i=1}^{N} r_{ji}, \quad s = \sum_{j=1}^{k-1} \sum_{i=1}^{N} s_{ji}, \quad \text{and} \quad t = \sum_{j=1}^{k-1} \sum_{i=1}^{N} t_{ji} \ .$$

i.e. we form the sum of all re-encryption exponents used by all mix-servers except the last, for the first second and third El Gamal pairs respectively. Since all mix-servers except M_k are honest, we have

$$(a', b', c', d', e', f') = (g^r, y^r, g^s, y^s, g^t, y^t) \ ,$$

which implies that α'_1 is a valid re-encryption of α_1. Thus M_k does not corrupt any inner triple by simulating an honest mix-server on the input L'_{k-1}. Since all senders are honest and the set of inner triples hidden in L_0 and L'_{k-1} are identical, there are no invalid inner triples. Thus cheating is not detected.

We conclude that the mix-servers decrypt the inner triples, i.e. the privacy of *all* senders is broken.

6.3.1 Why the Attack is Possible

The second attack above exploits that the last mix-server M_k is not forced to take the output L_{k-1} of the next to last mix-server as input. This allows M_k to use a slightly modified version of L_0 instead, which breaks the privacy of all senders.

6.4 Third Attack: Two Corrupt Mix-Servers

In this section we assume that the first and last mix-servers, M_1 and M_k, are corrupted. We give an attack that breaks the privacy of any given sender Alice. Let $L_0 = \{\alpha_i\}_{i=1}^N$, where $\alpha_i = [(a_{0,i}, b_{0,i}), (c_{0,i}, d_{0,i}), (e_{0,i}, f_{0,i})]$. Without loss we let α_1 and α_2 be the tuples sent by Alice and Bob respectively. Let $\mathbb{Z}_p^* = G_q \times G_\kappa$ and let $1 \neq \zeta \in G_\kappa$. Thus ζ is an element *outside* of the group G_q. The adversary corrupts M_1 and M_k, and they proceed as follows.

1. M_1 computes two modified elements

$$\begin{aligned} \alpha_1' &= [(\zeta a_{0,1}, b_{0,1}), (c_{0,1}, d_{0,1}), (e_{0,1}, f_{0,1})] \ , \text{ and} \\ \alpha_2' &= [(\zeta^{-1} a_{0,2}, b_{0,2}), (c_{0,2}, d_{0,2}), (e_{0,2}, f_{0,2})] \ . \end{aligned}$$

   Then it forms the modified list $L_0' = \{\alpha_1', \alpha_2', \alpha_3, \ldots, \alpha_N\}$ and instructs M_1 to simulate an honest mix-server on input L_0'. Note that the first component of α_1' and α_2' respectively is no longer contained in G_q.

2. M_k simulates an honest mix-server on input L_{k-1}, but it does not write the output L_k of this simulation on the bulletin board. Let $L_k = \{\beta_i\}_{i=1}^N$, where $\beta_i = [(a_{k,i}, b_{k,i}), (c_{k,i}, d_{k,i}), (e_{k,i}, f_{k,i})]$. M_k finds $l, l' \in \{1, \ldots, N\}$ such that

$$a_{k,l}^q = \zeta^q, \quad \text{and} \quad a_{k,l'}^q = \zeta^{-q} \ .$$

   Then it computes the tuples

$$\begin{aligned} \beta_l' &= [(\zeta^{-1} a_{k,l}, b_{k,l}), (c_{k,l}, d_{k,l}), (e_{k,l}, f_{k,l})] \quad , \text{and} \\ \beta_{l'}' &= [(\zeta a_{k,l'}, b_{k,l'}), (c_{k,l'}, d_{k,l'}), (e_{k,l'}, f_{k,l'})] \ , \end{aligned}$$

   form the list

$$L_k' = \{\beta_1, \ldots, \beta_{l-1}, \beta_l', \beta_{l+1}, \ldots, \beta_{l'-1}, \beta_{l'}', \beta_{l'+1}, \ldots, \beta_N\} \ ,$$

   and writes L_k' on the bulletin board.

3. The mix-net outputs $(m_1, \ldots, m_N)$ and the adversary concludes that Alice sent m_l.

Since all mix-servers except M_1 and M_k are honest there exists $l, l' \in \{1, \ldots, N\}$ and $r, r' \in \mathbb{Z}_q$ such that

$$a_{k,l} = g^r \zeta a_{0,1}, \quad \text{and} \quad a_{k,l'} = g^{r'} \zeta^{-1} a_{0,2} \ .$$

This implies that

$$a_{k,l}^q = \zeta^q (g^r a_{0,1})^q = \zeta^q, \quad \text{and} \quad a_{k,l'}^{-q} = \zeta^{-q} \ .$$

since $\beta^q = 1$ for any $\beta \in G_q$. We have $\zeta^q \neq 1 \neq \zeta^{-q}$, since the order of ζ does not divide q. On the other hand we have $a_{k,i}^q = 1$ for $i \neq l, l'$, since $a_{k,i} \in G_q$ for $i \neq l, l'$. Thus the adversary successfully identifies Alice's cryptotext if the cheating of M_1 and M_k is not detected.

Clearly, the values of b_1, c_1, d_1, e_1, and f_1 are not changed by replacing L_0 with L_0' in Step 1. Neither is a_1, since

$$(\zeta a_{0,1})(\zeta^{-1} a_{0,2}) \prod_{i=3}^{N} a_{0,i} = \prod_{i=1}^{N} a_{0,i} = a_1 \ .$$

Similarly, b_k, c_k, d_k, e_k, and f_k are not changed by replacing L_k with L_k' in Step 2. Neither is a_k, since $(\zeta^{-1} a_{k,l})(\zeta a_{k,l'}) \prod_{i \neq l,l'}^{N} a_{k,i} = \prod_{i=1}^{N} a_{k,i}$. Similarly as in the attack of Section 6.3, we conclude that the proofs of knowledge produced by M_1 and M_k are deemed valid by the other mix-servers. If we assume that Alice and Bob are honest, their inner triples are never traced back and cheating is not detected.

If $\zeta = \zeta^{-1}$ the adversary can only conclude that Alice sent a message from the set $\{m_l, m_{l'}\}$. This breaks the privacy of Alice, but the adversary can also identify Alice's message uniquely by choosing Bob to be a corrupted sender.

Remark 6.4. Our attack may be viewed as a novel combination of the ideas in Anderson [6], Lim and Lee [99] and Pfitzmann [126, 125] in that we use elements in $\mathbb{Z}_p^* \backslash G_q$ to execute a "relation attack".

Both [6] and [99] focus on key recovery attack. The idea is to trick the holder of a private key x to compute $v = u^x$ for some element u outside G_q and then recover some bits of x by solving the logarithm of v^q in the basis u^q in G_κ. Lim and Lee explicitly propose to avoid their attacks by using the subgroup G_q of prime order q of $\mathbb{Z}_p^*$, where $p = 2q + 1$ is a strong prime. They argue that this is safe, since the adversary can only recover a single bit.

Our attack is quite different. We do not try to recover the key, but use ζ to mark elements and track them through the protocol. Thus, our attack illustrate that unless it is explicitly proved that elements can be allowed to live outside G_q, the protocol must ensure that this is the case.

6.4.1 Why the Attack is Possible

The attack exploits that a mix-server M_j does not verify that all elements in its input L_{j-1} are in G_q. M_1 uses this to "tag" elements in L_0, which lets them be identified by the last mix-server M_k.

6.5 Fourth Attack: One Corrupt Mix-Server

In Proposition 3 of Golle et al. [79] the authors claim that their protocol satisfies the following strong definition of public verifiability if there is a group G_q in which the discrete logarithm problem is hard.

Definition 1. *(Public Verifiability (cf. [79])) A mix net is* public verifiable *if there exists a polynomially bounded verifier that takes as input the transcript of the mixing posted on the bulletin board, outputs "valid" if the set of valid outputs is a permuted decryption of all valid inputs, and otherwise outputs "invalid" with overwhelming probability. Note that to prove public verifiability, we consider an adversary that can control all mix-servers and all users.*

Unfortunately, Proposition 3 in [79] is false. The following is a counter-example.

Suppose that there are 4 senders, and that the adversary corrupts two of the senders and the first mix-server M_1. Let

$$\begin{aligned}(u_i, v_i, w_i) &= (g^{r_i}, y^{r_i} m_i, h(u_i, v_i)), \quad \text{and} \\ \alpha_i &= ((a_{0,i}, b_{0,i}), (c_{0,i}, d_{0,i}), (e_{0,i}, f_{0,i})) = (E_y(u_i), E_y(v_i), E_y(w_i))\end{aligned}$$

for $i = 1, 2, 3, 4$. Then define

$$\begin{aligned}\alpha'_3 &= ((a_{0,3}, b_{0,3}), (c_{0,3}, ad_{0,3}), (e_{0,3}, f_{0,3})), \quad \text{and} \\ \alpha'_4 &= ((a_{0,4}, b_{0,4}), (c_{0,4}, a^{-1}d_{0,4}), (e_{0,4}, f_{0,4}))\end{aligned}$$

for some $1 \neq a \in G_q$. Suppose that α_1 and α_2 are sent to the mix-net by honest senders, and α'_3 and α'_4 are sent to the mix-net by the corrupted senders with corresponding proofs of knowledge.

M_1 replaces α'_3, and α'_4 with α_3 and α_4. This does not change the value of the component-wise product $d_1 = v_1 v_2 v'_3 v'_4$ since $v'_3 v'_4 = v_3 v_4$, and the cheating is not detected since α_3 and α_4 corresponds to valid inner triples and thus not traced back. On the other hand the tuples α'_3 and α'_4 correspond to invalid inner triples

$$(u_3, v_3, aw_3), \quad \text{and} \quad (u_4, v_4, a^{-1}w_4) \ .$$

We conclude that the sets of valid inner triples in the input and output respectively differ, public verifiability is broken, and Proposition 3 of [79] is false.

Some may argue that this is not important since the adversary may only choose to correct invalid messages which it has previously prepared for this particular purpose. We do not agree. Note that the adversary may *choose* whether to correct α'_3 and α'_4. If it chooses not to correct invalid triples they are simply traced back and considered benign.

The following application shows the importance of this subtlety. We use the mix-net construction to run two independent elections (using different keys). First all votes for both elections are collected, and after some time both elections are closed. Then the mix-net is executed for the first election. Finally the mix-net is

executed for the second election. In this scenario the adversary can insert specially prepared invalid triples (i.e. votes) in the second election and then *decide* whether to correct these triples *based on the outcome* of the first election. This should clearly not be allowed, but may be acceptable in certain non-voting scenarios.

6.5.1 Why the Attack is Possible

The attack exploits the fact that the first mix-server can choose whether to correct specially prepared invalid inner triples or not without detection.

6.6 Further Applications of the Attacks

All the attacks described above work also if $\mathbb{Z}_p^* \supset G_q$ are replaced by any other similarly related groups, e.g. elliptic curves.

The attacks of Section 6.2 and 6.3 exploit the particular structure of the protocol of Golle et al. [79]. We have found no other protocol vulnerable to these attacks.

In Section 6.6.1 below we sketch how the attack of Section 6.4 can be applied to break the privacy of "Flash Mix" by Jakobsson [86], including the fixed protocol proposed by Mitomo and Kurosawa [108]. We note that the latter proposal is given without security claims.

In Section 6.6.2 below we sketch how the attack of Section 6.5 breaks the robustness of Jakobsson and Juels [90].

6.6.1 Attack for "Flash Mix"

In this section we assume that the reader is familiar with Jakobsson [86] and sketch how this protocol can be broken by a natural adaptation of the novel attack of Section 6.3.

The attack is employed during the "second re-encryption". The adversary corrupts the first and the last mix-servers, M_1 and M_k, in the mix-chain. During the "second re-encryption" M_1 "tags" two arbitrary El Gamal pairs in its input by multiplying their first components with ζ and ζ^{-1} respectively (similarly as in Section 6.3). Then the honest mix-servers perform their re-encryption and mixing. When the last mix-server M_k is about to re-encrypt and mix the output of the previous mix-server M_{k-1} it identifies the "tagged" El Gamal pairs, removes the "tags" by multiplying the first components by ζ^{-1} and ζ respectively, and then finally re-encrypts and mix the resulting list honestly.

After the verification of the "first re-encryption" this breaks the privacy of some randomly chosen sender, if the cheating goes undetected. Although the attack is weak, it does break privacy.

Cheating is detected if one of two things happen; the adversary by chance chooses a "dummy element" that is later traced back through the mix-chain, or if M_1 or M_k fails to play its part in the computation of the "relative permutations" correctly. The first event is very unlikely since by construction there are very

few "dummy elements". Since the "tags" are removed by M_k, and both M_1 and M_k follow the protocol except for the use of the tags, it follows that the cheating is not detected. It is easy to see that the changes introduced by Mitomo and Kurosawa [108] do not prevent the above attack.

6.6.2 Attack for "An optimally robust hybrid mix network"

Jakobsson and Juels [90] presents a hybrid mix-net. We assume familiarity with their protocol and sketch how it is broken using the attack of Section 6.5.

Suppose that there are four senders, and that the i:th sender forms a cryptotext $(c_0^{(i)}, \mu_0^{(i)}, y_0^{(i)})$ of a message m_i. The adversary corrupts the last two senders and modifies their cryptotexts as follows before they hand them to the mix-net. It replaces $y_0^{(3)}$ by $\overline{y}_0^{(3)} = a y_0^{(3)}$ by and $y_0^{(4)}$ by $\overline{y}_0^{(4)} = a^{-1} y_0^{(4)}$ for some $a \neq 1$.

The adversary corrupts M_1. M_1 then replaces $\overline{y}_0^{(3)}$ by $y_0^{(3)}$ and $\overline{y}_0^{(4)}$ by $y_0^{(4)}$ and simulates an honest M_1 on the modified input. Similarly as in the original attack this does not change the component-wise product $P_0 = y_0^{(1)} y_0^{(2)} \overline{y}_0^{(3)} \overline{y}_0^{(4)} = y_0^{(1)} y_0^{(2)} y_0^{(3)} y_0^{(4)}$. The VerifyComplaint procedure is never invoked, since all cryptotexts are valid. Thus the cheating is not detected.

We conclude that the set of cleartext messages corresponding to the set of valid cryptotexts in the input differs from the set of cleartext messages in the output of the mix-net. This breaks the robustness, i.e. Definition 4(b) of [90].

Chapter 7

A Definition of Security of a Mix-Net

In this chapter we define what it means for a mix-net to be secure. We first discuss some informal security requirements on mix-nets. Then we formalize security in the UC-framework and describe an ideal mix-net functionality. We also argue that our definition captures the notion of a mix-net in a natural way. This chapter is based on the results in the papers Wikström [147] and Wikström [148].

7.1 Informal Requirements and Previous Definitions

The foremost security requirements a mix-net must satisfy are privacy and robustness. A mix-net is said to be private (or secure) if it preserves the anonymity of each sender. A mix-net is robust if it outputs the correct result despite that some parties are corrupted.

In most papers that propose mix-net constructions informal definitions of privacy and robustness are given, but the only rigorous definition we are aware of that predates our work is given by Abe and Imai [5]. They propose a definition of a mix-net and corresponding experiments that are meant to capture robustness and anonymity. The idea is to view a mix-net as a form of generalized decryption oracle and define privacy similarly as chosen ciphertext security of cryptosystems. Unfortunately, they do not provide any example of a mix-net that satisfies their definitions. Furthermore, they do not adress the issues of using a mix-net as a subprotocol, i.e., if it can be composed securely with other protocols. Thus, it is hard to say if these definitions are useful.

An additional security requirement apart from privacy and robustness is universal verifiability. A mix-net is universally verifiable if anybody can verify the correctness of an execution. In other words a mix-net is universally verifiable if the adversary can not modify the output without this being noticed even if it corrupts all mix-servers. We remark that universal verifiability does not imply the anonymity of the senders. If all zero-knowledge proofs in the protocol have the property that all messages of the verifiers are random strings, one can assume the existence

of a trusted source of random bits and let the mix-servers prove that they follow the protocol using the trusted random bits as challenges. Then the protocol becomes universally verifiable. Alternatively, we can simply require that the transcript of an execution of the mix-net is a non-interactive zero-knowledge proof of correctness of the execution, but this still requires a trusted source of a long random string. Furthermore, it is not known how to construct efficient non-interactive zero-knowledge proofs. An efficient non-interactive zero-knowledge proof can be constructed in the random oracle model. In short, universal verifiability requires the assumption of a trusted party, either a trusted source of random bits or a random oracle. Our constructions can be made universally verifiable, but we do not investigate this in great detail in this thesis. This is for several reasons.

It is far from obvious where to get hold of a trusted source of random bits. In practice a protocol for generation of such bits would have to be implemented using cryptographic techniques. Then security only holds under computational assumptions and/or assumptions on how many parties can be corrupted, and universal verifiability is lost. This is in fact essentially what we do in our constructions.

An alternative is to use the random oracle model, but as explained in Section 2.10 the random oracle model only gives heuristic security and should be avoided when possible. Furthermore, each party that wishes to verify the correctness of an execution must implement their own verifier, or at least fully understand an implementation on a very low level. Otherwise it can simply trust the provider of the implementation. Due to the complexity of proofs of shuffle and the level of skill needed to understand these protocols, we do not expect many organizations to implement verification software, and those that do can easily be handled by interactive protocols.

We believe that sender verifiability, i.e., the possibility for a sender to verify that its individual cryptotext is processed correctly in a simple way is far more important in practice to convince skeptic users of the correctness of an execution. In particular in the voting application. The protocol we propose and analyze in Chapter 8 has this property.

7.2 The Ideal Mix-Net

We take a different approach than Abe and Imai [5], and formalize the security of a mix-net in the UC-framework, which is described in Chapter 4.

There are several advantages of this approach. Firstly, it is relatively easy to define an ideal functionality that performs the service we expect from a "perfect" mix-net. Thus, it is easier to convince oneself that the "right" definition has been found. This should be contrasted with the complex experiments in [5] and previous informal ad-hoc definitions. Secondly, although many variations of the UC-framework have evolved, the UC-model we use is natural, robust and general. Thirdly, the requirements that must be satisfied by a protocol to securely realize an ideal functionality are very demanding. Thus, we expect that a UC-secure mix-net

is secure according to most natural definitions of security with only minor modifications. Finally, if we manage to prove the security of a mix-net protocol in the UC-framework, it can be used as a subprotocol in any setting.

We remark that not all notions are easily cast as ideal functionalities. Indeed, the ideal functionality for digital signatures first proposed by Canetti [41] was flawed. The corrected functionalities proposed by Backes and Hofheinz [9] and Canetti [42] are prohibitively complicated.

7.2.1 The Ideal G_q-Based Mix-Net

The definition of an ideal mix-net functionality presented below is essentially taken from Wikström [147] and geared towards an El Gamal based mix-net, but we use slightly different notation. The El Gamal cryptosystem is described in Chapter 3. The definition formalizes a trusted party that waits for messages in a group G_q from senders, and then when a majority of the mix-servers request it, outputs these messages but in lexicographical order. We prove the security of the El Gamal based mix-net in Chapter 8 with regards to this definition.

Functionality 7.1 (G_q-Based Mix-Net). The ideal functionality for a G_q-based mix-net, $\mathcal{F}_{\text{MN}}$, running with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows

1. Initialize a list $L = \emptyset$, a database D, a counter $c = 0$, and set $J_S = \emptyset$ and $J_M = \emptyset$.

2. Repeatedly wait for inputs

   - Upon receipt of $(S_i, \texttt{Send}, m_i)$ with $m_i \in G_q$ and $i \notin J_S$ from $\mathcal{C}_{\mathcal{I}}$, store this tuple in D under the index c, set $c \leftarrow c+1$, and hand $(\mathcal{S}, S_i, \texttt{Input}, c)$ to $\mathcal{C}_{\mathcal{I}}$.
   - Upon receipt of $(M_j, \texttt{Run})$ from $\mathcal{C}_{\mathcal{I}}$, store $(M_j, \texttt{Run})$ in D under the index c, set $c \leftarrow c + 1$, and hand $(\mathcal{S}, M_j, \texttt{Input}, c)$ to $\mathcal{C}_{\mathcal{I}}$.
   - Upon receipt of $(\mathcal{S}, \texttt{AcceptInput}, c)$ such that something is stored under the index c in D do
     a) If $(S_i, \texttt{Send}, m_i)$ with $i \notin J_S$ is stored under c, then append m_i to L, set $J_S \leftarrow J_S \cup \{i\}$, and hand $(\mathcal{S}, S_i, \texttt{Send})$ to $\mathcal{C}_{\mathcal{I}}$.
     b) If $(M_j, \texttt{Run})$ is stored under c, then set $J_M \leftarrow J_M \cup \{j\}$. If $|J_M| > k/2$, then sort the list L lexicographically to form a list L', hand $((\mathcal{S}, M_j, \texttt{Output}, L'), \{(M_l, \texttt{Output}, L')\}_{l=1}^{k})$ to $\mathcal{C}_{\mathcal{I}}$ and ignore further messages. Otherwise, hand $\mathcal{C}_{\mathcal{I}}$ the list $(\mathcal{S}, M_j, \texttt{Run})$.

The functionality differs from the one in [147] in that the adversary can prohibit senders and mix-servers to give input to the functionality. This modification is necessary when the bulletin board functionality is defined as in Section 5.3 instead of as in [147].

The functionality accepts only one input from each sender, and it can not handle several sessions, i.e., after it has output the messages in lexicographical order it halts. This is to give a more natural and simple definition, but it is not difficult to modify the definition to allow more than one input from each sender and to execute several sessions. Furthermore, the constructions in Chapter 8 and Chapter 11 are easily modified to satisfy the modified definitions.

The functionality is not completely general. In fact it is meant to define an ideal mix-net functionality that can be securely realized by an El Gamal based mix-net protocol executing in a group G_q. Thus, the input messages are expected to be in G_q. Formally, G_q is a fixed family $\{G_{q_\kappa}\}_{\kappa\in\mathbb{N}}$ of groups, where $\log_2 q_\kappa = \kappa$. Thus, when the ideal functionality is run on input 1^κ it expects inputs in G_{q_κ}. An alternative definition would be to allow inputs in $\{0,1\}^\kappa$, but this does not bring any additional security and only makes the functionality more complicated. It also makes it harder to securely realize the functionality using an El Gamal based mix-net. The problem is that in the mix-net protocol a sender must prove that its cryptotext contains a message that is contained in $\{0,1\}^\kappa$, since otherwise the adversary can easily distinguish between an execution of the real protocol, where inputs in G_q that can not be encoded as elements in $\{0,1\}^\kappa$, and interacting with the ideal functionality.

Another variation is to let the ideal mix-net functionality choose the group G_{q_κ} randomly when executed with security parameter 1^κ. For example, the ideal mix-net could choose a random $(\kappa+1)$-bit safe prime $p = 2q+1$, and define G_q as the unique subgroup of order q in $\mathbb{Z}_p^*$. Using this definition the mix-net would output a representation of G_q before accepting any inputs. The advantage of this variant is that it allows a weaker DDH-assumption. The ideal functionality above is geared towards a mix-net protocol that is secure under the DDH-assumption in a specific family of groups, whereas the variation is geared towards a mix-net protocol that is secure under the DDH-assumption of a randomly chosen group.

We use the definition above for ease of exposition. Our results are easily modified to hold with regards to the variant definition. The only essential difference is that the mix-net protocol must jointly generate a random input to the algorithm that generates the random group.

7.2.2 The Ideal Relaxed Mix-Net

To prove the security of the Paillier based mix-net of Chapter 11 we use a slightly relaxed definition of the ideal mix-net. The definition is relaxed in that corrupt senders may input messages with κ bits whereas honest senders may only input messages with $\kappa - \kappa_r$ bits. In the final output all messages are truncated to $\kappa - \kappa_r$. Here κ_r is an additional security parameter. In the protocol this security parameter is used to decide the completeness and the statistical zero-knowledge. It is chosen such that $2^{-\kappa_r}$ is negligible in κ.

Functionality 7.2 (Relaxed Mix-Net). The relaxed ideal mix-net, $\mathcal{F}_{\text{RMN}}$, running with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows

1. Initialize a list $L = \emptyset$, a database D, a counter $c = 0$, and set $J_S = \emptyset$ and $J_M = \emptyset$.

2. Repeatedly wait for new inputs and do

   - Upon receipt of $(S_i, \texttt{Send}, m_i)$ from $\mathcal{C}_\mathcal{I}$ do the following. If $i \notin J_S$ and S_i is not corrupted and $m_i \in \{0,1\}^{\kappa - \kappa_r}$ or if S_i is corrupted and $m_i \in \{0,1\}^\kappa$ then store this tuple in D under the index c, set $c \leftarrow c + 1$, and hand $(\mathcal{S}, S_i, \texttt{Input}, c)$ to $\mathcal{C}_\mathcal{I}$. Otherwise the input is ignored.
   - Upon receipt of $(M_j, \texttt{Run})$ from $\mathcal{C}_\mathcal{I}$, store $(M_j, \texttt{Run})$ in D under the index c, set $c \leftarrow c + 1$, and hand $(\mathcal{S}, M_j, \texttt{Input}, c)$ to $\mathcal{C}_\mathcal{I}$.
   - Upon receipt of $(\mathcal{S}, \texttt{AcceptInput}, c)$ such that something is stored under the index c in D do
     a) If $(S_i, \texttt{Send}, m_i)$ is stored under c and $i \notin J_S$, then append m_i to the list L, set $J_S \leftarrow J_S \cup \{i\}$, and hand $(\mathcal{S}, S_i, \texttt{Send})$ to $\mathcal{C}_\mathcal{I}$.
     b) If $(M_j, \texttt{Run})$ is stored under c, then set $J_M \leftarrow J_M \cup \{j\}$. If $|J_M| > k/2$, then truncate all strings in L to $\kappa - \kappa_r$ bits and sort the result lexicographically to form a list L'. Sort the list L to form a list L''. Then hand $((\mathcal{S}, M_j, \texttt{Output}, L''), \{(M_l, \texttt{Output}, L')\}_{l=1}^k)$ to $\mathcal{C}_\mathcal{I}$. Otherwise, hand $\mathcal{C}_\mathcal{I}$ the list $(\mathcal{S}, M_j, \texttt{Run})$.

It is hard to imagine a situation where the relaxation is a real disadvantage, but if it is, it may be possible to eliminate this beauty flaw by an erasure-free proof of membership in the correct interval is used in the submission phase of the protocol. This is discussed further in Remark 11.7 in Chapter 11.

Chapter 8

A Sender Verifiable Mix-Net

In this chapter we introduce a new type of El Gamal based mix-net in which each mix-server only decrypts and permutes its input. No re-encryption is necessary. This allows an individual sender to verify non-interactively that its message was processed correctly, and point out the corrupted mix-server when this is not the case. We call this property *sender verifiability*. Although some older constructions have this property, our is the first provably secure scheme.

Our mix-net is provably secure in the UC-framework against static adversaries corrupting any minority of the mix-servers. This holds under the decision Diffie-Hellman assumption, and assuming an ideal bulletin board and an ideal zero-knowledge proof of knowledge of a correct shuffle and an ideal proof of knowledge of the cleartext of an El Gamal cryptotext.

This chapter is based on the results in Wikström [148].

8.1 Adversary Model

In this chapter we consider a static adversary, i.e., the adversary chooses which parties to corrupt before the mix-net is executed.

Definition 8.1 (Static Mix-Net Adversaries). We define $\mathcal{M}_l$ to be the set of static adversaries that corrupt less than l out of k parties of the mix-server type, and arbitrarily many parties of the sender type.

Note that there are no requirements on the number of corrupted senders. In this respect the adversarial model is very strong. On the other hand we require that the adversary only corrupts a minority of the mix-servers. One could consider a stronger adversary with regards to the number of corrupted mix-servers, e.g. assuming the existence of a single honest mix-server. It seems that any mix-net secure against such an adversary requires the senders to be present during the phase when all cryptotexts are shuffled and decrypted. The problem is that if all mix-servers are needed to process a cryptotext, then if a single mix-server behaves

badly, all senders must resend their message using another public key and prove that their newly submitted message is identical to the original. This follows since otherwise the corrupted mix-servers can complete the computation on their own, and anonymity can not be guaranteed. The requirement that senders must be present during the execution of the protocol is a severe drawback in electronic elections. The semantics of the mix-net also changes, e.g. a sender may decide not to resend its message and potentially this can be done based on the set of originally submitted messages if the adversary corrupts the mix-server that first learns the output of the mix-net. This type of semantics may not be adequate for electronic elections.

We believe that most of our methods are applicable to the above setting as well, but we have not investigated this in detail.

8.2 On Re-encryption in El Gamal Based Mix-Nets

In recent El Gamal based mix-nets, e.g. [113, 69, 147], the mix-servers form a chain, and each mix-server randomly permutes, partially decrypts, and *re-encrypts* the output of the previous mix-server. In most older constructions decryption is instead carried out jointly at the end of the chain. Our construction is different in that each mix-server *partially decrypts* and *sorts* the output of the previous mix-server. Thus, no cryptotext is re-encrypted and the permutation is not random, but determined by the lexicographical order of the cryptotexts.

Let us consider why re-encryption is often considered necessary. In several previous mix-nets each mix-server M_j holds a private key $x_j \in \mathbb{Z}_q$ corresponding to a public key $y_j = g^{x_j}$. A joint public key $y = \prod_{j=1}^{k} y_j$ is used by a sender S_i to compute a cryptotext $(u_{0,i}, v_{0,i}) = (g^{r_i}, y^{r_i} m_i)$ of a message m_i for a random $r_i \in \mathbb{Z}_q$. The mix-servers take turns and compute

$$(u_{j,i}, v_{j,i})_{i=1}^{N} = \left(g^{s_{j,i}} u_{j-1,\pi_j(i)}, \left(\prod_{l=j+1}^{k} y_l \right)^{s_{j,i}} v_{j-1,\pi_j(i)} u_{j-1,\pi_j(i)}^{-x_j} \right)_{i=1}^{N},$$

for random $s_{j,i} \in \mathbb{Z}_q$ and $\pi_j \in \Sigma_N$, i.e., each mix-server permutes, partially decrypts and re-encrypts its input. In the end $(v_{k,i})_{i=1}^{N} = (m_{\pi(i)})_{i=1}^{N}$ for some random joint permutation π. The reason that re-encryption is necessary with this type of scheme is that otherwise the first component $u_{0,i}$ of each cryptotext remains unchanged during the transformation, which allows anybody to break the anonymity of all senders. For the older type of construction it is obvious why re-encryption is necessary.

8.3 Our Modification

We modify the El Gamal cryptosystem to ensure that also the first component $u_{j-1,i}$ is changed during partial decryption. Each mix-server is given a private key

$(w_j, x_j) \in \mathbb{Z}_q^2$ and a corresponding public key $(z_j, y_j) = (g^{w_j}, g^{x_j})$. To partially decrypt and permute its input it computes

$$(u_{j-1,i}^{1/w_j}, v_{j-1,i} u_{j-1,i}^{-x_j/w_j})_{i=1}^{N} \ , \tag{8.1}$$

from L_{j-1}, and sorts the result lexicographically. The result is denoted by $L_j = (u_{j,i}, v_{j,i})_{i=1}^{N}$. Note that both components of each cryptotext are transformed using the private key of the mix-server. For this transformation to make any sense we must also modify the way the joint key is formed. We define

$$(\bar{z}_{k+1}, \bar{y}_{k+1}) = (g, 1) \quad \text{and} \quad (\bar{z}_j, \bar{y}_j) = (\bar{z}_{j+1}^{w_j}, \bar{y}_{j+1} \bar{z}_{j+1}^{x_j}) \ . \tag{8.2}$$

The joint keys must be computed jointly by the mix-servers. A sender encrypts its message using the public key $(\bar{z}_1, \bar{y}_1)$, i.e., $(u_{0,i}, v_{0,i}) = (\bar{z}_1^{r_i}, \bar{y}_1^{r_i} m_i)$ for some random r_i. The structure of the keys is chosen such that a cryptotext on the form $(u_{j-1,i}, v_{j-1,i}) = (\bar{z}_j^{r_i}, \bar{y}_j^{r_i} m_i)$ given as input to mix-server M_j satisfies

$$\begin{aligned} (u_{j-1,i}^{1/w_j}, v_{j-1,i} u_{j-1,i}^{-x_j/w_j}) &= (\bar{z}_j^{r_i/w_j}, \bar{y}_j^{r_i} \bar{z}_j^{-r_i x_j/w_j} m_i) \\ &= ((\bar{z}_j^{1/w_j})^{r_i}, (\bar{y}_j \bar{z}_j^{-x_j/w_j})^{r_i} m_i) = (\bar{z}_{j+1}^{r_i}, \bar{y}_{j+1}^{r_i} m_i) \ . \end{aligned}$$

Thus, each mix-server M_j transforms a cryptotext $(u_{j-1,i}, v_{j-1,i})$ encrypted with the public key $(\bar{z}_j, \bar{y}_j)$ into a cryptotext $(u_{j,i}, v_{j,i})$ encrypted with the public key $(\bar{z}_{j+1}, \bar{y}_{j+1})$. Note that $\mathsf{Sort}(\{v_{k,i}\}_{i=1}^{N}) = \mathsf{Sort}(\{m_i\}_{i=1}^{N})$, since $\bar{y}_{k+1} = 1$.

There are several seemingly equivalent ways to set up the scheme, but some of these do not allow a reduction of the security of the mix-net to the DDH-assumption. In fact the relation in Equation (8.1) is carefully chosen to allow such a reduction.

8.4 Sender Verifiability

An important consequence of our modification is that a sender can compute a pair $(\bar{z}_{j+1}^{r_i}, \bar{y}_{j+1}^{r_i} m_i)$ and verify that this pair is contained in L_j for $j = 1, \ldots, k$. Furthermore, if this is not the case the sender can easily prove to any outsider that its message was tampered with. We call this *sender verifiability*, since it allows a sender to verify non-interactively that its cryptotext is processed correctly by the mix-servers. This is not a new property. In fact Chaum's original construction [45] has this property, but our construction is the first provably secure scheme with this property.

We think that sender verifiability is an important property that deserves more attention. The verification process is unconditional and easily explained to anybody with only a modest background in mathematics, and a verification program can be implemented with little skills in programming. This means that in the main application of mix-nets, electronic elections, a sender can convince herself that her vote was processed correctly without a deep understanding of cryptography. We stress that this verification does not guarantee anonymity or correct processing of

any other cryptotext. Thus, a proof of the overall security of the mix-net is still required.

The reader may worry that sender verifiability allows a voter to point out its vote to a coercer. This is the case, but the sender can do this in previous mix-nets as well by pointing at its message in the original list L_0 of cryptotexts and revealing the randomness used during encryption, so this problem is not specific to our scheme. Furthermore, our scheme becomes coercion-free whenever the sender does not know the randomness of its cryptotext, as other El Gamal based mix-nets, but sender verifiability is then lost.

8.5 A Technical Advantage

There is also an important technical advantage of the lack of re-encryption in the mixing process. The witness of our shuffle relation consists of a pair (w_j, x_j), which makes it easy to turn our proof of knowledge into a secure realization of the ideal functionality $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$. This should be contrasted with all previous shuffle relations, where the witness contains a long list of random exponents used to re-encrypt the input that must somehow be extracted by the ideal adversary in the UC-setting.

A potential alternative to our approach is to formalize the proof of a shuffle as a proof of membership [116] in the UC-framework. However, a proof of membership is not sufficient for the older constructions where decryption is carried out jointly at the end of the mixing chain. The problem is that the adversary could corrupt the last mix-server M_k and instruct it to output L_0 instead of a re-encryption and permutation of L_{k-1}. This would obviously break the anonymity of all senders. The malicious behavior is not detected, since the ideal proof of membership only expects an element in the language and no witness from corrupted parties, and L_0 is in fact a re-encryption and permutation of L_{k-1}. In Chapter 11 we show that a proof of knowledge in the classical sense with rewinding is sufficient to construct a universally composable mix-net.

It is an open question if a proof of membership suffices for mix-nets where each mix-server *partially decrypts* and then re-encrypts and permutes its input.

8.6 Additional Ideal Functionalities

We describe the mix-net in a hybrid model as defined in the UC-framework. This means that the mix-servers and senders have access to a set of ideal functionalities introduced in this section.

We assume the existence of an ideal authenticated bulletin board functionality $\mathcal{F}_{\mathrm{BB}}$ as defined in Functionality 5.1 in Section 5.3. Recall that all parties can write to it, but no party can erase any message from it.

We also assume an ideal functionality corresponding to the key set-up sketched in Section 8.3. This is given below.

Functionality 8.2 (Special El Gamal Private Key Sharing). The ideal Special El Gamal Private Key Sharing functionality over G_q, $\mathcal{F}_{\text{SKG}}$, with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows.

1. Initialize sets $J_j = \emptyset$ for $j = 0, \ldots, k$.

2. Until $|J_0| = k$, repeatedly wait for inputs. If $(M_j, \texttt{MyKey}, w_j, x_j)$ is received from $\mathcal{C}_{\mathcal{I}}$ such that $w_j, x_j \in \mathbb{Z}_q$ and $j \notin J_0$. Set $J_0 \leftarrow J_0 \cup \{j\}$ compute $z_j = g^{w_j}$ and $y_j = g^{x_j}$, and hand $(\mathcal{S}, \texttt{PublicKey}, M_j, w_j, z_j)$ to $\mathcal{C}_{\mathcal{I}}$.

3. Set $(\bar{z}_{k+1}, \bar{y}_{k+1}) = (g, 1)$ and $(\bar{z}_j, \bar{y}_j) = (\bar{z}_{j+1}^{w_j}, \bar{y}_{j+1}\bar{z}_{j+1}^{x_j})$. Then hand $((\mathcal{S}, \texttt{PublicKeys}, (\bar{z}_j, \bar{y}_j, z_j, y_j)_{j=1}^k), \{(S_i, \texttt{PublicKeys}, (\bar{z}_j, \bar{y}_j, z_j, y_j)_{j=1}^k)\}_{i=1}^N$, $\{(M_l, \texttt{Keys}, w_l, x_l, (\bar{z}_j, \bar{y}_j, z_j, y_j)_{j=1}^k)\}_{l=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$.

4. Repeatedly wait for inputs. If $(M_j, \texttt{Recover}, M_l)$ is received from $\mathcal{C}_{\mathcal{I}}$, set $J_l \leftarrow J_l \cup \{j\}$. If $|J_l| > k/2$, then hand $((\mathcal{S}, \texttt{Recovered}, M_l, w_l, x_l)$, $\{(M_j, \texttt{Recovered}, M_l, w_l, x_l)\}_{j=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$, and otherwise hand $(\mathcal{S}, M_j, \texttt{Recover}, M_l)$ to $\mathcal{C}_{\mathcal{I}}$.

The above functionality can be securely realized by letting each mix-server secret share its private key using Feldman's [63] verifiable secret sharing scheme. Note that the functionality explicitly allows corrupted mix-servers to choose their keys in a way that depends on the public keys of uncorrupted mix-servers. The special joint keys would then be computed iteratively using Equation (8.2), and during this process each mix-server would prove that it does this correctly. We do not investigate this solution in detail, but general methods can be used [44] to securely realize the functionality in the common random string model.

Each mix-server partially decrypts each cryptotext and sorts the resulting list of cryptotexts. Thus, proving correct behavior corresponds to proving knowledge of a private key (w, x) such that the cryptotexts (u_i, v_i) input to a mix-server are related to the cryptotexts (u_i', v_i') it outputs by the following relation.

Definition 8.3 (Knowledge of Correct Decryption-Permutation). Define for each N a relation $R_{\text{DP}} \subset (G_q^3 \times G_q^{2N} \times G_q^{2N}) \times (\mathbb{Z}_q \times \mathbb{Z}_q)$, by

$$((g, z, y, \{(u_i, v_i)\}_{i=1}^N, \{(u_i', v_i')\}_{i=1}^N), (w, x)) \in R_{\text{DP}}$$

precisely when $z = g^w$, $y = g^x$ and $(u_i', v_i') = (u_{\pi(i)}^{1/w}, v_{\pi(i)} u_{\pi(i)}^{-x/w})$ for $i = 1, \ldots, N$ and some permutation $\pi \in \Sigma_N$ such that the list $\{(u_i', v_i')\}_{i=1}^N$ is sorted lexicographically.

To avoid a large class of "relation attacks" [126, 125, 146] no sender can be allowed to construct a cryptotext of a message related to the message encrypted by some other sender. To ensure this each sender is required to prove knowledge of the randomness it uses to form its cryptotexts. This corresponds to the following relation.

Definition 8.4 (Knowledge of Cleartext). Define a relation $R_{\mathrm{C}} \subset G_q^4 \times \mathbb{Z}_q$ by $((\bar{z}, \bar{y}, u, v), r) \in R_{\mathrm{C}}$ precisely when $\log_{\bar{z}} u = r$.

Formally, we need the ideal zero-knowledge functionality given as Functionality 5.4 in Section 5.3 parameterized by the two relations above, i.e., we need the functionalities $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$. The first functionality is securely realized in Chapter 10.2 and the second in Chapter 10.1.

8.7 The Mix-Net

We now give the details of our mix-net. It executes in the $(\mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{PKG}}, \mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}, \mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}})$-hybrid model.

Protocol 8.5 (El Gamal Based Mix-Net). The mix-net protocol $\pi_{\mathrm{MN}} = (S_1, \ldots, S_N, M_1, \ldots, M_k)$ consists of senders S_i, and mix-servers M_j.

Sender S_i. Each sender S_i proceeds as follows.

1. Wait for $(\texttt{PublicKeys}, (\bar{z}_j, \bar{y}_j, z_j, y_j)_{j=1}^k)$ from $\mathcal{F}_{\mathrm{SKG}}$.
2. Wait for an input $(\texttt{Send}, m_i)$, $m_i \in G_q$. Then choose $r_i \in \mathbb{Z}_q$ randomly and compute $(u_i, v_i) = E_{(\bar{z}_1, \bar{y}_1)}(m_i, r_i) = (\bar{z}_1^{r_i}, \bar{y}_1^{r_i} m_i)$.
3. Hand $(\texttt{Prover}, (\bar{z}_1, \bar{y}_1, u_i, v_i), r_i)$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$.
4. Hand $(\texttt{Write}, (u_i, v_i))$ to $\mathcal{F}_{\mathrm{BB}}$.

Mix-Server M_j. Each mix-server M_j proceeds as follows.

1. Choose $w_j, x_j \in \mathbb{Z}_q$ randomly and hand $(\texttt{MyKey}, w_j, x_j)$ to $\mathcal{F}_{\mathrm{SKG}}$.
2. Wait for $(\texttt{Keys}, (w_j, x_j), (\bar{z}_j, \bar{y}_j, z_j, y_j)_{j=1}^k)$ from $\mathcal{F}_{\mathrm{SKG}}$.
3. Wait until an input $(\texttt{Run})$ is received or more than $k/2$ different mix-servers have written $\texttt{Run}$ on $\mathcal{F}_{\mathrm{BB}}$. If the first event occurs, then hand $(\texttt{Write}, \texttt{Run})$ to $\mathcal{F}_{\mathrm{BB}}$.
4. Wait until more than $k/2$ different mix-servers have written $\texttt{Run}$ on $\mathcal{F}_{\mathrm{BB}}$, and let the last entry of this type be $(c_{\texttt{Run}}, M_i, \texttt{Run})$.
5. Form the list $L_* = \{(u_\gamma, v_\gamma)\}_{\gamma \in I_*}$, for some index set I_*, by choosing for $\gamma = 1, \ldots, N$ the entry $(c, S_\gamma, (u_\gamma, v_\gamma))$ on $\mathcal{F}_{\mathrm{BB}}$ with the smallest $c < c_{\mathrm{run}}$ such that $u_\gamma, v_\gamma \in G_q$, if present. To do this read all entries on $\mathcal{F}_{\mathrm{BB}}$ with index less than c_{run}.
6. For each $\gamma \in I_*$ do the following,
   a) Hand $(\texttt{Question}, S_\gamma, (\bar{z}_1, \bar{y}_1, u_\gamma, v_\gamma))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$.
   b) Wait for $(\texttt{Verifier}, S_\gamma, (\bar{z}_1, \bar{y}_1, u_\gamma, v_\gamma), b_\gamma)$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$.

Then form $L_0 = \{(u_{0,i}, v_{0,i})\}_{i=1}^{N'}$ consisting of pairs (u_γ, v_γ) such that $b_\gamma = 1$.

7. For $l = 1, \ldots, k$ do

   a) If $l = j$, then compute

   $$L_j = \{(u_{j,i}, v_{j,i})\}_{i=1}^{N'} = \mathsf{Sort}(\{(u_{j-1,i}^{1/w_j}, v_{j-1,i} u_{j-1,i}^{-x_j/w_j})\}_{i=1}^{N'}) \ ,$$

   Finally hand $(\texttt{Prover}, (g, z_j, y_j, L_{j-1}, L_j), (w_j, x_j))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, and hand $(\texttt{Write}, (\texttt{List}, L_j))$ to $\mathcal{F}_{\mathrm{BB}}$.

   b) If $l \neq j$, then do

      i. Wait until an entry $(c, M_l, (\texttt{List}, L_l))$ appears on $\mathcal{F}_{\mathrm{BB}}$, where L_l is on the form $\{(u_{l,i}, v_{l,i})\}_{i=1}^{N'}$ for $u_{l,i}, v_{l,i} \in G_q$.

      ii. Hand $(\texttt{Question}, M_l, (g, z_l, y_l, L_{l-1}, L_l))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, and wait for $(\texttt{Verifier}, M_l, (g, z_l, y_l, L_{l-1}, L_l), b_l)$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$.

      iii. If $b_l = 0$, then hand $(\texttt{Recover}, M_l)$ to $\mathcal{F}_{\mathrm{SKG}}$, and wait for $(\texttt{Recovered}, M_l, (w_l, x_l))$ from $\mathcal{F}_{\mathrm{SKG}}$. Then compute

      $$L_l = \{(u_{l,i}, v_{l,i})\}_{i=1}^{N'} = \mathsf{Sort}(\{(u_{l-1,i}^{1/w_l}, v_{l-1,i} u_{l-1,i}^{-x_l/w_l})\}_{i=1}^{N'}) \ .$$

8. Output $(\texttt{Output}, \mathsf{Sort}(\{v_{k,i}\}_{i=1}^{N'}))$.

The number of mix-servers actually doing any shuffling of cryptotexts can be reduced to $\lfloor k/2 \rfloor + 1$ without any loss in security, thus reducing the overall complexity. This follows since any set of $\lfloor k/2 \rfloor + 1$ mix-servers can recover all the private keys anyway. We state the protocol in a symmetrical way for sake of simplicity.

Theorem 8.6. *The ideal functionality $\mathcal{F}_{\mathrm{MN}}$ is securely realized by π_{MN} in the $(\mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{SKG}}, \mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}, \mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}})$-hybrid model with respect to $\mathcal{M}_{k/2}$-adversaries under the DDH-assumption in G_q.*

Proof. We describe an ideal adversary $\mathcal{S}(\cdot)$ that runs any hybrid adversary $\mathcal{A}$ as a black-box. Then we show that if $\mathcal{S}(\mathcal{A})$ does not simulate $\mathcal{A}$ sufficiently well, we can break the DDH-assumption.

The Ideal Adversary $\mathcal{S}$. Let I_S and I_M be the set of indices of parties corrupted by $\mathcal{A}$ of the sender type and the mix-server type respectively. The ideal adversary $\mathcal{S}$ corrupts the dummy parties $\tilde{S}_i$ for which $i \in I_S$, and the dummy parties $\tilde{M}_i$ for which $i \in I_M$. The ideal adversary is best described by starting with a copy of the original hybrid ITM-graph

$$(V, E) = \mathcal{Z}'(\mathcal{H}(\mathcal{A}, \pi^{(\tilde{\pi}_1^{\mathcal{F}_{\mathrm{BB}}}, \tilde{\pi}_2^{\mathcal{F}_{\mathrm{SKG}}}, \tilde{\pi}_3^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}}, \tilde{\pi}_4^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}})})) \ ,$$

where $\mathcal{Z}$ is replaced by a machine $\mathcal{Z}'$.

The adversary $\mathcal{S}$ simulates all machines in V except the corrupted machines S_i for $i \in I_S$ and M_i for $i \in I_M$ under $\mathcal{A}$'s control. We now describe how each machine is simulated.

$\mathcal{S}$ simulates the machines S_i, $i \notin I_S$ and the ideal functionalities $\mathcal{F}_{\mathrm{BB}}$, $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ and $\mathcal{F}_{\mathrm{SKG}}$ honestly. All M_j for $j \notin I_M$ are also simulated honestly, except for M_l, where l is chosen as the maximal index not in I_M, i.e. the last honest mix-server. The machine M_l plays a special role.

Simulation of Links $(\mathcal{Z}, \mathcal{A})$, $(\mathcal{Z}, S_i)$ for $i \in I_S$, and $(\mathcal{Z}, M_j)$ for $j \in I_M$. $\mathcal{S}$ simulates $\mathcal{Z}'$, $\tilde{S}_i$, for $i \in I_S$, and $\tilde{M}_j$ for $j \in I_M$, such that it appears as if $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ and S_i for $i \in I_S$, and $\mathcal{Z}$ and M_j for $j \in I_M$ are linked directly.

1. When $\mathcal{Z}'$ receives m from $\mathcal{A}$, m is written to $\mathcal{Z}$, by $\mathcal{S}$. When $\mathcal{S}$ receives m from $\mathcal{Z}$, m is written to $\mathcal{A}$ by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and $\mathcal{A}$ are linked directly.

2. When $\mathcal{Z}'$ receives m from S_i for $i \in I_S$, m is written to $\mathcal{Z}$ by $\tilde{S}_i$. When $\tilde{S}_i$, $i \in I_S$, receives m from $\mathcal{Z}$, m is written to S_i by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and S_i are linked directly for $i \in I_S$.

3. When $\mathcal{Z}'$ receives m from M_j for $j \in I_M$, m is written to $\mathcal{Z}$ by $\tilde{M}_j$. When $\tilde{M}_j$, $j \in I_M$, receives m from $\mathcal{Z}$, m is written to M_j by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and M_j are linked directly for $j \in I_M$.

Extraction from Corrupt Mix-Servers and Simulation of Honest Mix-Servers. When a corrupt mix-server M_j, for $j \in I_M$, writes `Run` on $\mathcal{F}_{\mathrm{BB}}$, $\mathcal{S}$ must make sure that $\tilde{M}_j$ sends (`Run`) to $\mathcal{F}_{\mathrm{MN}}$. Otherwise it may not be possible to deliver an output to honest mix-servers at a later stage. If an honest dummy mix-server $\tilde{M}_j$, for $j \notin I_M$, receives (`Run`) from $\mathcal{Z}$, $\mathcal{S}$ must make sure that M_j receives (`Run`) from $\mathcal{Z}'$. In both instances $\mathcal{S}$ must do this in two steps, first sending and then instructing $\mathcal{F}_{\mathrm{MN}}$ to accept the submitted message. If an honest mix-server M_j, for $j \notin I_M$, outputs $(\texttt{Output}, L')$, $\mathcal{S}$ must make sure that $\tilde{M}_j$ does the same. This is done as follows.

1. Let $j \in I_M$. If $\mathcal{F}_{\mathrm{BB}}$ receives $(M_j, \texttt{Write}, \texttt{Run})$, then $\mathcal{S}$ continues the simulation until $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, c, M_j, \texttt{Run})$ to $\mathcal{C}_{\mathcal{I}}$. Then the simulation of $\mathcal{F}_{\mathrm{BB}}$ is interrupted and $\tilde{M}_j$ hands (`Run`) to $\mathcal{F}_{\mathrm{MN}}$. When $\mathcal{S}$ receives $(tilde M_j, \texttt{Input}, c')$ from $\mathcal{C}_{\mathcal{I}}$ it stores the pair (c, c') and continues the simulation of $\mathcal{F}_{\mathrm{BB}}$.

2. Let $j \notin I_M$. If $\mathcal{S}$ receives $(\mathcal{S}, \texttt{Input}, c', \tilde{M}_j)$ from $\mathcal{C}_{\mathcal{I}}$, then $\mathcal{Z}'$ hands (`Run`) to M_j and continues the simulation until $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, c, \texttt{Input}, \tilde{M}_j, \texttt{Run})$ to $\mathcal{C}_{\mathcal{I}}$. Then the pair (c, c') is stored and the simulation is continued.

3. If $\mathcal{F}_{\mathrm{BB}}$ receives $(\mathcal{A}, M_j, \texttt{AcceptInput}, c)$ the simulation of $\mathcal{F}_{\mathrm{BB}}$ is interrupted. If there is a pair (c, c') for some c', then $\mathcal{S}$ hands $(\mathcal{F}_{\mathrm{MN}}, \texttt{AcceptInput}, c')$ to

$\mathcal{C}_{\mathcal{I}}$ and waits until it receives $(\mathcal{S}, \tilde{M}_j, \texttt{Run})$ or $((\mathcal{S}, \tilde{M}_j, \texttt{Output}, L'), \{(\tilde{M}_l, \tau_l)\}_{l=1}^{k})$ from $\mathcal{C}_{\mathcal{I}}$. Then the simulation of $\mathcal{F}_{\mathrm{BB}}$ is continued.

4. Let $j \notin I_M$. If $\mathcal{Z}'$ receives $(\texttt{Output}, L')$ from M_j, $\mathcal{S}$ sends $(1, \tau_j)$ to $\mathcal{C}_{\mathcal{I}}$, i.e. $\mathcal{S}$ instructs $\mathcal{C}_{\mathcal{I}}$ to deliver $(\texttt{Output}, L')$ to $\tilde{M}_j$.

Extraction from Corrupt Senders and Simulation of Honest Senders. If a corrupt sender S_i, for $i \in I_S$, in the hybrid protocol produces a cryptotext and informs $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ such that its input is deemed valid, then $\mathcal{S}$ must instruct $\tilde{S}_i$ to hand this as input to $\mathcal{F}_{\mathrm{MN}}$.

When an honest dummy sender $\tilde{S}_i$, for $i \notin I_S$, receives a message m_i from $\mathcal{Z}$, $\mathcal{S}$ must ensure that S_i receives some message m_i' from $\mathcal{Z}'$. But $\mathcal{S}$ cannot see m_i, and must therefore hand some other message $m_i' \neq m_i$ to S_i, and then later correct this flaw in the simulation before $\mathcal{A}$ or $\mathcal{Z}$ notice it. This is done as follows.

1. Let $i \in I_S$. Until $\mathcal{S}$ receives $((\mathcal{S}, M_j, \texttt{Output}, L'), \{(M_l, \tau_l)\}_{l=1}^{k})$ from $\mathcal{C}_{\mathcal{I}}$.

   a) If $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ receives a message $(S_i, \texttt{Prover}, (\bar{z}_1, \bar{y}_1, u_i, v_i), r_i)$ such that $((\bar{z}_1, \bar{y}_1, u_i, v_i), r_i) \in R_{\mathrm{DP}}$, then consult the first database of $\mathcal{F}_{\mathrm{BB}}$ and look for a pair $(c, S_i, (u_i, v_i))$.

   b) If $\mathcal{F}_{\mathrm{BB}}$ receives $(S_i, \texttt{Write}, (u_i, v_i))$ then look if $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ stored r_i under $(S_i, (\bar{z}_1, \bar{y}_1, u_i, v_i))$ such that $((\bar{z}_1, \bar{y}_1, u_i, v_i), r_i) \in R_{\mathrm{DP}}$.

   If such a pair $[(c, S_i, (u_i, v_i)), (S_i, (\bar{z}_1, \bar{y}_1, u_i, v_i), r_i)]$ is found then $\tilde{S}_i$ sends $m_i = v_i \bar{y}_1^{-r_i}$ to $\mathcal{F}_{\mathrm{MN}}$ and $\mathcal{S}$ waits until it receives $(\mathcal{S}, \tilde{S}_i, \texttt{Input}, c')$ from $\mathcal{C}_{\mathcal{I}}$. Then (c, c') is stored and the simulation, of $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ or $\mathcal{F}_{\mathrm{BB}}$ respectively, is continued.

2. Let $i \notin I_S$. When $\mathcal{S}$ receives $(\mathcal{S}, \tilde{S}_i, \texttt{Input}, c')$ then $\mathcal{Z}'$ hands a randomly chosen message $m_i' \in G_q$ to S_i. By definition S_i chooses a random $r_i' \in \mathbb{Z}_q$ and forms its cryptotext as $(u_i, v_i) = (\bar{z}_1^{r_i'}, \bar{y}_1^{r_i'} m_i')$. Note that this corresponds to a pair of random elements in G_q. Then the simulation is interrupted when $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, \texttt{Input}, c, S_i, (u_i, v_i))$ to $\mathcal{C}_{\mathcal{I}}$. Then $\mathcal{S}$ stores (c, c') and continues the simulation.

3. If $\mathcal{F}_{\mathrm{BB}}$ receives $(\mathcal{A}, S_i, \texttt{AcceptInput}, c)$ the simulation of $\mathcal{F}_{\mathrm{BB}}$ is interrupted. If there is a pair (c, c') for some c', then $\mathcal{S}$ hands $(\mathcal{F}_{\mathrm{MN}}, \texttt{AcceptInput}, c')$ to $\mathcal{C}_{\mathcal{I}}$ and waits until it receives $(\mathcal{S}, \tilde{S}_i, (u_i, v_i))$. Then the simulation of $\mathcal{F}_{\mathrm{BB}}$ is continued.

How M_l and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ correct the flaw in the simulation. $\mathcal{S}$ must make sure that the faulty messages $m_i' \neq m_i$ introduced during simulation of honest senders, because it does not know the real messages m_i of the honest dummy parties $\tilde{S}_i$ for $i \in I_S$, are not noticed. This is done by modifying M_l and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ as follows.

1. If $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$ receives a tuple $(M_j, \texttt{Question}, M_l, (g, z_l, y_l, L_{l-1}, L_l))$ it verifies that a tuple on the form $(M_l, \texttt{Prover}, (g, z_l, y_l, L_{l-1}, L_l), \cdot)$ has been received. If so it sets $b = 1$ and otherwise $b = 0$. Finally, it hands $((\mathcal{A}, M_j, \texttt{Verifier}, M_l, (g, z_l, y_l, L_{l-1}, L_l), b)$, $(M_j, \texttt{Verifier}, M_l, (g, z_l, y_l, L_{l-1}, L_l), b))$ to $\mathcal{C}_{\mathcal{I}}$.

2. Note that by construction $\mathcal{S}$ has received $((\mathcal{S}, M_j, \texttt{Output}, L'), \ldots)$, i.e. it knows the output L'. Let $\{m_{\pi(i)}\}_{i=1}^{N'}$ be the messages in L', where $m_{\pi(i)}$ is the message sent by S_i for all $i \in I_S$. Note that $\mathcal{S}$ knows r_i and m_i for all $i \in I_S$, since it simulated the handing of these to $\mathcal{F}_{\text{MN}}$ itself. On the other hand the simulator has no further knowledge of π.

   M_l does the following instead of Step 7a in the protocol. It chooses $r_i \in \mathbb{Z}_q$ randomly, for $i \notin I_S$, and computes the list

$$L_l = \{(u_{l,i}, v_{l,i})\}_{i=1}^{N'} = \mathsf{Sort}\left(\left\{\left(\bar{z}_{l+1}^{r_i}, \bar{y}_{l+1}^{r_i} m_i\right)\right\}_{i=1}^{N'}\right) \ .$$

   Finally it hands $(\texttt{Prover}, (g, z_l, y_l, L_{l-1}, L_l), \cdot)$ to $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$, and it hands $(\texttt{Write}, (\texttt{List}, L_l))$ to $\mathcal{F}_{\text{BB}}$.

The first step ensures that $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$ plays along with M_l and pretends to other M_j that M_l did prove his knowledge properly. The second step ensures that M_l corrects the flaw in the simulation introduced by $\mathcal{S}$ at the point when it did not know the messages sent by honest dummy parties $\tilde{S}_i$, for $i \notin I_S$.

Note that the cryptotexts of all corrupted parties are identically distributed as in the real protocol. The same randomness r_i is used to encrypt the message m_i sent by S_i for $i \in I_S$.

This concludes the definition of the ideal adversary $\mathcal{S}$.

Reaching a Contradiction. Next we show, using a hybrid argument, that if the ideal adversary $\mathcal{S}$ defined above does not imply the security of Protocol 8.5, then we can break the DDH-assumption.

Suppose that $\mathcal{S}$ does not imply the security of the protocol. Then there exists a hybrid adversary $\mathcal{A}$, an environment $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$, a constant $c > 0$ and an infinite index set $\mathcal{N} \subset \mathbb{N}$ such that for $n \in \mathcal{N}$

$$|\Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\text{MN}}})) = 1] - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}', \pi^{(\tilde{\pi}_1^{\mathcal{F}_{\text{BB}}}, \tilde{\pi}_2^{\mathcal{F}_{\text{SKG}}}, \tilde{\pi}_3^{\mathcal{F}_{\text{ZK}}^{R_{\text{C}}}}, \tilde{\pi}_4^{\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}})}))) = 1]| \geq \frac{1}{n^c}$$

where $\mathcal{S}$ runs $\mathcal{A}$ as a black-box as described above, i.e. $\mathcal{S} = \mathcal{S}(\mathcal{A})$.

Defining the Hybrids. Without loss we assume that $\{1, \ldots, N\} \backslash I_S = \{1, \ldots, \eta\}$, and define an array of hybrid machines $T_0, \ldots, T_\eta$. Set $T_0 = \mathcal{Z}_z(\mathcal{I}(\mathcal{S}(\mathcal{A}), \tilde{\pi}^{\mathcal{F}_{\text{MN}}}))$, and then define T_s by the following modification to T_0.

1. When $\mathcal{S}$ receives $(\tilde{S}_i, \texttt{Send})$ from $\mathcal{F}_{\text{MN}}$, for $i \notin I_S$, it checks if $i \in \{1, \ldots, s\}$.

a) If so it consults the storage of $\mathcal{F}_{\text{MN}}$ to find the message m_i sent by $\tilde{S}_i$. Then $\mathcal{Z}'$ sends m_i to S_i and treats i as if $i \in I_S$ in the simulation of M_l. This means that $r_i = r'_i$, and $m'_i = m_i$.

b) Otherwise $\mathcal{Z}'$ sends a random message $m'_i \in \mathbb{Z}_q$ to S_i, as in the original simulation.

By inspection of the constructions we see that the output of T_η is identically distributed to the output of $\mathcal{Z}_z(\mathcal{H}(\mathcal{A}', \pi^{(\tilde{\pi}_1^{\mathcal{F}_{\text{BB}}}, \tilde{\pi}_2^{\mathcal{F}_{\text{SKG}}}, \tilde{\pi}_3^{\mathcal{F}_{\text{ZK}}^{R_{\text{C}}}}, \tilde{\pi}_4^{\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}})}))$ since the only essential difference is that M_l does not hand knowledge of his transformation to $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$, but $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$ ignores M_l's inability so this is not discovered by $\mathcal{A}$ or $\mathcal{Z}$.

If we set $p_s = \Pr[T_s = 1]$, we have $\frac{1}{\kappa^c} \leq |p_0 - p_\eta| \leq \sum_{i=1}^{\eta} |p_{s-1} - p_s|$, which implies that there exists some fixed $0 < s \leq \eta$ such that $|p_{s-1} - p_s| \geq \frac{1}{\eta\kappa^c} \geq \frac{1}{N\kappa^c}$.

Defining a Distinguisher. We are now finally ready to define a distinguisher D for the experiment considered in Lemma 3.8, i.e., a variation of the DDH-experiment.

D is confronted with the following test. An oracle first chooses $r, w, r', x, r'' \in \mathbb{Z}_q$ and a bit $b \in \{0, 1\}$ randomly.

1. If $b = 0$, then it defines $(a, y, u, z, v) = (g^r, g^w, g^{r'}, g^x, g^{r''})$.
2. If $b = 1$, then it defines $(a, y, u, z, v) = (g^r, g^w, g^{rw}, g^x, g^{rx})$.

The distinguisher D must guess the value of b, i.e., which type of input it gets.

It also replaces (z_l, y_l) by (z, y) in M_l's key generation and instructs $\mathcal{F}_{\text{SKG}}$ not to complain about the lack of corresponding secret keys (w_l, x_l). This does not change the distribution of this key and thus does not change any of the hybrids. D computes $\bar{z}_j$ and $\bar{y}_j$ as follows. It computes $\bar{z}_l = z_l^{\prod_{j=l+1}^{k} w_j}$, and for $j \neq l$ it computes $\bar{z}_j = \bar{z}_{j+1}^{w_j}$ as usual. It computes $\bar{y}_l = \bar{y}_{l+1} y_l^{\prod_{j=l+1}^{k} w_j}$, and for $j \neq l$ it computes $\bar{y}_j = \bar{y}_{j+1} \bar{z}_{j+1}^{x_j}$ as usual. Note that the scheme is chosen to allow the simulator to generate $\bar{z}_j$ and $\bar{y}_j$ without knowledge of w or x.

Since M_l appears to behave honestly (with the help of $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$), the fact that M_l does not know $w = \log_g z_l$ or $x = \log_g y_l$ is never revealed, and since less than $k/2$ mix-servers are corrupted w or x is never be recovered. The reader should think of r as the randomness of a sender, w as w_l and x as x_l. D simulates T_s until S_s receives the message $(\texttt{Send}, m_s)$, at which point it forms a cryptotext

$$(u_s, v_s) = \left(u^{\prod_{j \neq l} w_j}, a^{\sum_{j \neq l} x_j \prod_{j'=j+1}^{k} w_{j'}} v^{\prod_{j=l+1}^{k} w_j} m_s \right) .$$

Note that if $b = 1$, we have $(u_s, v_s) = (\bar{z}_1^r, \bar{y}_1^r m_s)$. The sender S_s is modified to hand $(\texttt{Write}, (u_s, v_s))$ to $\mathcal{F}_{\text{BB}}$, and the tuple $(\texttt{Prover}, (\bar{z}_1, \bar{y}_1, u_s, v_s), 1)$ to $\mathcal{F}_{\text{ZK}}^{R_{\text{C}}}$. Furthermore, $\mathcal{F}_{\text{ZK}}^{R_{\text{C}}}$ is modified to a handle this message as if we had $((\bar{z}_1, \bar{y}_1, u_s, v_s), 1) \in R_{\text{C}}$, i.e. it lies on S_i's behalf. Finally, we must change the way M_l forms its output. Suppose that $(u_{j,l-1}, v_{j,l-1})$ corresponds to the input (u_s, v_s), which can be verified

as above. Then instead of decrypting this pair M_l replaces it by

$$(u'_s, v'_s) = \left(a^{\prod_{j=l+1}^{k} w_j}, a^{\sum_{j=l+1}^{k} x_j \prod_{j'=j+1}^{k} w_{j'}} m_s\right) = (\bar{z}_{l+1}^r, \bar{y}_{l+1}^r m_s) \ .$$

D then continues the simulation of T_s until it outputs a bit b', which is then output by D.

If $b = 0$, then u and v are random elements in G_q. This implies that (u_s, v_s) is identically distributed to the corresponding cryptotext in the simulation and the output of D is identically distributed to the output of T_{s-1}.

If $b = 1$, then (u_s, v_s) is a valid encryption of m_s with randomness r, and (u'_s, v'_s) is a partial decryption of (u_s, v_s) using the private keys $w_l = w$ and $x_l = x$. This implies that the output of D is identically distributed to the output of T_s.

We conclude that

$$\begin{aligned} &|\Pr[D(g^r, g^w, g^{rw}, g^x, g^{rx}) = 1] - \Pr[D(g^r, g^w, g^{r'}, g^x, g^{r''}) = 1]| \\ &= |p_{s-1} - p_s| \geq \frac{1}{Nn^c} \ . \end{aligned}$$

Lemma 3.8 in Section 3.5 shows that this contradicts the DDH-assumption. □

Chapter 9

A New Efficient Proof of A Shuffle

In this chapter we present a new approach to construct an efficient zero-knowledge proof of knowledge of a witness of a correct decryption-permutation of El Gamal cryptotexts. This protocol is then used in Chapter 10 to securely realize the ideal decryption-permutation functionality $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$. Our approach is not a variation of existing methods [112, 70, 80]. It is based on a novel idea of independent interest, and we argue that it is at least as efficient as previous constructions. We also give a zero-knowledge proof of knowledge of correct re-encryption of Paillier cryptotexts.

This chapter is based on the results in Wikström [148].

9.1 An Informal Description of Our Approach

The protocol for proving the relation R_{DP} is complex, but the underlying ideas are simple. To simplify the exposition we follow the example of Neff [112, 113] and consider the problem of proving that a list of elements in G_q are exponentiated and permuted. We also omit numerous technical details. In particular we remove several blinding factors, hence the protocols are not zero-knowledge as sketched here. More precisely, let $y, u_1, \ldots, u_N, u'_1, \ldots, u'_N \in G_q$ be defined by $y = g^x$ and $u'_i = u_{\pi(i)}^x$ for a permutation π. Only the prover knows x and π and it must show that the elements satisfy such a relation.

9.1.1 Extraction Using Linear Independence

Recall that the security parameter κ_r denotes the size of the random padding in proofs of knowledge over groups of unknown order. We also denote by κ_p an additional security parameter, which is used as follows. The verifier chooses a list $P = (p_i)_{i=1}^N \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]^N$ of random primes and computes $U = \prod_{i=1}^N u_i^{p_i}$. Then it requests that the prover computes $U' = \prod_{i=1}^N (u'_i)^{p_{\pi(i)}}$, proves that $U' = U^x$ and that it knows a permutation π such that $U' = \prod_{i=1}^N (u'_i)^{p_{\pi(i)}}$.

The idea is then that if a prover succeeds in doing this it can be rewound and run several times with different random vectors P_j, giving different U_j and U'_j, until a set $P_1, \ldots, P_N$ of linearly independent vectors in $\mathbb{Z}_q^N$ are found. Linear independence implies that there are coefficients $a_{l,j} \in \mathbb{Z}_q$ such that $\sum_{j=1}^N a_{l,j} P_j$ equals the lth unity vector e_l, i.e., the vector with a one in the lth position and all other elements zero. We would then like to conclude that

$$u_l^x = \left(\prod_{j=1}^N U_j^{a_{l,j}} \right)^x = \prod_{j=1}^N (U'_j)^{a_{l,j}} = \prod_{j=1}^N \left(\prod_{i=1}^N (u'_i)^{p_{j,\pi(i)}} \right)^{a_{l,j}} = u'_{\pi^{-1}(l)} \ , \qquad (9.1)$$

since that would imply that the elements satisfy the shuffle-relation.

9.1.2 Proving a Permutation of Prime Exponents

The prover can use standard techniques to prove knowledge of integers $\rho_1, \ldots, \rho_N$ such that $U' = \prod_{i=1}^N (u'_i)^{\rho_i}$, but it must also prove that $\rho_i = p_{\pi(i)}$ for some permutation π.

Suppose that $\prod_{i=1}^N p_i = \prod_{i=1}^N \rho_i$ over $\mathbb{Z}$. Then unique factorization in $\mathbb{Z}$ implies that each ρ_i equals some product of the p_i and -1. If in addition we demand that $\rho_i \in [-2^{\kappa_p} + 1, 2^{\kappa_p} - 1]$, no such product can contain more than one factor. This implies that every product must contain exactly one factor. Thus, $\rho_i = \pm p_{\pi(i)}$ for some permutation π. If we also have $\sum_{i=1}^N p_i = \sum_{i=1}^N \rho_i$, then we must clearly have $\rho_i = p_{\pi(i)}$.

We observe that proving the above is relatively simple over a group of unknown order such as the group $\mathrm{QR}_{\mathbf{N}}$ of squares modulo an RSA modulus $\mathbf{N}$. The prover forms commitments

$$\mathbf{b}_0 = \mathbf{g} \ , \quad (\mathbf{b}_i, \mathbf{b}'_i)_{i=1}^N = (\mathbf{h}^{t_i} \mathbf{b}_{i-1}^{p_{\pi(i)}}, \mathbf{h}^{t'_i} \mathbf{g}^{p_{\pi(i)}})_{i=1}^N \ ,$$

with random t_i and t'_i and proves, using standard methods, knowledge of ρ_i, τ_i, τ'_i such that

$$U' = \prod_{i=1}^N (u'_i)^{\rho_i} \ , \quad \mathbf{b}_i = \mathbf{h}^{\tau_i} \mathbf{b}_{i-1}^{\rho_i} \ , \quad \text{and} \quad \mathbf{b}'_i = \mathbf{h}^{\tau'_i} \mathbf{g}^{\rho_i} \ . \qquad (9.2)$$

Note that $\mathbf{b}_N = \mathbf{h}^{\tau} \mathbf{g}^{\prod_{i=1}^N \rho_i}$ for some τ, so the verifier can check that $\prod_{i=1}^N \rho_i = \prod_{i=1}^N p_i$ by asking the prover to show that it knows τ such that $\mathbf{b}_N / \mathbf{g}^{\prod_{i=1}^N p_i} = \mathbf{h}^{\tau}$. Under the strong RSA-assumption equality then holds over the integers, since otherwise one may extract a non-trivial RSA-root as defined in Definition 3.11. We then note that a standard proof of knowledge over a group of unknown order also gives an upper bound on the bit-size of the exponents, i.e., it implicitly proves that $\rho_i \in [-2^{\kappa_p + \kappa_r} + 1, 2^{\kappa_p + \kappa_r} - 1]$. If $\kappa_r < \kappa_p$ this is sufficient to conclude that $\rho_i = \pm p_{\pi(i)}$. Finally, since $\prod_{i=1}^N \mathbf{b}'_i = \mathbf{h}^{\tau'} \mathbf{g}^{\sum_{i=1}^N \rho_i}$ for a $\tau' = \sum_{i=1}^N \tau'_i$, the verifier can check that $\sum_{i=1}^N \rho_i = \sum_{i=1}^N p_i$ by asking the prover to show that it knows τ' such that $\prod_{i=1}^N \mathbf{b}'_i / \mathbf{g}^{\sum_{i=1}^N p_i} = \mathbf{h}^{\tau'}$.

9.1.3 Fixing a Permutation

In Equation (9.1) above it is assumed that a fixed permutation π is used for all prime vectors $P_1, \ldots, P_N$. Unfortunately, this is not necessarily the case, i.e., the permutation used in the jth proof may depend on j and we should really write π_j.

To solve this technical problem we force the prover to commit to a fixed permutation π before it receives the prime vector P. The commitment is on the form $(w_i)_{i=1}^N = (g^{r_i'} g_{\pi^{-1}(i)})_{i=1}^N$, where $g, g_1, \ldots, g_N \in G_q$ are randomly chosen generators of G_q. The verifier then computes $W = \prod_{i=1}^N w_i^{p_i}$ and the prover proves that $W = g^{r'} \prod_{i=1}^N g_i^{\rho_i}$ in addition to Equations (9.2). The idea is that the prover must use π to permute the ρ_i or find a non-trivial representation of $1 \in G_q$ using $g, g_1, \ldots, g_N$, which is infeasible under the DL-assumption.

9.2 A Proof of Knowledge of a Shuffle of El Gamal Cryptotexts

In this section we describe our proof of a shuffle in detail. Although we consider a decrypt-permutation relation, our approach can be generalized to a proof of a shuffle for the other shuffle relations considered in the literature. In Section 9.7 we adapt the proof to prove a re-encryption shuffle of Paillier cryptotexts. In [149] we also consider the two standard types of El Gamal based shuffle relations.

We introduce several security parameters. We use κ to denote the number of bits in q, the order of the group G_q, and similarly $\kappa_{\mathbf{N}}$ to denote the number of bits in the RSA-modulus $\mathbf{N}$. We use κ_p to denote the number of bits used in the random primes mentioned above. At some point in the protocol the verifier hands a challenge to the prover. We use κ_c to denote the number of bits in this challenge. At several points exponents must be padded with random bits to achieve statistical zero-knowledge, as is standard for proofs of knowledge of exponents over groups of unknown order. We use κ_r to denote the number of additional random bits used to do this. We assume that the security parameters are chosen such that $\kappa_p + \kappa_c + \kappa_r < \kappa, \kappa_{\mathbf{N}}$, and $\kappa_r < \kappa_p - 2$. Below the protocol we explain how the informal description above relates to the different components of the protocol.

Protocol 9.1 (Proof of Decryption-Permutation). The common input consists of an RSA-modulus $\mathbf{N}$ and $\mathbf{g}, \mathbf{h} \in \mathrm{QR}_{\mathbf{N}}$, generators $g, g_1, \ldots, g_N \in G_q$, a public key $(z, y) \in G_q^2$, and two lists $L = (u_i, v_i)_{i=1}^N$ and $L' = (u_i', v_i')_{i=1}^N$ in G_q^{2N}. The private input to the prover consists of $(w, x) \in \mathbb{Z}_q^2$ such that $(z, y) = (g^w, g^x)$ and $(u_i', v_i') = (u_{\pi(i)}^{1/w}, v_{\pi(i)} u_{\pi(i)}^{-x/w})$ for a permutation $\pi \in \Sigma_N$ such that L' is lexicographically sorted.

1. The prover chooses $r_i' \in \mathbb{Z}_q$ randomly, computes $(w_i)_{i=1}^N = (g^{r_i'} g_{\pi^{-1}(i)})_{i=1}^N$, and hands $(w_i)_{i=1}^N$ to the verifier.
2. The verifier chooses random primes $p_1, \ldots, p_N \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$, and hands $(p_i)_{i=1}^N$ to the prover.

3. Both parties compute $(U, V, W) = (\prod_{i=1}^{N} u_i^{p_i}, \prod_{i=1}^{N} v_i^{p_i}, \prod_{i=1}^{N} w_i^{p_i})$.

4. The prover chooses the following elements randomly $k_1, k_2, k_3, k_4, k_5 \in \mathbb{Z}_q$, $l_1, \ldots, l_7, l_{r'}, l_{1/w}, l_{x/w}, l_w, l_x \in \mathbb{Z}_q$, $t_i, t_i' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r} - 1]$, $s_i, s_i' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, $r_i \in [0, 2^{\kappa_p+\kappa_c+\kappa_r} - 1]$ for $i = 1, \ldots, N$, $s \in [0, 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} - 1]$, and $s' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r+\kappa_c+\log_2 N} - 1]$. Then the prover computes

$$(b_1, b_2) = (g^{k_1} U^{1/w}, g^{k_2} U^{x/w}) \tag{9.3}$$

$$(b_3, b_4, b_5) = (g_1^{k_3} g^{1/w}, g_1^{k_4} b_3^{x}, g_1^{k_5} b_3^{w}) \tag{9.4}$$

$$(\beta_1, \beta_2) = \left(g^{l_1} U^{l_{1/w}}, g^{l_2} U^{l_{x/w}}\right) \tag{9.5}$$

$$(\beta_3, \beta_4) = (g_1^{l_3} g^{l_{1/w}}, g_1^{l_6} g^{l_{x/w}}) \tag{9.6}$$

$$(\beta_5, \beta_6, \beta_7, \beta_8, \beta_9) = (g_1^{l_4} b_3^{l_x}, g^{l_x}, g_1^{l_5} b_3^{l_w}, g^{l_w}, g_1^{l_7}) \tag{9.7}$$

$$(\alpha_1, \alpha_2, \alpha_3) = \left(g^{l_1} \prod_{i=1}^{N} (u_i')^{r_i}, g^{-l_2} \prod_{i=1}^{N} (v_i')^{r_i}, g^{l_{r'}} \prod_{i=1}^{N} g_i^{r_i}\right) \tag{9.8}$$

$$\mathbf{b}_0 = \mathbf{g} \tag{9.9}$$

$$(\mathbf{b}_i, \mathbf{b}_i')_{i=1}^{N} = (\mathbf{h}^{t_i} \mathbf{b}_{i-1}^{p_{\pi(i)}}, \mathbf{h}^{t_i'} \mathbf{g}^{p_{\pi(i)}})_{i=1}^{N} \tag{9.10}$$

$$(\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_i')_{i=1}^{N} = (\mathbf{h}^{s_i} \mathbf{b}_{i-1}^{r_i}, \mathbf{h}^{s_i'} \mathbf{g}^{r_i})_{i=1}^{N} \tag{9.11}$$

$$(\boldsymbol{\gamma}, \boldsymbol{\gamma}') = (\mathbf{h}^{s}, \mathbf{h}^{s'}) \ , \tag{9.12}$$

and $((b_i)_{i=1}^{5}, (\beta_i)_{i=1}^{9}, (\alpha_1, \alpha_2, \alpha_3), (\mathbf{b}_i, \mathbf{b}_i')_{i=1}^{N}, (\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_i')_{i=1}^{N}, (\boldsymbol{\gamma}, \boldsymbol{\gamma}'))$ is handed to the verifier.

5. The verifier chooses $c \in [2^{\kappa_c - 1}, 2^{\kappa_c} - 1]$ randomly and hands c to the prover.

6. Define $t = t_N + p_{\pi(N)}(t_{N-1} + p_{\pi(N-1)}(t_{N-2} + p_{\pi(N-2)}(t_{N-3} + p_{\pi(N-3)}(\ldots))))$, $t' = \sum_{i=1}^{N} t_i'$, $r' = \sum_{i=1}^{N} r_i' p_i$, $k_6 = k_4 + k_3 x$, and $k_7 = k_5 + k_3 w$. The prover computes

$$\begin{aligned}
(f_i)_{i=1}^{7} &= (ck_i + l_i)_{i=1}^{7} && \bmod q \\
f_{1/w} &= c/w + l_{1/w} && \bmod q \\
f_{x/w} &= cx/w + l_{x/w} && \bmod q \\
f_w &= cw + l_w && \bmod q \\
f_x &= cx + l_x && \bmod q \\
f_{r'} &= cr' + l_{r'} && \bmod q \\
(e_i, e_i')_{i=1}^{N} &= (ct_i + s_i, ct_i' + s_i')_{i=1}^{N} && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} \\
(d_i)_{i=1}^{N} &= (cp_{\pi(i)} + r_i)_{i=1}^{N} && \bmod 2^{\kappa_p+\kappa_c+\kappa_r} \\
e &= ct + s && \bmod 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} \\
e' &= ct' + s' && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_r+\kappa_c+\log_2 N}
\end{aligned}$$

and hands $(((f_i)_{i=1}^7, f_{1/w}, f_{x/w}, f_w, f_x, f_{r'}), (e_i, e_i')_{i=1}^N, (d_i)_{i=1}^N, (e, e'))$ to the verifier.

7. The verifier checks that $b_i, \beta_i, \alpha_i \in G_q$, and that L' is lexicographically sorted and that

$$(b_1^c\beta_1, b_2^c\beta_2) = (g^{f_1}U^{f_{1/w}}, g^{f_2}U^{f_{x/w}}) \tag{9.13}$$

$$(b_3^c\beta_3, b_4^c\beta_4) = (g_1^{f_3}g^{f_{1/w}}, g_1^{f_6}g^{f_{x/w}}) \tag{9.14}$$

$$(b_4^c\beta_5, y^c\beta_6) = (g_1^{f_4}b_3^{f_x}, g^{f_x}) \tag{9.15}$$

$$(b_5^c\beta_7, z^c\beta_8, (b_5/g)^c\beta_9) = (g_1^{f_5}b_3^{f_w}, g^{f_w}, g_1^{f_7}) \tag{9.16}$$

$$(b_1^c\alpha_1, (V/b_2)^c\alpha_2, W^c\alpha_3) = \left(g^{f_1}\prod_{i=1}^N (u_i')^{d_i}, g^{-f_2}\prod_{i=1}^N (v_i')^{d_i}, g^{f_{r'}}\prod_{i=1}^N g_i^{d_i}\right) \tag{9.17}$$

$$(\mathbf{b}_i^c\boldsymbol{\gamma}_i, (\mathbf{b}_i')^c\boldsymbol{\gamma}_i')_{i=1}^N = (\mathbf{h}^{e_i}\mathbf{b}_{i-1}^{d_i}, \mathbf{h}^{e_i'}\mathbf{g}^{d_i})_{i=1}^N \tag{9.18}$$

$$(\mathbf{g}^{-\prod_{i=1}^N p_i}\mathbf{b}_N)^c\boldsymbol{\gamma} = \mathbf{h}^e \tag{9.19}$$

$$\left(\mathbf{g}^{-\sum_{i=1}^N p_i}\prod_{i=1}^N \mathbf{b}_i'\right)^c \boldsymbol{\gamma}' = \mathbf{h}^{e'} \ . \tag{9.20}$$

Equations (9.3)–(9.7) are used to prove $(b_1, V/b_2) = (g^{\kappa_1}U^{1/w}, g^{-\kappa_2}VU^{-x/w})$ using standard Schnorr-like proofs of knowledge of logarithms. Equations (9.10) contain commitments corresponding to those in the outline of our approach. Equations (9.11) are used to prove knowledge of exponents τ_i, τ_i', ρ_i such that $(\mathbf{b}_i, \mathbf{b}_i') = (\mathbf{h}^{\tau_i}\mathbf{b}_{i-1}^{\rho_i}, \mathbf{h}^{\tau_i'}\mathbf{g}^{\rho_i})$. We remark that the verifier need not check that the elements $\mathbf{b}_i, \mathbf{b}_i', \boldsymbol{\gamma}_i, \boldsymbol{\gamma}_i', \boldsymbol{\gamma}, \boldsymbol{\gamma}' \in \mathrm{QR}_{\mathbf{N}}$ for our analysis to go through. Equations (9.12) are used to prove that $\prod_{i=1}^N \rho_i = \prod_{i=1}^N p_i$ and $\sum_{i=1}^N \rho_i = \sum_{i=1}^N p_i$, i.e., that ρ_i in fact equals $p_{\pi(i)}$ for some permutation π. Equations (9.8) is used to prove that $(b_1, V/b_2)$ also equals $(g^{k_1}\prod_{i=1}^N (u_i^{1/w_j})^{p_i}, g^{-k_2}\prod_{i=1}^N (v_i u_i^{-x_j/w_j})^{p_i})$. If the two ways of writing b_1 and b_2 are combined we have

$$(U^{1/w}, VU^{-x/w}) = \left(\prod_{i=1}^N (u_i^{1/w_j})^{p_i}, \prod_{i=1}^N (v_i u_i^{-x_j/w_j})^{p_i}\right) \ ,$$

which by the argument in Section 9.1 implies that $((g, z, y, L, L'), (w, x)) \in R_{\mathrm{DP}}$.

9.3 Generation of Prime Vectors From a Small Number of Public Coins

In our protocol the verifier must generate vectors in $\mathbb{Z}_q^N$ such that each component is a "randomly" chosen prime in $[2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$. In our application this must be done

openly, i.e., the randomness used to define the random primes must be generated jointly by the mix-servers.

We describe two algorithms $\mathsf{PGen}^{\mathsf{c}}$ and PGen for doing this. Both takes a list of integers and random bits used as internal randomness as input and then try to extract a list of primes.

We define $\mathsf{PGen}^{\mathsf{c}}$ as follows. Let $p(n)$ be the smallest prime at least as large as n. Our generator $\mathsf{PGen}^{\mathsf{c}}$ takes as input N random integers $n_1, \ldots, n_N \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$ and internal randomness r, and defines $p_i = p(n_i)$. To find p_i it first redefines n_i such that it is odd by incrementing by one if necessary. Then it executes the Miller-Rabin primality test for $n_i, n_i + 2, n_i + 4, \ldots$ until it finds a prime.

We put an explicit bound on the running time of the generator by bounding the number of integers it considers and the number of iterations of the Miller-Rabin test it performs in total. If the generator stops due to one of these bounds it outputs $\perp$.

The generator is not allowed to test more than $N' = \ln n(N + \sqrt{N\kappa_p})$ integers, and it is not allowed to do more than $N'' = N\kappa_p + 2(N' + \sqrt{N'\kappa_p})$ iterations of the Miller-Rabin test in total.

Let $p_N(n_1, \ldots, n_{N'})$ denote the first N primes in the list of integers $n_1, \ldots, n_{N'}$ if they exist. We define PGen as follows. It takes as input N' random integers $n_1, \ldots, n_{N'} \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$ and internal randomness r and then executes the Miller-Rabin primality test for $n_1, n_2, \ldots$ until it has found N probable primes. We bound the running time of PGen in the same way we bound the running time of $\mathsf{PGen}^{\mathsf{c}}$.

The generator $\mathsf{PGen}^{\mathsf{c}}$ (or PGen) can be used to turn the protocol above into a protocol where all messages of the verifier are random strings. The verifier sends $(n_1, \ldots, n_N, r)$ (or $(n_1, \ldots, n_{N'}, r)$) to the prover instead of $p_1, \ldots, p_N$ and the prover and verifier generates the primes used in the protocol by computing $(p_1, \ldots, p_N) = \mathsf{PGen}^{\mathsf{c}}(n_1, \ldots, n_N, r)$ (or $(p_1, \ldots, p_N) = \mathsf{PGen}(n_1, \ldots, n_{N'}, r)$).

The primes output by PGen are randomly distributed by definition, but the primes output by $\mathsf{PGen}^{\mathsf{c}}$ may be biased. A result by Baker and Harman [10] implies that the bias is relatively small.

Theorem 9.2 (cf. [10]). *For large integers n there exists a prime in $[n - n^{0.535}, n]$.*

Corollary 9.3. *For large κ_p it holds that for every prime p in the interval $[2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$ we have $\Pr[p(n) = p] \leq 2^{-0.465(\kappa_p - 1)}$, where the probability is taken over a random choice of $n \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$*

The corollary gives a very pessimistic bound. It is commonly believed that the theorem is true with 0.465 replaced by any constant less than one. Furthermore, Cramér argues probabilistically that there is a prime in every interval $[n - \log^2 n, n]$. See Ribenboim [131] for a discussion on this.

We must argue that the generators fail with negligible probability. There are two ways the generators can fail. Either they output $p_1, \ldots, p_N$, where $p_i \neq p(n_i)$ for some i, or they output $\perp$.

Lemma 9.4. *The probability that* $\mathsf{PGen}^{\mathsf{c}}(n_1,\ldots,n_N,r) \neq (p(n_1),\ldots,p(n_N))$ *conditioned on* $\mathsf{PGen}^{\mathsf{c}}(n_1,\ldots,n_N,r) \neq \perp$ *is negligible. Similarly, the probability that* $\mathsf{PGen}(n_1,\ldots,n_{N'},r) \neq p(n_1,\ldots,n_{N'})$ *conditioned on* $\mathsf{PGen}(n_1,\ldots,n_{N'},r) \neq \perp$ *is negligible.*

Proof. We prove the first claim. The proof of the second claim is almost identical and omitted. Suppose that $\mathsf{PGen}^{\mathsf{c}}(n_1,\ldots,n_N,r) \neq \perp$. Then $\mathsf{PGen}^{\mathsf{c}}$ outputs a list of integers $(p_1,\ldots,p_N)$ and for each integer n between n_i and p_i an iteration of the Miller-Rabin primality test has found a witness that n is composite. Thus, there exists no primes between n_i and p_i. The probability that p_i is considered to prime despite that it is not, is bounded by $2^{-\kappa_p}$ since κ_p iterations of the Miller-Rabin test are executed. The union bound implies that the probability that $p_i \neq p(n_i)$ for any i is bounded by $N2^{-\kappa_p}$, which is negligible. □

Using the prime number theorem it is not hard to bound the probability that $\mathsf{PGen}(n_1,\ldots,n_{N'},r) = \perp$. Unfortunately, the current understanding of the distribution of the primes does not allow a strict analysis of the probability that $\mathsf{PGen}^{\mathsf{c}}(n_1,\ldots,n_N,r) = \perp$. Instead we give a heuristic analysis in Cramér's probabilistic model of the primes defined below.

Definition 9.5 (Cramér's Model). For each integer n, let X_n be an independent binary random variable such that $\Pr[X_n = 1] = 1/\ln n$. An integer n is said to be prime* if $X_n = 1$.

The idea is to consider the primality of the integers as a typical outcome of the sequence $(X_n)_{n\in\mathbb{Z}}$. Thus, when we analyze the generator we assume that the primality of an integer n is given by X_n, and our analysis is both over the internal randomness of $\mathsf{PGen}^{\mathsf{c}}$ and the randomness of X_n.

The lemma below follows by a standard argument, but we need a precise bound to estimate the complexity of the protocol.

Lemma 9.6. *In Cramér's model the probability that* $\mathsf{PGen}^{\mathsf{c}}(n_1,\ldots,n_N,r) = \perp$ *is negligible. The probability that* $\mathsf{PGen}(n_1,\ldots,n_{N'},r) = \perp$ *is negligible.*

Proof. We prove the first claim. The second claim follows similarly using the prime number theorem.

We must bound the probability that the generator needs too many invocations of Miller-Rabin or too many iterations in total.

Denote by Y_i the event that the integer tested in the ith invocation of Miller-Rabin is a prime. For simplicity we assume that no integer is tested twice for primality. Thus, we may assume that the Y_is are independent. Set $Y = \sum_{i=1}^{N'} Y_i$. Then $\mathrm{E}[Y] = \frac{N'}{\ln n} = N + \sqrt{N\kappa_p}$. We bound the probability that the generator fails by having to check to many integers by

$$\Pr[Y \leq N] = \Pr\left[Y \leq \mathrm{E}[Y] - \sqrt{N\kappa_p}\right] \leq e^{-2\kappa_p} \ .$$

Suppose now that at least N of the Y_i are ones. We must bound the probability that the generator need more than N'' iterations in the Miller-Rabin test. At most N primes are tested and for each such prime κ_p iterations are needed giving $N\kappa_p$ iterations for the primes. Along the way a number of composite numbers are tested. Our bound stipulates that at most $2(N' + \sqrt{N'\kappa_p})$ iterations can be spent on composites, and at most N' integers can be tested at all. Denote by Z_i the indicator variable for the event that the ith iteration of the Miller-Rabin test, when run on a composite outputs "composite". Define $Z = \sum_{i=1}^{2(N'+\sqrt{N'\kappa_p})} Z_i$. In each iteration of the Miller-Rabin test on a composite the test outputs "composite" with probability at least 1/2 so $\Pr[Z_i] \geq \frac{1}{2}$ and $\mathrm{E}[Z] \geq N' + \sqrt{N'\kappa_p}$.

The probability of failure is bounded by the probability that the generator is not allowed to perform more Miller-Rabin tests despite that there are more integers to be tested. This probability is captured below.

$$\Pr\left[Z \leq N'\right] = \Pr\left[Z \leq \mathrm{E}[Z] - \sqrt{N'\kappa_p}\right] \leq e^{-2\kappa_p} \ .$$

Thus, it follows that the probability that the generator outputs $\perp$ negligible. □

Although we now have a protocol it requires many random bits. This can be avoided by use of a pseudo-random generator PRG as suggested by Groth [81]. Instead of choosing $(n_1, \ldots, n_N, r)$ (or $(n_1, \ldots, n_{N'}, r)$) randomly and sending these integers to the prover, the verifier chooses a random seed $s \in [0, 2^\kappa - 1]$ and hands this to the prover. The prover and verifier then computes $(n_1, \ldots, n_N, r) = \mathsf{PRG}(s)$ (or $(n_1, \ldots, n_{N'}, r) = \mathsf{PRG}(s)$) and computes the primes from the integers as described above using $\mathsf{PGen}^{\mathsf{c}}$ (or PGen). The output $(p_1, \ldots, p_N)$ may not appear to the prover as random, since it has access to the seed s. We prove that if we define $P_j = \mathsf{PGen}^{\mathsf{c}}(\mathsf{PRG}(s))$ (or $P_j = \mathsf{PGen}(\mathsf{PRG}(s))$) and let $P_1, \ldots, P_{j-1} \in \mathbb{Z}_q^N$ be any linearly independent vectors, the probability that $P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})$ or $p_{j,i} = p_{j,l}$ for some $i \neq l$ is negligible for all $1 \leq j \leq N$. This is all we need in our application.

9.4 Security Analysis

Formally, the security properties of our protocol are captured by the following propositions. The zero-knowledge property is relatively straightforward, so most of our effort is spent on analyzing the soundness of the protocol.

Proposition 9.7 (Zero-Knowledge). *Protocol 9.1 is honest verifier statistical zero-knowledge.*

Proof. The simulator chooses $p_1, \ldots, p_N$ and c honestly. Then it chooses random elements $f_1, \ldots, f_7, f_{1/w}, f_{x/w}, f_w, f_x, f_{r'} \in \mathbb{Z}_q$, $(e_i, e_i')_{i=1}^N \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]^N$, $(d_i)_{i=1}^N \in [0, 2^{\kappa_p+\kappa_c+\kappa_r} - 1]^N$, $e \in [0, 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} - 1]$, and $e' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r+\kappa_c+\log_2 N} - 1]$. It also chooses $b_1, \ldots, b_5 \in G_q$ and $\mathbf{b}_i, \mathbf{b}_i' \in \mathrm{QR}_{\mathbf{N}}$ randomly. Finally, the simulator defines, $(\beta_1, \ldots, \beta_9)$ by Equations (9.13)–(9.16),

$(\alpha_1, \alpha_2, \alpha_3)$ by Equations (9.17), $\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}'_i$ by Equations (9.18), $\boldsymbol{\gamma}$ by Equation (9.19), and $\boldsymbol{\gamma}'$ by Equation (9.20) respectively. It is easy to see that the distribution of the elements is statistically close to the distribution of the corresponding elements in the protocol. □

The protocol could be modified by adding a first step, where the verifier chooses $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(g_1, \ldots, g_N)$. This would give a computationally sound proof of knowledge, but in our application we wish to choose these parameters jointly and only once, and then let the mix-servers execute the proof with these parameters as common inputs. Thus, there may be a negligible portion of the parameters on which the prover can convince the verifier of false statements. Because of this we cannot hope to prove that the protocol is a proof of knowledge in the formal sense. Damgård and Fujisaki [55] introduce the notion of a computationally convincing proof of knowledge to deal with situations like these. In Section 2.8 we introduce the definition of computationally convincing proofs of knowledge that we use here.

Proposition 9.8. *Protocol 9.1 is a computationally convincing proof of knowledge for the relation R_{DP} with regards to the distribution of $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(g_1, \ldots, g_N)$.*

Proposition 9.9. *Protocol 9.1 has overwhelming completeness, also if* PGen *is used for prime generation. If* $\mathsf{PGen}^{\mathsf{c}}$ *is used instead, then this holds in Cramér's model of the primes.*

Proof. The honest verifier accepts a proof of an honest prover if prime vector generation succeeds and there is no modular reduction in the computation of e_i, e'_i, d_i, e, e' in the proof. We know from Lemma 9.4 and Lemma 9.6 that prime vector generation fails with negligible probability. A modular reduction in the computation of a single value occurs with probability $2^{-\kappa_r}$. The claim now follows from the union bound. □

9.4.1 Proof of Proposition 9.8

Recall the setting of computationally convincing proofs of knowledge from Section 2.8.1. We consider a malicious prover P^* which is given $\boldsymbol{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $\overline{g} = (g, g_1, \ldots, g_N)$ as input and is run with internal randomness r_{p}. The prover outputs an instance $I_{P^*}(\boldsymbol{\Gamma}, \overline{g}, r_{\mathrm{p}})$, i.e., public keys $z, y \in G_q$ and two lists $L, L' \in G_q^{2N}$ and then interacts with the honest verifier on the common input consisting of $(\boldsymbol{\Gamma}, \overline{g}, z, y, L, L')$. Recall that $\mathrm{view}^V_{P^*}(\boldsymbol{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}})$ denotes the view of the honest verifier when it executes with the malicious prover P^* and that the predicate Acc_V takes a view T as input and outputs 1 if the transcript is accepting and 0 otherwise. Let $L_{R_{\mathrm{DP}}}$ be the language corresponding to the decryption-permutation relation R_{DP}.

Although our proof of a shuffle is complex the analysis is similar in structure to most analyses of proofs of knowledge. First we identify conditions on a set of related transcripts that allow us to extract the knowledge held by the prover. This

corresponds to finding a "fork" in standard Schnorr-like proofs of knowledge of a logarithm. Then we describe how such transcripts can be generated by interacting with a prover.

Notation

We need to consider many transcripts and be able to distinguish between these. We denote the jth view of the honest verifier in a list by $T_j = (I_j, W_j, P_j, C_j, c_j, R_j)$ where I_j denotes the common input, W_j denotes the list of commitments w_i, and

$$\begin{aligned} P_j &= (p_{j,1}, \ldots, p_{j,N}) \\ C_j &= ((b_{j,i})_{i=1}^{5}, (\beta_{j,i})_{i=1}^{9}, (\alpha_{j,1}, \alpha_{j,2}, \alpha_{j,3}), (\mathbf{b}_{j,i}, \mathbf{b}'_{j,i})_{i=1}^{N}, (\boldsymbol{\gamma}_{j,i}, \boldsymbol{\gamma}'_{j,i})_{i=1}^{N}, (\boldsymbol{\gamma}_j, \boldsymbol{\gamma}'_j)) \\ R_j &= ((f_{j,i})_{i=1}^{7}, f_{j,1/w}, f_{j,x/w}, f_{j,w}, f_{j,x}, f_{j,r'}), (e_{j,i}, e'_{j,i})_{i=1}^{N}, (d_{j,i})_{i=1}^{N}, (e_j, e'_j)) \ . \end{aligned}$$

Since I_j and W_j are fixed for all j in most of our analysis we do not introduce any notation indexed on j for their parts. We think of (P_j, C_j, c_j, R_j) as "the primes", "the commitments", "the challenge" and "the reply" in the jth transcript.

Extraction From Suitable Transcripts

Recall that for standard Schnorr-like proofs of knowledge it suffices to find a "fork", i.e., two accepting transcripts with identical commitments from the prover, but distinct challenges from the verifier, to extract the knowledge held by the prover.

Our protocol has a similar, but more complicated property. The lemma below shows that given $2N$ transcripts of a special form, we can either extract a witness of the decryption-permutation relation, or one of a small number of special cases occur.

In a later section we show that if one of the special cases occur with non-negligible probability we can break a standard complexity assumption, so the reader should think of the lemma as saying that we can extract a witness for the decryption-permutation relation.

Lemma 9.10. *Let $T_1, \ldots, T_{2N}$ be accepting transcripts such that*

1. $(I_1, W_1) = (I_2, W_2) = \ldots = (I_N, W_N)$ *and* $I_1 = (g, z, y, L, L')$,
2. $(P_j, C_j) = (P_{j+N}, C_{j+N})$ *and* $c_j \neq c_{j+N}$, *and*
3. $\mathrm{Span}(P_1, \ldots, P_N) = \mathbb{Z}_q^N$ *and* $p_{j,i} \neq p_{j,l}$ *for* $i \neq l$.

Then we can find elements of one of the following types in polynomial time

1. Main Conclusion. *Elements $w, x \in \mathbb{Z}_q$ and a permutation $\pi \in \Sigma_N$ s.t.*

$$((g, z, y, L, L'), (w, x)) \in R_{\mathrm{DP}} \ .$$

2. *An element* $\mathbf{b} \in \mathrm{QR}_{\mathbf{N}}$*, and integers* $\eta_0 \neq 0$*, and* η_1*,* η_2*, not both zero such that one of the following holds.*
   a) *The integer* η_0 *does not divide both* η_1 *and* η_2*, and* $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1}\mathbf{g}^{\eta_2}$*.*
   b) *The integer* η_0 *does not divide* η_1*, and* $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1} = 1$*.*
3. *Integers* η_1*,* η_2*, not both zero, such that* $\mathbf{h}^{\eta_1}\mathbf{g}^{\eta_2} = 1$*.*
4. *Elements* ρ'_i *and* $\rho'_{i,j}$ *in* $\mathbb{Z}_q$ *such that* $w_i = g^{\rho'_i}\prod_{j=1}^{N} g_j^{\rho'_{i,j}}$*, and such that* $(\rho'_{ij})_{i,j=1}^{N}$ *is not a permutation matrix.*
5. *Elements* $\eta_0, \ldots, \eta_N \in \mathbb{Z}_q$*, not all zero, such that* $g^{\eta_0}\prod_{i=1}^{N} g_i^{\eta_i} = 1$*.*

Proof. The proof is quite complex and is divided into a number of cases, some of which are subdivided into sub-cases. The main track of the proof leads to the main conclusion. To aid the reader we adopt the convention that whenever the proof divides into two sub-cases it is always the first sub-case that leads to the main conclusion.

We first analyze the integer commitments of the protocol, and only then proceed with the analysis of the components in G_q. Our integer commitments are not standard, since the bases $\mathbf{b}_{j,i}$ are chosen by the adversary during the protocol, and we compose commitments in various ways. This said, we do use ideas from Fujisaki and Okamoto [67] and Damgård and Fujisaki [55].

The Prover Can Open $\mathbf{b}_{j,i}$ and $\mathbf{b}'_{j,i}$ To Some ρ_i. We consider Equations (9.18) iteratively for $i = 1, \ldots, N$. For a given i, the equations imply that

$$\begin{aligned} \mathbf{b}_{j,i}^{c_{j+N}-c_j} &= \mathbf{h}^{e_{j+N,i}-e_{j,i}}\mathbf{b}_{j,i-1}^{d_{j+N,i}-d_{j,i}} \ , \text{ and} \\ (\mathbf{b}'_{j,i})^{c_{j+N}-c_j} &= \mathbf{h}^{e'_{j+N,i}-e'_{j,i}}\mathbf{g}^{d_{j+N,i}-d_{j,i}} \ . \end{aligned}$$

There are several cases to consider.

1. If $c_{j+N} - c_j$ divides $e_{j+N,i} - e_{j,i}$, $d_{j+N,i} - d_{j,i}$ and $e'_{j+N,i} - e'_{j,i}$, we set $\tau_i = (e_{j+N,i} - e_{j,i})/(c_{j+N} - c_j)$, $\tau'_i = (e'_{j+N,i} - e'_{j,i})/(c_{j+N} - c_j)$ and

$$\rho_i = (d_{j+N,i} - d_{j,i})/(c_{j+N} - c_j) \ , \tag{9.21}$$

   and conclude that

$$\mathbf{b}_{j,i} = \mathbf{h}^{\tau_i}\mathbf{b}_{j,i-1}^{\rho_i} \ , \text{ and } \quad \mathbf{b}'_{j,i} = \mathbf{h}^{\tau'_i}\mathbf{g}^{\rho_i} \ . \tag{9.22}$$

   We apply the equalities iteratively and get

$$\mathbf{b}_{j,N} = \mathbf{h}^{\tau}\mathbf{g}^{\prod_{i=1}^{N}\rho_i}$$

   where $\tau = \tau_N + \rho_N(\tau_{N-1} + \rho_{N-1}(\tau_{N-2} + \rho_{N-2}(\tau_{N-3} + \rho_{N-3}(\ldots))))$.

2. Suppose now that $c_{j+N} - c_j$ divides $e_{j+N,i} - e_{j,i}$, $e'_{j+N,i} - e'_{j,i}$ and $d_{j+N,i} - d_{j,i}$ for all $i < l$, but not for $i = l$. Then we can only conclude that

$$\mathbf{b}_{j,l-1} = \mathbf{h}^{\tau_*}\mathbf{g}^{\prod_{i=1}^{l-1}\rho_i}$$

where $\tau_* = \tau_{l-1} + \rho_{l-1}(\tau_{l-2} + \rho_{l-2}(\tau_{l-3} + \rho_{l-3}(\tau_{l-4} + \rho_{l-4}(\ldots))))$. This implies that

$$\mathbf{b}_{j,l}^{c_{j+N}-c_j} = \mathbf{h}^{\tau_*(d_{j+N,l}-d_{j,l})+(e_{j+N,l}-e_{j,l})}\mathbf{g}^{(\prod_{i=1}^{l-1}\rho_i)(d_{j+N,i}-d_{j,i})} \quad .$$

Remark 9.11. To readers familiar with the proof in Damgård and Fujisaki [55], we point out that here it seems impossible to conclude that one of the exponents on the left is not divisible by the exponent on the right, which is necessary to reach a contradiction to the strong RSA-assumption. This is different from [55], where no additional factors corresponding to τ_* and $\prod_{i=1}^{l-1}\rho_i$ are present. The sole purpose of the $\mathbf{b}'_{j,i}$ commitments is to handle this problem. Thus, if the problem could be solved otherwise this would simplify the protocol and improve its efficiency somewhat.

There are two cases to consider.

a) If $c_{j+N} - c_j$ does not divide $e'_{j+N,l} - e'_{j,l}$ and $d_{j+N,l} - d_{j,l}$, then set $\mathbf{b} = \mathbf{b}'_{j,l}$, $\eta_0 = c_{j+N} - c_j$, $\eta_1 = e'_{j+N,l} - e'_{j,l}$, and $\eta_2 = d_{j+N,l} - d_{j,l}$. This implies that $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1}\mathbf{g}^{\eta_2}$ and η_0 does not divide both η_1 and η_2, i.e. Conclusion 2a of the lemma is satisfied.

b) If $c_{j+N} - c_j$ divides $e'_{j+N,i} - e'_{j,i}$ and $d_{j+N,l} - d_{j,l}$, then set

$$\mathbf{b} = \mathbf{b}_{j,l}\left(\mathbf{h}^{\tau_*}\mathbf{g}^{\prod_{i=1}^{l-1}\rho_i}\right)^{-\frac{d_{j+N,l}-d_{j,l}}{c_{j+N}-c_j}} \quad ,$$

$\eta_0 = c_{j+N} - c_j$, and $\eta_1 = e_{j+N,l} - e_{j,l}$. By assumption η_0 does not divide η_1. This implies that $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1}$ and η_0 does not divide η_1, i.e. Conclusion 2b of the lemma is satisfied.

There is no need to consider Case 2 further, but we must show that Case 1 leads to one of the conclusions in the lemma.

Each ρ_i Equals a Prime $p_{j,\pi_j(i)}$ Up to Sign For Some $\pi_j \in \Sigma_N$. Equation (9.19) implies that

$$(\mathbf{g}^{-\prod_{i=1}^{N} p_{ji}}\mathbf{b}_{j,N})^{c_{j+N}-c_j} = \mathbf{h}^{e_{j+N}-e_j} \quad .$$

There are two cases to consider.

1. If $c_{j+N} - c_j$ divides $e_{j+N} - e_j$ we set $\tau_* = (e_{j+N} - e_j)/(c_{j+N} - c_j)$ and conclude that

$$\mathbf{b}_{j,N} = \mathbf{h}^{\tau_*}\mathbf{g}^{\prod_{i=1}^{N} p_{ji}} \quad .$$

There are two sub-cases to consider.

a) If $(\tau_*, \prod_{i=1}^N p_{ji}) = (\tau, \prod_{i=1}^N \rho_i)$, then it follows from unique factorization in $\mathbb{Z}$ that $\rho_i = \prod_{l\in\Gamma_i} p_{j,l}$ for some subset $\Gamma_i \subset \{1,\ldots,N\}$. We also have $c_j\rho_i \in [-2^{\kappa_p+\kappa_c+\kappa_r}+1, 2^{\kappa_p+\kappa_c+\kappa_r}-1]$ by definition. Since $c_j \in [2^{\kappa_c-1}, 2^{\kappa_c}-1]$, this is only possible if $\rho_i \in [-2^{\kappa_p+\kappa_r}+1, 2^{\kappa_p+\kappa_r}-1]$. We know that $p_{j,l} \in [2^{\kappa_p-1}, 2^{\kappa_p}-1]$ and $\kappa_r < \kappa_p - 2$ so no product of more than one prime can be contained in $[-2^{\kappa_p+\kappa_r}+1, 2^{\kappa_p+\kappa_r}-1]$, i.e. $|\Gamma_i| \leq 1$. This implies that each ρ_i has at least one factor. We conclude that $\rho_i = \pm p_{j,\pi_j(i)}$ for some permutation $\pi_j \in \Sigma_N$.

b) If $(\tau_*, \prod_{i=1}^N p_{ji}) \neq (\tau, \prod_{i=1}^N \rho_i)$, we have

$$\mathbf{h}^{\tau-\tau_*}\mathbf{g}^{\prod_{i=1}^N \rho_i - \prod_{i=1}^N p_{ji}} = 1 \ .$$

If we set $\eta_1 = \tau - \tau_*$, and $\eta_2 = \prod_{i=1}^N \rho_i - \prod_{i=1}^N p_{ji}$, these integers satisfy Conclusion 3 of the lemma.

2. If $c_{j+N} - c_j$ does not divide $e_{j+N} - e_j$, then set $\mathbf{b} = \mathbf{g}^{-\prod_{i=1}^N p_{ji}}\mathbf{b}_{j,N}$, $\eta_0 = c_{j+N} - c_j$, and $\eta_1 = e_{j+N} - e_j$. Then $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1}$ and η_0 does not divide η_1, i.e., Conclusion 2a of the lemma is satisfied.

The only case we must consider further is Case 1a.

All ρ_i Are Positive. Equation (9.22) implies that

$$\prod_{i=1}^N \mathbf{b}'_{j,i} = \mathbf{h}^{\sum_{i=1}^N \tau'_i}\mathbf{g}^{\sum_{i=1}^N \rho_i} \ . \tag{9.23}$$

From Equation (9.20) we conclude that

$$\left(\mathbf{g}^{-\sum_{i=1}^N p_{j,i}} \prod_{i=1}^N \mathbf{b}'_{j,i}\right)^{c_{j+N}-c_j} = \mathbf{h}^{e'_{j+N}-e'_j} \ .$$

There are two cases to consider.

1. If $c_{j+N} - c_j$ divides $e'_{j+N} - e'_j$, we define $\tau' = (e'_{j+N} - e'_j)/(c_{j+N} - c_j)$ and conclude that

$$\prod_{i=1}^N \mathbf{b}'_{j,i} = \mathbf{h}^{\tau'}\mathbf{g}^{\sum_{i=1}^N p_{j,i}} \ .$$

There are two sub-cases to consider.

a) If $(\tau', \sum_{i=1}^N p_{j,i}) = (\sum_{i=1}^N \tau_i, \sum_{i=1}^N \rho_i)$, we conclude that ρ_i is positive. To see this, note that $p_{j,i} > 0$ and $\rho_i = \pm p_{j,\pi_j(i)} \neq 0$. Thus, if any ρ_i is negative we have $\sum_{i=1}^N p_{j,i} > \sum_{i=1}^N \rho_i$.

b) If $(\tau', \sum_{i=1}^{N} p_{j,i}) \neq (\sum_{i=1}^{N} \tau_i, \sum_{i=1}^{N} \rho_i)$, we have

$$\mathbf{h}^{\tau' - \sum_{i=1}^{N} \tau_i} \mathbf{g}^{\sum_{i=1}^{N} p_{j,i} - \sum_{i=1}^{N} \rho_i} = 1 \ .$$

If we set $\eta_1 = \tau' - \sum_{i=1}^{N} \tau_i$ and $\eta_2 = \sum_{i=1}^{N} p_{j,i} - \sum_{i=1}^{N} \rho_i$, these integers satisfy Conclusion 3 of the lemma.

2. If $c_{j+N} - c_j$ does not divide $e'_{j+N} - e'_j$, then set $\mathbf{b} = \mathbf{g}^{-\sum_{i=1}^{N} p_{j,i}} \prod_{i=1}^{N} \mathbf{b}'_{j,i}$, $\eta_0 = c_{j+N} - c_j$, and $\eta_1 = e'_{j+N} - e'_j$. This implies that $\mathbf{b}^{\eta_0} = \mathbf{h}^{\eta_1}$ and η_0 does not divide η_1, i.e., Conclusion 2b of the lemma is satisfied.

We have now established that either we have $\rho_i = p_{j,\pi_j(i)}$ for some permutation π_j, or one of the Conclusions 2a, 2b, or 3 of the lemma holds. We stress that we do not necessarily have $\pi_j = \pi_{j'}$ for $j \neq j'$.

From this point on we compute in G_q, and q is prime so the exponents live in the finite field $\mathbb{Z}_q$, i.e., every non-zero element can be inverted.

The Commitments w_i Are On The Expected Form. We now argue that the commitments w_i are a commitment of a permutation π. From Equation (9.17) we have

$$W_j^{c_{j+N} - c_j} = g^{f_{j+N,r'} - f_{j,r'}} \prod_{i=1}^{N} g_i^{d_{j+N,i} - d_{j,i}} \ .$$

We define $\gamma'_j = (f_{j+N,r'} - f_{j,r'})/(c_{j+N} - c_j)$ and conclude that

$$\prod_{i=1}^{N} w_i^{p_{j,i}} = g^{\gamma'_j} \prod_{i=1}^{N} g_i^{p_{j,\pi_j(i)}} \ . \tag{9.24}$$

Since the vectors $P_1, \ldots, P_N$ are linearly independent over $\mathbb{Z}_q^N$ by assumption, this means that for each i there exists coefficients $a_{i,1}, \ldots, a_{i,N} \in \mathbb{Z}_q$ such that $\sum_{j=1}^{N} a_{i,j} P_j = (\delta_{i,1}, \ldots, \delta_{i,N})$ with $\delta_{i,j} = 0$ for $j \neq i$ and $\delta_{i,i} = 1$. This implies that

$$w_i = \prod_{j=1}^{N} \left(\prod_{l=1}^{N} w_l^{p_{j,l}} \right)^{a_{i,j}} = \prod_{j=1}^{N} \left(g^{\gamma'_j} \prod_{l=1}^{N} g_l^{p_{j,\pi_j(l)}} \right)^{a_{i,j}} = g^{\rho'_i} \prod_{l=1}^{N} g_l^{\rho'_{i,l}} \ ,$$

where $\rho'_i = \sum_{j=1}^{N} \gamma'_j a_{i,j}$ and $\rho'_{i,l} = \sum_{j=1}^{N} p_{j,\pi_j(l)} a_{i,j}$.

We expect that

$$\begin{pmatrix} \rho'_{11} & \rho'_{12} & \cdots & \rho'_{1N} \\ \rho'_{21} & \rho'_{22} & \cdots & \rho'_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \rho'_{N1} & \rho'_{N2} & \cdots & \rho'_{NN} \end{pmatrix}$$

is a permutation matrix. If it is not, then Conclusion 4 is satisfied, so we assume it is a permutation matrix such that

$$w_i = g^{\rho'_i} g_{\pi^{-1}(i)} \ . \tag{9.25}$$

All Permutations Used Are Equal. We must show that all permutations π_j are equal. If $\pi_j \neq \pi$, we have from Equation (9.24) and (9.25) that

$$g^{\gamma_j'} \prod_{i=1}^{N} g_i^{p_{j,\pi_j(i)}} = \prod_{i=1}^{N} w_i^{p_{j,i}} = g^{\sum_{i=1}^{N} \rho_i' p_{j,i}} \prod_{i=1}^{N} g_i^{p_{j,\pi(i)}}$$

with some $p_{j,\pi_j(i)} - p_{j,\pi(i)} \neq 0$, since $p_{j,i} \neq p_{j,l}$ if $i \neq l$ by assumption. Then setting $\eta_0 = \gamma_j' - \sum_{i=1}^{N} \rho_i' p_{j,i}$ and $\eta_i = p_{j,\pi_j(i)} - p_{j,\pi(i)}$ implies $g^{\eta_0} \prod_{i=1}^{N} g_i^{\eta_i} = 1$ and Conclusion 5 of the lemma is satisfied.

The Common Input (g, z, y, L, L') Satisfies $((g, z, y, L, L'), (w, x)) \in R_{\mathrm{DP}}$. From the above we may assume that $\pi_j = \pi$ for all $j = 1, \ldots, N$, so we drop the subscript and simply write π from now on. Equations (9.13) and (9.14) imply that

$$\begin{aligned}
b_1^{c_{j+N}-c_j} &= g^{f_{j+N,1}-f_{j,1}} U_j^{f_{j+N,1/w}-f_{j,1/w}} \ , \\
b_2^{c_{j+N}-c_j} &= g^{f_{j+N,2}-f_{j,2}} U_j^{f_{j+N,x/w}-f_{j,x/w}} \ , \\
b_3^{c_{j+N}-c_j} &= g_1^{f_{j+N,3}-f_{j,3}} g^{f_{j+N,1/w}-f_{j,1/w}} \ , \text{ and} \\
b_4^{c_{j+N}-c_j} &= g_1^{f_{j+N,6}-f_{j,6}} g^{f_{j+N,x/w}-f_{j,x/w}} \ .
\end{aligned}$$

Define

$$\begin{aligned}
\omega' &= (f_{j+N,1/w} - f_{j,1/w})/(c_{j+N} - c_j) \ , \\
\xi' &= (f_{j+N,x/w} - f_{j,x/w})/(c_{j+N} - c_j) \ , \\
\kappa_1 &= (f_{j+N,1} - f_{j,1})/(c_{j+N} - c_j) \ , && (9.26) \\
\kappa_2 &= (f_{j+N,2} - f_{j,2})/(c_{j+N} - c_j) \ , && (9.27) \\
\kappa_3 &= (f_{j+N,3} - f_{j,3})/(c_{j+N} - c_j) \ , \text{ and} && (9.28) \\
\kappa_6 &= (f_{j+N,6} - f_{j,6})/(c_{j+N} - c_j) \ . && (9.29)
\end{aligned}$$

Then we have

$$\begin{aligned}
b_1 &= g^{\kappa_1} U_j^{\omega'} \ , && (9.30) \\
b_2 &= g^{\kappa_2} U_j^{\xi'} \ , && (9.31) \\
b_3 &= g_1^{\kappa_3} g^{\omega'} \ , \text{ and} && (9.32) \\
b_4 &= g_1^{\kappa_6} g^{\xi'} \ . && (9.33)
\end{aligned}$$

Equations (9.15) and (9.16) imply that

$$\begin{aligned}
b_4^{c_{j+N}-c_j} &= g_1^{f_{j+N,4}-f_{j,4}} b_3^{f_{j+N,x}-f_{j,x}} \ , && (9.34) \\
y^{c_{j+N}-c_j} &= g^{f_{j+N,x}-f_{j,x}} \ , && (9.35) \\
b_5^{c_{j+N}-c_j} &= g_1^{f_{j+N,5}-f_{j,5}} b_3^{f_{j+N,w}-f_{j,w}} \ , && (9.36) \\
z^{c_{j+N}-c_j} &= g^{f_{j+N,w}-f_{j,w}} \ , && (9.37) \\
(b_5/g)^{c_{j+N}-c_j} &= g_1^{f_{j+N,7}-f_{j,7}} \ . && (9.38)
\end{aligned}$$

Define

$$\begin{aligned}
\omega &= (f_{j+N,w} - f_{j,w})/(c_{j+N} - c_j) \ , \\
\xi &= (f_{j+N,x} - f_{j,x})/(c_{j+N} - c_j) \ , \\
\kappa_4 &= (f_{j+N,4} - f_{j,4})/(c_{j+N} - c_j) \ , && (9.39) \\
\kappa_5 &= (f_{j+N,5} - f_{j,5})/(c_{j+N} - c_j) \ , \text{ and} && (9.40) \\
\kappa_7 &= (f_{j+N,7} - f_{j,7})/(c_{j+N} - c_j) \ . && (9.41)
\end{aligned}$$

Then we have $g^x = y = g^\xi$ and $g^w = z = g^\omega$, so $\xi = x$ and $\omega = w$. This means that we have

$$\begin{aligned}
b_4 &= g_1^{\kappa_4} b_3^x && (9.42) \\
b_5 &= g_1^{\kappa_5} b_3^w && (9.43) \\
b_5 &= g_1^{\kappa_7} g && (9.44)
\end{aligned}$$

If we combine Equation (9.32) and Equation (9.43) we get

$$b_5 = g_1^{\kappa_5} g_1^{w\kappa_3} g^{w\omega'} \ . \tag{9.45}$$

There are two cases

1. If $\omega' = 1/w$ we conclude that

$$\begin{aligned}
b_3 &= g_1^{\kappa_3} g^{1/w} \ , \text{ and} && (9.46) \\
b_4 &= g_1^{\kappa_4 + x/\kappa_3} g^{x/w} \ . && (9.47)
\end{aligned}$$

   If $\xi' \neq x/w$, we set $\eta_0 = \kappa_4 + x\kappa_3 - \kappa_6$ and $\eta_1 = x/w - \xi'$. This gives $g^{\eta_0} g_1^{\eta_1} = 1$, i.e., Conclusion 5 is satisfied. Thus, we may assume that $\xi' = x/w$. Combined with Equations (9.30) and (9.31) this gives

$$\begin{aligned}
b_1 &= g^{\kappa_1} U_j^{1/w} \ , \text{ and} && (9.48) \\
b_2 &= g^{\kappa_2} U_j^{x/w} \ . && (9.49)
\end{aligned}$$

2. If $\omega' \neq 1/w$ we define $\eta_0 = w\omega' - 1$ and $\eta_1 = \kappa_5 + w\kappa_3 - \kappa_7$. From Equations (9.45) and (9.44) we conclude that $g^{\eta_0} g_1^{\eta_1} = 1$, i.e., Conclusion 5 is satisfied.

From Equation (9.17) we have

$$\begin{aligned}
b_1^{c_{j+N} - c_j} &= g^{f_{j+N,1} - f_{j,1}} \prod_{i=1}^{N} (u_i')^{d_{j+N,i} - d_{j,i}} \ , \text{ and} \\
(V_j/b_2)^{c_{j+N} - c_j} &= g^{-(f_{j+N,2} - f_{j,2})} \prod_{i=1}^{N} (v_i')^{d_{j+N,i} - d_{j,i}} \ .
\end{aligned}$$

Our definitions of κ_1 and κ_2 in Equations (9.26) and (9.27), and the definition of ρ_i, which equals $p_{j,\pi(i)}$ in Equation (9.21) imply that

$$\begin{aligned} b_1 &= g^{\kappa_1} \prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \ , \text{ and} \\ V_j/b_2 &= g^{-\kappa_2} \prod_{i=1}^{N} (v'_i)^{p_{j,\pi(i)}} \ . \end{aligned}$$

If we combine Equations (9.48) and (9.49) with the two equations above we have

$$\prod_{i=1}^{N} (u_i^{1/w})^{p_{ji}} = U_j^{1/w} = \prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \ , \text{ and} \tag{9.50}$$

$$\prod_{i=1}^{N} (v_i u_i^{-x/w})^{p_{ji}} = V_j U_j^{-x/w} = \prod_{i=1}^{N} (v'_i)^{p_{j,\pi(i)}} \ , \tag{9.51}$$

for $j = 1, \ldots, N$. We apply the coefficients $a_{l,1}, \ldots, a_{l,N} \in \mathbb{Z}_q$ introduced above to the Equations (9.50) and (9.51) and conclude that

$$\begin{aligned} u_l^{1/w} &= \prod_{j=1}^{N} \left(\prod_{i=1}^{N} (u_i^{1/w})^{p_{ji}} \right)^{a_{l,j}} = \prod_{j=1}^{N} \left(\prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \right)^{a_{l,j}} = u'_{\pi^{-1}(l)} \ , \text{ and} \\ v_l u_l^{-x/w} &= \prod_{j=1}^{N} \left(\prod_{i=1}^{N} (v_i u_i^{-x/w})^{p_{ji}} \right)^{a_{l,j}} = \prod_{j=1}^{N} \left(\prod_{i=1}^{N} (v'_i)^{p_{j,\pi(i)}} \right)^{a_{l,j}} = v'_{\pi^{-1}(l)} \ . \end{aligned}$$

This concludes the proof. □

Random Prime Vectors are Linearly Independent

We show that given linearly independent vectors $P_1, \ldots, P_{j-1} \in \mathbb{Z}_q^N$, a random prime vector P_j is contained in the subspace spanned by $P_1, \ldots, P_{j-1}$ with negligible probability as long as $j \leq N$ and $p_{j,i}$ equals any fixed prime with negligible probability.

Lemma 9.12. *Let $P_1, \ldots, P_{j-1} \in \mathbb{Z}_q^N$ be linearly independent vectors. If $P_j = (p_{j,1}, \ldots, p_{j,N})$ is a list of independently and identically distributed primes such that $\Pr[p_{j,i} = p] \leq \epsilon$ for every prime p, we have*

$$\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})] \leq \epsilon^{N-j+1} \ .$$

Proof. The vectors $P_1, \ldots, P_{j-1}$ form a matrix

$$P = \left(\begin{array}{cccc|ccc} p_{1,1} & p_{1,2} & \cdots & p_{1,j-1} & p_{1,j} & \cdots & p_{1,N} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j-1} & p_{2,j} & \cdots & p_{2,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ p_{j-1,1} & p_{j-1,2} & \cdots & p_{j-1,j-1} & p_{j-1,j} & \cdots & p_{j-1,N} \end{array} \right) \ .$$

Suppose that we replace $P_1, \ldots, P_{j-1}$ with a set of rows $P'_1, \ldots, P'_{j-1}$ formed by elementary row operations. Then clearly $\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})] = \Pr[P_j \in \mathrm{Span}(P'_1, \ldots, P'_{j-1})]$, since $\mathrm{Span}(P'_1, \ldots, P'_{j-1}) = \mathrm{Span}(P_1, \ldots, P_{j-1})$. Given a permutation π of N elements, we denote by P_i^π the vector with permuted components defined as $(p_{i,\pi(1)}, p_{i,\pi(2)}, \ldots, p_{i,\pi(N)})$. Since the primes $p_{j,1}, \ldots, p_{j,N}$ are identically and independently distributed we conclude that

$$\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})] = \Pr[P_j \in \mathrm{Span}(P_1^\pi, \ldots, P_{j-1}^\pi)]$$

for every permutation π of N elements.

Thus, we can by elementary row operations and by permuting the columns in the matrix form a new matrix P' from P on the form

$$P' = \begin{pmatrix} P'_1 \\ P'_2 \\ \vdots \\ P'_{j-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & p'_{1,j} & \cdots & p'_{1,N} \\ 0 & 1 & \cdots & 0 & p'_{2,j} & \cdots & p'_{2,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p'_{j-1,j} & \cdots & p'_{j-1,N} \end{pmatrix},$$

with $\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})] = \Pr[P_j \in \mathrm{Span}(P'_1, \ldots, P'_{j-1})]$. We have $P_j \in \mathrm{Span}(P'_1, \ldots, P'_{j-1})$ if and only if

$$\sum_{i=1}^{j-1} p_{j,i} p'_{i,l} = p_{j,l}$$

for $l = j, \ldots, N$. From independence we conclude that the probability of this event is at most ϵ^{N-j+1}, which concludes the proof. □

Corollary 9.13. *Let $P_1, \ldots, P_{j-1} \in \mathbb{Z}_q^N$ be linearly independent vectors and let $s \in [0, 2^\kappa - 1]$ be a randomly chosen seed.*

If $P_j = \mathsf{PGen}^{\mathsf{c}}(\mathsf{PRG}(s))$, $\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})]$ is negligible in Cramér's model. If $P_j = \mathsf{PGen}(\mathsf{PRG}(s))$, then $\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})]$ is negligible.

Proof. We prove the first claim. The second follows by a similar argument. Suppose that $\Pr[P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})] \geq 1/\kappa^c$ for some linearly independent vectors $P_1, \ldots, P_{j-1} \in \mathbb{Z}_q^N$ and κ in some infinite set $\mathcal{N}$ of security parameters. Consider the distinguisher A that is given $(n_1, \ldots, n_N, r)$ generated either by choosing $n_i \in [0, 2^{\kappa_p} - 1]$ randomly, or by choosing $s \in [0, 2^\kappa - 1]$ randomly and computing $(n_1, \ldots, n_N, r) = \mathsf{PRG}(s)$. The distinguisher simply checks if $P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})$ and outputs 1 or 0 depending on the result. It follows from Lemma 9.12 that A can distinguish uniformly and independently generated integers from pseudo-randomly generated integers. This contradicts the fact that PRG is a pseudo-random generator as in Definition 2.9. □

A Non-Permutation Matrix Does Not Behave Like One

Lemma 9.14. *Let $B = (b_{ij})$ be an $N \times N$-matrix over $\mathbb{Z}_q$ and let $X = (X_1, \ldots, X_N)$ consist of independently and identically distributed random variables such that $\Pr[X_i = x_i] \leq \epsilon$ for all $x_i \in \mathbb{Z}_q$. Then if B is not a permutation matrix*

$$\Pr[\exists \pi \in \Sigma_N \ \ s.t. \ \ BX = X^\pi] \leq N\epsilon \ .$$

Proof. The set of permutation matrices are characterized as the matrices such that in each row and in each column exactly one element equal one and the rest are zero. We say that a column or row is bad if it does not have the above property. Thus, if B is not a permutation matrix, it must have a bad row or a bad column.

Suppose first that there is a bad row. Without loss we assume that the first row is bad. We have

$$\Pr[\exists \pi \in \Sigma_N \ \text{s.t.} \ BX = X^\pi] \leq \Pr\left[\exists 1 \leq j \leq N \ \text{s.t.} \ \sum_{l=1}^{N} b_{1l} X_l - X_j = 0\right]$$
$$\leq \sum_{j=1}^{N} \Pr\left[\sum_{l=1}^{N} b_{1l} X_l - X_j = 0\right] \leq N\epsilon \ .$$

where we use the union bound, and the fact that $\Pr[\sum_{l=1}^{N} b_{il} X_l - X_j = 0] \leq \epsilon$, since the expression $\sum_{l=1}^{N} b_{il} X_l - X_j$ is not identically zero for any j.

Suppose now that no row is bad. Then there are exactly N ones in total in the matrix B, so if some column is bad there must be an all zero column as well. Suppose that the jth column is an all zero column, i.e., $b_{i,j} = 0$ for $i = 1, \ldots, N$. This implies that $\Pr[\sum_{l=1}^{N} b_{il} X_l = X_j] \leq \epsilon$ for all i, since X_j is independent of all expressions $\sum_{l=1}^{N} b_{il} X_l$ for $i = 1, \ldots, N$. Thus, it follows that $\Pr[\exists \pi \in \Sigma_N \ \text{s.t.} \ BX = X^\pi] \leq N\epsilon$ also in this case. □

Corollary 9.15. *Let $B = (b_{ij})$ be an $N \times N$-matrix over $\mathbb{Z}_q$, let $s \in [0, 2^\kappa - 1]$ be a randomly chosen seed, and let B be a non-permutation matrix.*

If $P = \mathsf{PGen}^{\mathsf{c}}(\mathsf{PRG}(s))$, then $\Pr[\exists \pi \in \Sigma_N \ s.t. \ BP = P^\pi]$ is negligible in Cramér's model. If $P = \mathsf{PGen}(\mathsf{PRG}(s))$, then $\Pr[\exists \pi \in \Sigma_N \ s.t. \ BP = P^\pi]$ is negligible.

Proof. The proof is almost identical to the proof of Corollary 9.13 and omitted. □

Generation of Suitable Transcripts

We describe how the adversary can extract transcripts that satisfy Lemma 9.10. We consider the transcript from the interaction of P^* with an honest verifier as a list of functions $\text{view}_{P^*}^V = (I_{P^*}, W_{P^*}, P, C_{P^*}, c, R_{P^*})$, where I_{P^*} is the part of the common input constructed by P^*, W_{P^*} is the first commitments computed by P^*, P is the list of N primes chosen by the verifier, C_{P^*} is the second set of

commitments of the prover, c is the challenge chosen by the verifier, and R_{P^*} is the reply of the prover. Denote by r_p and r_v the random input of the prover and verifier respectively. Then $\mathrm{view}^V_{P^*}$ is clearly a function of $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$, $\overline{g} = (g, g_1, \ldots, g_N)$, r_p, and r_v, but not all parts depend on all variables. If we divide r_v in two parts r_v' and r_v'', where the former is used to construct the list of primes P, and the latter is used to construct the challenge c the dependencies are given by

$$\begin{aligned}\mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v}) \quad = \quad & \big(I_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}), W_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}), P(r_\mathrm{v}'), C_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v}'), \\ & c(r_\mathrm{v}''), R_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v}', r_\mathrm{v}'')\big) \ .\end{aligned}$$

Recall our notation from Section 2.8

$$\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) = \Pr_{r_\mathrm{v}}[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v})) = 1] \ .$$

This is the probability that the view of the honest verifier is accepting when interacting with P^* on a fixed special parameter $(\mathbf{\Gamma}, \overline{g})$ and fixed random string r_p of P^*. The probability is taken over the random choices of the honest verifier.

Denote by B the distribution defined by $\Pr_{b \leftarrow B}[b = 1] = \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})/32$. This distribution can be sampled efficiently without knowledge of $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})$ by choosing $r_\mathrm{v} \in \{0,1\}^*$ and $n \in \{1, \ldots, 32\}$ randomly and outputting 1 if we have $\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v})) = 1$ and $n = 1$, and 0 otherwise.

We define an algorithm FF_{P^*}, called the fork-finder, that given $(\mathbf{\Gamma}, \overline{g})$, r_p, and a list $(P_1, \ldots, P_{j-1})$ of vectors in $\mathbb{Z}_q^N$ outputs a pair of transcripts (T_j, T_{j+N}) such that $\mathrm{Acc}_V(T_j) = 1$, $\mathrm{Acc}_V(T_{j+N}) = 1$, $(I_j, W_j, P_j) = (I_{j+N}, W_{j+N}, P_{j+N})$, and $c_j \neq c_{j+N}$. Furthermore, we have $P_j \notin \mathrm{Span}(P_1, \ldots, P_{j-1})$ and $p_{j,i} \neq p_{j,l}$ for $i \neq l$. Note that this is a fork in the sense of the forking lemma.

Algorithm 9.16 (Fork Finder).

$\mathsf{FF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, (P_1, \ldots, P_{j-1}))$
`Loop`
  `Do {`
    $r_\mathrm{v}', r_\mathrm{v}'' \leftarrow \{0,1\}^*$
    $T_j \leftarrow \mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, r_\mathrm{v}', r_\mathrm{v}'')$
  `} While` $(\mathrm{Acc}_V(T_j) = 0$ or $P_j \in \mathrm{Span}(P_1, \ldots, P_{j-1})$
      or $\exists i, l : i \neq l$ and $p_{j,i} = p_{j,l})$
  `Do {`
    $r_\mathrm{v}''' \leftarrow \{0,1\}^*$, $b \leftarrow B$
    $T_{j+N} \leftarrow \mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{v}', r_\mathrm{v}''')$
  `} While` $((\mathrm{Acc}_V(T_{j+N}) = 0$ or $c_{j+N} = c_j)$ and $b \neq 1)$
  `If` $(\mathrm{Acc}_V(T_{j+N}) = 1$ and $c_{j+N} \neq c_j)$ `Then`
    `Return` (T_j, T_{j+N})
  `EndIf`
`EndLoop`

Lemma 9.17. *Consider an execution of* FF_{P^*} *on input* $(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, (P_1, \ldots, P_{j-1}))$. *If* $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$ *is non-negligible the expected number of times it invokes* P^* *is* $O(1/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}))$.

Proof. Consider a fixed iteration of the outer loop. By definition, the first condition in the first while-loop is satisfied with probability $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$. Corollary 9.13 and Corollary 9.3 imply that the second and third conditions are satisfied with overwhelming probability. The union bound then implies that all conditions are satisfied with probability at least $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/2$. Thus, the expected number of iterations of the loop and the number of invocations of P^* is bounded by $2/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$.

Consider now the second while-loop. Denote by I_{heavy} the set of ${r_{\mathrm{v}}}'$ such that

$$\Pr_{r_{\mathrm{v}}}[\mathrm{Acc}_V(\mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge P_j \notin \mathrm{Span}(P_1, \ldots, P_{j-1}) \\ \wedge (i \neq l \Rightarrow p_{j,i} \neq p_{j,l}) \mid {r_{\mathrm{v}}}' \in I_{\mathrm{heavy}}] \geq \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/4 \ .$$

An averaging argument implies that

$$\Pr_{r_{\mathrm{v}}}[{r_{\mathrm{v}}}' \in I_{\mathrm{heavy}} \mid \mathrm{Acc}_V(\mathrm{view}^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge P_j \notin \mathrm{Span}(P_1, \ldots, P_{j-1}) \\ \wedge (i \neq l \Rightarrow p_{j,i} \neq p_{j,l})] \geq 1/2 \ .$$

Thus, with probability $1/2$ the value of ${r_{\mathrm{v}}}'$ fixed in the first loop belongs to I_{heavy}.

By definition of the distribution B we have $\Pr[b = 1] = \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/32$. Thus, if the first condition of the while-loop is removed the expected number of iterations is $32/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$, and the probability that more than $16/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$ iterations are needed is at least $(1 - \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/32)^{16/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})}$. Set $\delta = \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/32$. Then this can be bounded by $(1 - \delta)^{1/2\delta} \geq (e^{-\delta - \delta^2/2})^{1/2\delta} \geq e^{-3/4} \geq 2/5$

The number of iterations in the unmodified while-loop is obviously also bounded by $32/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$. Thus, the expected number of invocations of P^* in the second while-loop is bounded by $64/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$, since one invocation is needed in the construction of the transcript and one invocation in the sampling of B.

We must estimate the probability that the last if-statement is satisfied, conditioned on ${r_{\mathrm{v}}}' \in I_{\mathrm{heavy}}$. We estimate the probability that the second while-loop is terminated by the first condition. The probability that $c_{j+N} = c_j$ is $2^{\kappa_c - 1}$. Thus, the union bound implies that the first condition is satisfied, conditioned on ${r_{\mathrm{v}}}' \in I_{\mathrm{heavy}}$, with probability at least $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})/8$. If the second condition of the while-loop is removed the expected number of iterations would then be bounded by $8/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$ and the probability that more than $16/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$ iterations are necessary is at most $1/2$ by Markov's inequality. This implies that the probability that the second loop is terminated by the first condition is at least $\frac{1}{2}\frac{2}{5} = \frac{1}{5}$, conditioned on ${r_{\mathrm{v}}}' \in I_{\mathrm{heavy}}$.

We have argued that the expected number of invocations in a single iteration of the outer loop is $O(1/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}))$ and the probability that the return statement is reached in such an iteration is at least $\frac{1}{2}\frac{1}{5} = \frac{1}{10}$, and the claim follows. □

Algorithm 9.18 (Transcript Finder).

$\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})$
`For` $(j = 1, \ldots, N)$ `Do`
  $(T_j, T_{j+N}) \leftarrow \mathsf{FF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}, (P_1, \ldots, P_{j-1}))$
`EndLoop`
`Return` $(T_1, \ldots, T_{2N})$

Corollary 9.19. *Consider an execution of the algorithm* TF_{P^*} *on input* $(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})$. *If* $\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})$ *is non-negligible, the expected number of times it invokes* P^* *is at most* $O(N/\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}))$. *The output satisfies the hypothesis of Lemma 9.10.*

Proof. This follows immediately from Lemma 9.17. □

Concluding the Proof

Denote by $\mathcal{C}_\mathrm{MAIN}$, $\mathcal{C}_\mathrm{RSA}$, $\mathcal{C}_{\mathrm{RSA}'}$, $\mathcal{C}_\mathrm{REP}$, and $\mathcal{C}_{\mathrm{REP}'}$, the algorithms that given transcripts $T_1, \ldots, T_{2N}$ that satisfy the hypothesis of Lemma 9.10 try to extract values corresponding to Conclusion 1, 2, 3, 4, and 5 respectively. If this is not possible the algorithms output $\perp$.

We define the extractor $\mathcal{X}$ required by definition of a computationally convincing proof of knowledge as the algorithm that on input $(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})$ outputs the output of $\mathcal{C}_\mathrm{MAIN}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}))$. Recall the definition of a computationally convincing proof of knowledge from Section 2.8.1. The first requirement on a computationally convincing proof of knowledge now follows from Corollary 9.19 and the fact that $\mathcal{C}_\mathrm{MAIN}$ is a polynomial-time algorithm.

Suppose that the second requirement is not satisfied. Then there exists an adversary P^*, a constant c, and an infinite index set $\mathcal{N}$ such that

$$\begin{aligned} \Pr[\delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}] &\geq \kappa^{-c} \quad (9.52) \\ \Pr[(I_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}), \mathcal{X}^{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \notin R_\mathrm{DP} \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}] &\geq \kappa^{-c} \ . \end{aligned}$$

for $\kappa \in \mathcal{N}$. This implies that

$$\begin{aligned} \Pr[&\mathcal{C}_\mathrm{RSA}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \ \vee \ \mathcal{C}_{\mathrm{RSA}'}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \quad (9.53) \\ &\vee \ \mathcal{C}_\mathrm{REP}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \ \vee \ \mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \\ &\mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}] \geq \kappa^{-c} \ . \end{aligned}$$

We show that this leads to a contradiction. Denote by TF'_{P^*} the algorithm that simulates TF_{P^*} except that if it invokes P^* more than $2N\kappa^{2c}$ it halts and outputs $\perp$. Corollary 9.19 and Markov's inequality implies that

$$\Pr[\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) = \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}] \leq \frac{1}{2\kappa^c} \ . \quad (9.54)$$

Claim 1. Both $\Pr[\mathcal{C}_\mathrm{RSA}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}]$
and $\Pr[\mathcal{C}_{\mathrm{RSA}'}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_\mathrm{p}) \geq \kappa^{-c}]$ are negligible under the strong RSA-assumption.

Proof. We prove the first statement. The second statement follows in a similar way. Suppose that the claim is false. Then we define an algorithm RSA that breaks the strong RSA-assumption. It accepts $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ as input and chooses r_{p} and $\overline{g} = (g, g_1, \ldots, g_N)$ randomly. Then it outputs $\mathcal{C}_{\mathrm{RSA}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}))$.

Equation (9.54) and the union bound implies that

$$\Pr[\mathcal{C}_{\mathrm{RSA}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \ .$$

Combined with Equation (9.52) this gives $\Pr[\mathcal{C}_{\mathrm{RSA}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp] \geq \frac{1}{2\kappa^{2c}}$. This means that

$$\begin{aligned}\Pr[\mathsf{RSA}(\mathbf{N}, \mathbf{g}, \mathbf{h}) = (\mathbf{b}, \eta_0, \eta_1, \eta_2) \wedge \eta_0 \neq 0 \wedge (\eta_0 \nmid \eta_1 \vee \eta_0 \nmid \eta_2) \\ \wedge \mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1} \mathbf{h}^{\eta_2} \bmod \mathbf{N}] \geq \frac{1}{2\kappa^{2c}} \ .\end{aligned}$$

Lemma 3.12 implies that this breaks the strong RSA-assumption. □

Claim 2. $\Pr[\mathcal{C}_{\mathrm{REP}}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}]$ is negligible under the DL-assumption.

Proof. Suppose that the claim is false. Then we define an algorithm DLOG that breaks the DL-assumption. It accepts $\overline{g} = (g, g_1, \ldots, g_N)$ as input and chooses $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ and r_{p} randomly. Then it outputs $\mathcal{C}_{\mathrm{REP}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}))$.

Equation (9.54) and the union bound implies that

$$\Pr[\mathcal{C}_{\mathrm{REP}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \ .$$

Combined with Equation (9.52) this gives $\Pr[\mathcal{C}_{\mathrm{REP}}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp] \geq \frac{1}{2\kappa^{2c}}$, which means that

$$\Pr[\mathsf{DLOG}(g, g_1, \ldots, g_N) = (\eta_1, \ldots, \eta_N) \neq 0 \wedge g^{\eta_0} \prod_{j=1}^{N} g_j^{\eta_j} = 1] \geq \frac{1}{2\kappa^c} \ .$$

Lemma 3.5 implies that this contradicts the DL-assumption in G_q. □

We combine Equation (9.53) with Claim 1 and Claim 2 and apply the union bound. This gives

$$\Pr[\mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \ . \tag{9.55}$$

under the strong RSA-assumption and the DL-assumption. We construct an algorithm DLOG' that outputs a non-trivial representation of $1 \in G_q$ with notable probability. On input $\overline{g} = (g, g_1, \ldots, g_N)$ it chooses $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ and r_{p} randomly and computes $(\rho', (\rho'_{ij})) = \mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}))$. Denote by FF'_{P^*} the algorithm that simulates FF_{P^*}, except that it outputs $\perp$ if FF_{P^*} invokes P^* more

than $4\kappa^{2c}$ times. The algorithm DLOG' computes $(T_{2N+1}, T_{2N+2}) = \mathsf{FF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})$ and defines $\eta_0 = \sum_{i=1}^{N} \rho'_i p_{2N+1,i} - \gamma'_{2N+1}$ and $(\eta_j)_{j=1}^{N} \leftarrow (\sum_{i=1}^{N} \rho'_{i,j} p_{2N+1,i} - p_{2N+1,\pi_{2N+1}(j)})_{j=1}^{N}$. Finally, it returns $(\eta_j)_{j=0}^{N}$.

From Equation (9.55), Lemma 9.17, and Markov's inequality we have

$$\Pr[\mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \quad , \quad \text{and}$$
$$\Pr[\mathsf{FF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) = \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \leq \frac{1}{4\kappa^c} \quad .$$

The union bound implies that

$$\Pr[\mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp \wedge \mathsf{FF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \neq \perp \mid \delta^V_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \frac{1}{4\kappa^c}$$

Consider an execution of DLOG'. If the outputs satisfy $\mathcal{C}_{\mathrm{REP}'}(\mathsf{TF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}})) \neq \perp$ and $\mathsf{FF}'_{P^*}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \neq \perp$, then Equation (9.24) in the proof of Lemma 9.10 implies that

$$\begin{aligned} g^{\gamma'_{2N+1}} \prod_{i=1}^{N} g_i^{p_{2N+1,\pi_{2N+1}(i)}} &= \prod_{i=1}^{N} w_i^{p_{2N+1,i}} = \prod_{i=1}^{N} \left(g^{\rho'_i} \prod_{j=1}^{N} g_j^{\rho'_{i,j}} \right)^{p_{2N+1,i}} \\ &= g^{\sum_{i=1}^{N} \rho'_i p_{2N+1,i}} \prod_{j=1}^{N} g_j^{\sum_{i=1}^{N} \rho'_{i,j} p_{2N+1,i}} \quad . \end{aligned}$$

Corollary 9.15 and Corollary 9.3 imply that the probability that $\sum_{i=1}^{N} \rho'_{i,j} p_{2N+1,i} = p_{2N+1,\pi_{2N+1}(j)}$ for $j = 1, \ldots, N$ is negligible. Thus, we conclude that

$$\Pr\left[\mathsf{DLOG}'(g, g_1, \ldots, g_N) = (\eta_0, \ldots, \eta_N) \neq 0 \wedge \prod_{j=0}^{N} g_j^{\eta_j} = 1 \right] \geq \frac{1}{8\kappa^{2c}} \quad .$$

Lemma 3.5 implies that this contradicts the DL-assumption in G_q.

9.5 Complexity Analysis

Comparing the complexity of protocols is delicate, since any comparison must take place for equal security rather than for equal security parameters. The only rigorous method to do this is to perform an exact security analysis of each protocol and choose the security parameters accordingly. Various optimization and precomputing techniques are also applicable to different degrees in different protocols and in different applications. We do not perform a complete exact analysis, but we estimate the computational complexity of the protocol without and also with optimizations. We also take the liberty to focus on the solution based on $\mathsf{PGen}^{\mathsf{c}}$ in our estimates.

Denote by t_{EXP} the time it takes to compute an exponentiation in G_q, i.e., with an κ-bit exponent modulo a κ-bit integer. If the exponent instead has κ bits, we assume that computing the exponentiation takes time $\frac{\kappa}{\kappa} t_{\text{EXP}}$. We must also relate the time it takes to compute an exponentiation with a κ-bit exponent and modulus to t_{EXP}. We assume that this takes time $(\frac{\kappa}{\kappa})^3 t_{\text{EXP}}$. This is reasonable since exponentiation normally takes cubic time in the bit-size of the inputs for inputs of practical size. We use the same conventions for exponentiation in $\text{QR}_{\mathbf{N}}$ with κ replaced by $\kappa_{\mathbf{N}}$.

We use the same convention as Furukawa [68] when we estimate the effect of standard optimization techniques [102], i.e., simultaneous exponentiation reduces the cost by a factor of $1/3$, and fixed base exponentiation reduces the cost by a factor of $1/12$. Furukawa [68] estimates the complexity of previous shuffle-proofs, and claims that his protocol is the most efficient and that it requires the least amount of rounds for a shuffle that involves decryption, namely 5 rounds. It requires $8N$ and $6N$ exponentiations for the prover and verifier respectively. Using standard optimizations, such as fixed-base exponentiation and simultaneous exponentiation [102], this corresponds to less than $2N$ general exponentiations in G_q for each party.

Our protocol requires 5 rounds as well. We estimate the complexity of our proof of a shuffle without any optimizations and then with the same optimizations as Furukawa [68] uses.

First we consider the theoretical requirements. For simplicity we set $\kappa = \kappa_{\mathbf{N}}$. Recall that, using the number field sieve, discrete logarithms modulo an κ-bit prime can be solved in time $\exp(O(\kappa^{1/3} \log \kappa)) \leq \exp(O(\kappa^{2/5}))$. Thus, from a theoretical point of view we may assume that $\kappa_p = \kappa^{2/5}$ to balance the security parameters. We count all terms which do not contribute terms with complexity $o(\kappa)$. It can be seen that the cost for the prover and verifier is roughly $5N$ and $2N$ exponentiations respectively. With optimizations the corresponding estimates would be roughly $N/2$ and $N/5$.

To give the reader an idea of the practical complexity of our protocol we estimate the complexity for some common parameters. We set κ relatively large to ensure long term security, e.g. $\kappa = 2048$. A much smaller value of $\kappa_{\mathbf{N}}$ suffices, since it need only guarantee security during the execution of the protocol, e.g. $\kappa_{\mathbf{N}} = 1024$. A small challenge suffices since the protocol is run interactively, e.g. $\kappa_c = 160$. The size of the primes need only guarantee that prime vectors are linearly independent with high probability, e.g. $\kappa_p = 100$. Finally, a small value of κ_r suffices to ensure the statistical zero-knowledge of the protocol, e.g. $\kappa_r = 50$. With these parameters the complexity is less than $2.5N$ and $1.6N$ exponentiations in G_q for the prover and verifier. With optimizations as in [68] this corresponds to $0.5N$ and $0.8N$ general exponentiations in G_q. This indicates that our scheme is at least as efficient as previous protocols.

Our estimates for the practical complexity of the protocol follows from the Scheme program below. We have simply counted the number of exponentiations using the assumptions above. Each function `veriStep`i computes the complexity of verifier in the ith step of the protocol and correspondingly for the prover

Parameters					Non-Optimized		Optimized	
κ	$\kappa_{\mathbf{N}}$	κ_c	κ_p	κ_r	Prover	Verifier	Prover	Verifier
1024	1024	160	100	50	$7.8N$	$8.2N$	$1.4N$	$4.8N$
2048	1024	160	100	50	$2.4N$	$1.5N$	$0.4N$	$0.8N$
2048	2048	160	100	50	$6.4N$	$3.7N$	$0.9N$	$1.1N$
3072	2048	160	100	50	$2.9N$	$1.4N$	$0.4N$	$0.4N$
3072	3072	160	100	50	$5.9N$	$3.0N$	$0.7N$	$0.6N$

Table 9.1: The table gives estimates of the complexity, without and with optimization, of the prover and verifier in terms of general exponentiations in G_q for some common security parameters.

and `proStep`i. In Table 9.1 we give the estimated complexity for some common parameters.

9.6 Universal Verifiability and the Use of Random Oracles

Several authors propose turning their proofs of a shuffle into non-interactive zero-knowledge proofs in the random oracle model using the Fiat-Shamir heuristic. This allows any outsider to check, non-interactively, that a mix-server behaves correctly. If the verification involves no trusted parameters the resulting mix-net is called "universally verifiable".

The Fiat-Shamir heuristic can be applied to our protocol as well, but we do not see how the prover can generate the RSA-parameters $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ by itself and not know the factorization of $\mathbf{N}$ or a non-trivial root. Thus, a verifier must trust that the RSA-parameters are generated in a secure way, and the resulting mix-net is not really universally verifiable.

We believe that universal verifiability can be achieved under the root assumption in class groups with prime discriminant. A class group is defined by its discriminant Δ. It is conjectured that finding non-trivial roots in a class group with discriminant $\Delta = -p$ for a prime p is infeasible (cf. Hamdy [85]). In other words if G_Δ is such a class group and $g \in G_\Delta$ is randomly chosen no adversary can find a $b \in G_\Delta$ and $\eta \neq \pm 1$ such that $b^\eta = g$ with non-negligible probability. The idea is to generate a prime p of suitable size from random coins handed to the prover by the verifier in the first round. Then the integer part of the protocol is executed in the class group defined by $\Delta = -p$ instead of over the group of squares modulo the RSA-modulus $\mathbf{N}$. With this modification the protocol gives a universally verifiable mix-net in the random oracle model.

It is important to understand that universal verifiability in the random oracle model only gives heuristic security because of the dependence of the random oracle model. Furthermore, in practice we expect only a handful of outsiders to implement software to verify the actions of the mix-servers, and given the efficiency of the proof

```
; Load file with (load "complexity.scm")
; Compute complexity with e.g.
; (comp 2048 1024 160 100 50 (/ 1 12) (/ 1 3))

(define (veriStep2 k kN kc kpri kr fixbasefak simulfak)
  (/ (* 6 kc kc kc kc) (* k k k)))

(define (veriStep3 k kN kc kpri kr fixbasefak simulfak)
  (* simulfak 3 (/ kc k)))

(define (veriStep7 k kN kc kpri kr fixbasefak simulfak)
  (+ (* simulfak 3 (/ (+ kc kpri kr) k))
     (* (/ (* kN kN kN) (* k k k))
        (+ (+ (* fixbasefak (/ (+ kN kpri (* 2 kr)) kN))
              (/ (+ kc kpri kr) kN))
           (* fixbasefak
              (+ (/ (+ kN kpri (* 2 kr)) kN) (/ (+ kc kpri kr) kN)))
           (/ kc kN)))))

(define (veri k kN kc kpri kr fixbasefak simulfak)
  (float (+ (veriStep2 k kN kc kpri kr fixbasefak simulfak)
            (veriStep3 k kN kc kpri kr fixbasefak simulfak)
            (veriStep7 k kN kc kpri kr fixbasefak simulfak))))

(define (proStep1 k kN kc kpri kr fixbasefak simulfak)
  fixbasefak)

(define (proStep3 k kN kc kpri kr fixbasefak simulfak)
  (veriStep3 k kN kc kpri kr fixbasefak simulfak))

(define (proStep4 k kN kc kpri kr fixbasefak simulfak)
  (+ (* simulfak 3 (/ (+ kc kpri kr) k))
     (* (/ (* kN kN kN) (* k k k))
        (+ (* fixbasefak (/ (+ kN kr) kN))
           (/ kc kN)
           (* fixbasefak (+ (/ (+ kN kr) kN) (/ kc kN)))
           (* fixbasefak (/ (+ kN kpri (* 2 kr)) kN))
           (/ (+ kc kpri kr) kN)
           (* fixbasefak
              (+ (/ (+ kN kpri (* 2 kr)) kN)
                 (/ (+ kc kpri kr) kN)))))))

(define (pro k kN kc kpri kr fixbasefak simulfak)
  (float (+ (proStep1 k kN kc kpri kr fixbasefak simulfak)
            (proStep3 k kN kc kpri kr fixbasefak simulfak)
            (proStep4 k kN kc kpri kr fixbasefak simulfak))))
```

Table 9.2: Scheme program for estimation of the complexity of the protocol.

of a shuffle the mix-servers can readily answer such requests interactively. In an interactive proof the outsider can choose the RSA parameters, since the protocol is statistically zero-knowledge as long as $\mathbf{g}$ is in the group generated by $\mathbf{h}$. For this to hold the outsider must prove that $\mathbf{g}$ is contained in the group generated by $\mathbf{h}$.

9.7 A Proof of Knowledge of a Shuffle of Paillier Cryptotexts

Recall that we define the Paillier cryptosystem in Section 3.12. In this section we show how a re-encryption shuffle for the Paillier cryptosystem can be constructed by a small modification to the protocol described above. This protocol is then used in Chapter 11.

Recall that to re-encrypt a cryptotext $u \in \mathbb{Z}_{\mathsf{N}}^*$ using the public key N and generator $\mathsf{g_f}$ of the group of $2\mathsf{N}$th residues, one can compute $u\mathsf{g_f}^s \bmod \mathsf{N}^2$ for a randomly chosen integer $s \in [0, 2^{\kappa+\kappa_r} - 1]$. Here we assume that N is a κ-bit Paillier modulus. The re-encryption-permutation relation corresponding to the transformation computed by each mix-server in a mix-net using the re-encryption-permutation approach is then defined as follows.

Definition 9.20 (Knowledge of Correct Re-encryption-Permutation). Define for each N and N a relation $R_{\mathrm{RP}}^{\mathsf{N}} \subset ((\mathbb{Z}_{\mathsf{N}}^*)^N \times (\mathbb{Z}_{\mathsf{N}}^*)^N) \times [0, 2^{\kappa+\kappa_r} - 1]^N$, by

$$((\{u_i\}_{i=1}^N, \{u_i'\}_{i=1}^N), (a, (x_i)_{i=1}^N)) \in R_{\mathrm{RP}}^{\mathsf{N}}$$

precisely when $a < \sqrt{\mathsf{N}}/4$ equals one or is prime and $(u_i')^a = \mathsf{g_f}^{x_{\pi(i)}} u_{\pi(i)}^a \mod \mathsf{N}^2$ for $i = 1, \ldots, N$ and some permutation $\pi \in \Sigma_N$ such that the list $\{\bar{u}_i\}_{i=1}^N$ is sorted lexicographically.

Note that when $a < \sqrt{\mathsf{N}}/4$, a is relatively prime to N, since we assume that the factors of N are safe primes. Thus, a is invertible in $\mathbb{Z}_{\mathsf{N}}$ and u_i' and u_i encrypts identical messages. The reason for introducing a is to allow extraction of other witnesses than the witness used by the prover. This simplifies the analysis.

Protocol 9.21 (Proof of Re-encryption-Permutation). The common input consists of an RSA-modulus $\mathbf{N}$ and $\mathbf{g}, \mathbf{h} \in \mathrm{QR}_{\mathbf{N}}$, generators $g, g_1, \ldots, g_N \in G_q$, a public key N, and two lists $L = (u_i)_{i=1}^N$ and $L' = (u_i')_{i=1}^N$ such that $u_i' = \mathsf{g_f}^{x_{\pi(i)}} u_{\pi(i)} \mod \mathsf{N}^2$ for some permutation π of N elements. The private input to the prover consists of $(x_i)_{i=1}^N$, where $x_i \in [0, 2^{\kappa+\kappa_r} - 1]$.

1. The prover chooses $r_i' \in \mathbb{Z}_q$ randomly, computes $(w_i)_{i=1}^N = (g^{r_i'} g_{\pi^{-1}(i)})_{i=1}^N$, and hands $(w_i)_{i=1}^N$ to the verifier.

2. The verifier chooses random primes $p_1, \ldots, p_N \in [2^{\kappa_p - 1}, 2^{\kappa_p} - 1]$, and hands $(p_i)_{i=1}^N$ to the prover. Then the prover defines $x_s = \sum_{i=1}^N x_i p_i$.

3. Both parties compute $(U, W) = (\prod_{i=1}^N u_i^{p_i}, \prod_{i=1}^N w_i^{p_i})$.

4. The prover chooses $k_1 \in [0, 2^{\kappa+\kappa_r}-1]$, $k_2 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r}-1]$, $k_3 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r}-1]$, $l_1 \in [0, 2^{\kappa+\kappa_c+2\kappa_r} - 1]$, $l_2 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, $l_3 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, $l_s \in [0, 2^{\kappa+\kappa_c+\kappa_p+2\kappa_r+\log_2 N} - 1]$, and $l_{r'} \in \mathbb{Z}_q$ randomly. Then it chooses $t_i, t'_i \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r} - 1]$, $s_i, s'_i \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, and $r_i \in [0, 2^{\kappa_p+\kappa_c+\kappa_r} - 1]$ for $i = 1, \ldots, N$ randomly. Then it chooses $s \in [0, 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} - 1]$, and $s' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_r+\log_2 N} - 1]$ randomly and computes

$$(b_1, b_2, b_3) = (\mathbf{g}_{\mathsf{f}}^{k_1}\mathbf{g}_{\mathsf{f}}^{x_s}, \mathbf{h}^{k_2}\mathbf{g}^{k_1}, \mathbf{h}^{k_3}\mathbf{g}^{x_s}) \tag{9.56}$$

$$(\beta_1, \beta_2, \beta_3) = \left(\mathbf{g}_{\mathsf{f}}^{l_1}\mathbf{g}_{\mathsf{f}}^{l_s}, \mathbf{h}^{l_2}\mathbf{g}^{l_1}, \mathbf{h}^{l_3}\mathbf{g}^{l_s}\right) \tag{9.57}$$

$$(\alpha_1, \alpha_3) = \left(\mathbf{g}_{\mathsf{f}}^{l_1}\prod_{i=1}^{N}(u'_i)^{r_i}, g^{l_{r'}}\prod_{i=1}^{N} g_i^{r_i}\right) \tag{9.58}$$

$$\mathbf{b}_0 = \mathbf{g} \tag{9.59}$$

$$(\mathbf{b}_i, \mathbf{b}'_i)_{i=1}^{N} = (\mathbf{h}^{t_i}\mathbf{b}_{i-1}^{p_{\pi(i)}}, \mathbf{h}^{t'_i}\mathbf{g}^{p_{\pi(i)}})_{i=1}^{N} \tag{9.60}$$

$$(\boldsymbol{\gamma}_i, \boldsymbol{\gamma}'_i)_{i=1}^{N} = (\mathbf{h}^{s_i}\mathbf{b}_{i-1}^{r_i}, \mathbf{h}^{s'_i}\mathbf{g}^{r_i})_{i=1}^{N} \tag{9.61}$$

$$(\boldsymbol{\gamma}, \boldsymbol{\gamma}') = (\mathbf{h}^{s}, \mathbf{h}^{s'}) \ , \tag{9.62}$$

and $((b_1, b_2, b_3), (\beta_1, \beta_2, \beta_3), (\alpha_1, \alpha_2, \alpha_3), (\mathbf{b}_i, \mathbf{b}'_i)_{i=1}^{N}, (\boldsymbol{\gamma}_i, \boldsymbol{\gamma}'_i)_{i=1}^{N}, (\boldsymbol{\gamma}, \boldsymbol{\gamma}'))$ is handed to the verifier.

5. The verifier chooses $c \in [2^{\kappa_c-1}, 2^{\kappa_c} - 1]$ randomly and hands c to the prover.

6. The prover defines $t = t_N + p_{\pi(N)}(t_{N-1} + p_{\pi(N-1)}(t_{N-2} + p_{\pi(N-2)}(t_{N-3} + p_{\pi(N-3)}(\ldots))))$, and $t' = \sum_{i=1}^{N} t'_i$, $r' = \sum_{i=1}^{N} r'_i p_i$, and computes

$$\begin{aligned}
f_1 &= ck_1 + l_1 && \bmod 2^{\kappa+\kappa_c+2\kappa_r} \\
f_2 &= ck_2 + l_2 && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} \\
f_3 &= ck_3 + l_3 && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} \\
f_s &= cx_s + l_s && \bmod 2^{\kappa+\kappa_p+2\kappa_r+\log_2 N} \\
f_{r'} &= cr' + l_{r'} && \bmod q \\
(e_i, e'_i)_{i=1}^{N} &= (ct_i + s_i, ct'_i + s'_i)_{i=1}^{N} && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} \\
(d_i)_{i=1}^{N} &= (cp_{\pi(i)} + r_i)_{i=1}^{N} && \bmod 2^{\kappa_p+\kappa_c+\kappa_r} \\
e &= ct + s && \bmod 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} \\
e' &= ct' + s' && \bmod 2^{\kappa_{\mathbf{N}}+\kappa_c+\kappa_r+\log_2 N}
\end{aligned}$$

Then it hands $((f_1, f_2, f_s, f_{r'}), (e_i, e'_i)_{i=1}^{N}, (d_i)_{i=1}^{N}, (e, e'))$ to the verifier.

7. The verifier checks that L' is lexicographically sorted and that

$$(b_1^c\beta_1, b_2^c\beta_2, b_3^c\beta_3) = (\mathsf{g_f}^{f_1}\mathsf{g_f}^{f_s}, \mathbf{h}^{f_2}\mathbf{g}^{f_1}, \mathbf{h}^{f_3}\mathbf{g}^{f_s}) \tag{9.63}$$

$$((b_1U)^c\alpha_1, W^c\alpha_3) = \left(\mathsf{g_f}^{f_1}\prod_{i=1}^{N}(u_i')^{d_i}, g^{f_{r'}}\prod_{i=1}^{N}g_i^{d_i}\right) \tag{9.64}$$

$$(\mathbf{b}_i^c\boldsymbol{\gamma}_i, (\mathbf{b}_i')^c\boldsymbol{\gamma}_i')_{i=1}^N = (\mathbf{h}^{e_i}\mathbf{b}_{i-1}^{d_i}, \mathbf{h}^{e_i'}\mathbf{g}^{d_i})_{i=1}^N \tag{9.65}$$

$$(\mathbf{g}^{-\prod_{i=1}^{N}p_i}\mathbf{b}_N)^c\boldsymbol{\gamma} = \mathbf{h}^e \tag{9.66}$$

$$\left(\mathbf{g}^{-\sum_{i=1}^{N}p_i}\prod_{i=1}^{N}\mathbf{b}_i'\right)^c\boldsymbol{\gamma}' = \mathbf{h}^{e'}\ . \tag{9.67}$$

Security Analysis

The protocol satisfies similar security properties as Protocol 9.1.

Proposition 9.22. *Protocol 9.21 is honest verifier statistical zero-knowledge.*

Proposition 9.22. The simulator chooses $p_1, \ldots, p_N$ and c honestly. Then it chooses $f_1 \in [0, 2^{\kappa+\kappa_c+2\kappa_r} - 1]$, $f_2 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, $f_3 \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]$, $f_s \in [0, 2^{\kappa+\kappa_c+\kappa_p+2\kappa_r+\log_2 N} - 1]$, $f_{r'} \in \mathbb{Z}_q$, $e \in [0, 2^{\kappa_{\mathbf{N}}+N\kappa_p+\kappa_c+\kappa_r+\log_2 N} - 1]$, $e' \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+\kappa_r+\log_2 N} - 1]$, $(e_i, e_i')_{i=1}^N \in [0, 2^{\kappa_{\mathbf{N}}+\kappa_c+2\kappa_r} - 1]^N$, and a list $(d_i)_{i=1}^N \in [0, 2^{\kappa_p+\kappa_c+\kappa_r} - 1]^N$ randomly. It also chooses $b_1, b_2, b_3 \in G_q$ and $\mathbf{b}_i, \mathbf{b}_i' \in \mathrm{QR}_{\mathbf{N}}$ randomly. Finally, the simulator defines, $(\beta_1, \beta_2, \beta_3)$ by Equations (9.63), (α_1, α_3) by Equations (9.64), $\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}_i'$ by Equation (9.65), $\boldsymbol{\gamma}$ by Equation (9.66), and $\boldsymbol{\gamma}'$ by Equation (9.67) respectively. It is easy to see that the distribution of the resulting elements is statistically close to the distribution of the corresponding elements in a real execution of the protocol. □

Proposition 9.23. *Protocol 9.21 is a computationally convincing proof of knowledge for the relation $R_{\mathrm{RP}}^{\mathsf{N}}$ with regards to the distribution of the special parameters $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(g, g_1, \ldots, g_N)$.*

Proposition 9.24. *Protocol 9.21 has overwhelming completeness, also if* PGen *is used for prime generation. If* $\mathsf{PGen}^{\mathsf{c}}$ *is used instead, then this holds in Cramér's model of the primes.*

The proof of this proposition is almost identical to the proof of Proposition 9.9 and omitted.

9.7.1 Proof Sketch of Proposition 9.23

The generation of suitable transcripts is almost identical to how this is done in the proof of Proposition 9.8. Lemma 9.14 on permutation matrices can be applied unchanged. Lemma 9.12 and Corollary 9.13 holds with minor modifications. One

must argue that a random prime vector is linearly independent both over $\mathbb{Z}_q$ and over $\mathbb{Z}_{\mathsf{N}}$, but this follows straightforwardly. The argument why the protocol is a computationally convincing proof of knowledge is unchanged except that the relation R_{DP} is replaced by $R^{\mathsf{N}}_{\mathrm{RP}}$. The only essential modification needed is in Lemma 9.10 and we give the details.

We denote the jth view in a list of views of the honest verifier by $T_j = (I_j, W_j, P_j, C_j, c_j, R_j)$ where I_j denotes the common input, W_j denotes the list of commitments w_i, and

$$\begin{aligned} P_j &= (p_{j,1}, \ldots, p_{j,N}) \\ C_j &= ((b_{j,1}, b_{j,2}, b_{j,3}), (\beta_{j,1}, \beta_{j,2}, \beta_{j,3}), (\alpha_{j,1}, \alpha_{j,3}), \\ &\quad (\mathbf{b}_{j,i}, \mathbf{b}'_{j,i})_{i=1}^{N}, (\boldsymbol{\gamma}_{j,i}, \boldsymbol{\gamma}'_{j,i})_{i=1}^{N}, (\boldsymbol{\gamma}_j, \boldsymbol{\gamma}'_j)) \\ R_j &= ((f_{j,1}, f_{j,2}, f_{j,3}, f_{j,x}, f_{j,s}, f_{j,r'}), (e_{j,i}, e'_{j,i})_{i=1}^{N}, (d_{j,i})_{i=1}^{N}, (e_j, e'_j)) \ . \end{aligned}$$

Lemma 9.25. *Lemma 9.10 holds also for Protocol 9.21, but with the modified notation and with the main conclusion replaced by*

1. Main Conclusion. *A prime integer $a < \sqrt{\mathsf{N}}/4$, elements $x_1, \ldots, x_N \in \mathbb{Z}$ such that*
$$((L^a, (L')^a), (x_i)_{i=1}^{N}) \in R^{\mathsf{N}}_{\mathrm{RP}} \ ,$$
where we by L^a and $(L')^a$ mean term-wise exponentiation.

Proof. The proof is identical to the proof of Lemma 9.10 except for the last section of the proof. This must be replaced by the following.

The Common Input (L, L') Satisfies $((L, L'), (a, (x_i)_{i=1}^{N})) \in R^{\mathsf{N}}_{\mathrm{RP}}$. From the above we may assume that $\pi_j = \pi$ for all $j = 1, \ldots, N$, so we drop the subscript and simply write π from now on. Equations (9.63) imply that

$$\begin{aligned} b_1^{c_{j+N}-c_j} &= \mathsf{g}_{\mathsf{f}}^{f_{j+N,1}-f_{j,1}} \mathsf{g}_{\mathsf{f}}^{f_{j+N,s}-f_{j,s}} \ , \\ b_2^{c_{j+N}-c_j} &= \mathbf{h}^{f_{j+N,2}-f_{j,2}} \mathbf{g}^{f_{j+N,1}-f_{j,1}} \ , \text{ and} \\ b_3^{c_{j+N}-c_j} &= \mathbf{h}^{f_{j+N,3}-f_{j,3}} \mathbf{g}^{f_{j+N,s}-f_{j,s}} \ . \end{aligned}$$

If $c_{j+N}-c_j$ does not divide $f_{j+N,1}-f_{j,1}$, $f_{j+N,2}-f_{j,2}$, $f_{j+N,3}-f_{j,3}$, and $f_{j+N,s}-f_{j,s}$ Conclusion 2a of the lemma is satisfied similarly as in the proof of Lemma 9.10.

Thus, we may assume that it does and define

$$\kappa_1 = (f_{j+N,1} - f_{j,1})/(c_{j+N} - c_j) \ , \tag{9.68}$$
$$\kappa_2 = (f_{j+N,2} - f_{j,2})/(c_{j+N} - c_j) \ , \tag{9.69}$$
$$\kappa_3 = (f_{j+N,3} - f_{j,3})/(c_{j+N} - c_j) \ , \text{ and} \tag{9.70}$$
$$\xi_{j,s} = (f_{j+N,s} - f_{j,s})/(c_{j+N} - c_j) \ . \tag{9.71}$$

Then we have

$$b_1 = \mathsf{g}_{\mathsf{f}}^{\kappa_1} \mathsf{g}_{\mathsf{f}}^{\xi_{j,s}} \ . \tag{9.72}$$

From Equation (9.64) we have

$$(b_1 U_j)^{c_{j+N}-c_j} = \mathsf{g}_\mathsf{f}^{f_{j+N,1}-f_{j,1}} \prod_{i=1}^{N} (u'_i)^{d_{j+N,i}-d_{j,i}} \ .$$

Our definitions of κ_1 and κ_2 in Equations (9.68) and (9.69), and the definition of ρ_i, which equals $p_{j,\pi(i)}$ in Equation (9.21) imply that

$$b_1 U_j = \mathsf{g}_\mathsf{f}^{\kappa_1} \prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \ .$$

If we combine Equation (9.72) with the equation above we have

$$\mathsf{g}_\mathsf{f}^{\xi_{j,s}} \prod_{i=1}^{N} u_i^{p_{ji}} = \mathsf{g}_\mathsf{f}^{\xi_{j,s}} U_j = \prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \ . \tag{9.73}$$

for $j = 1, \ldots, N$. Here we can not apply the coefficients $a_{l,1}, \ldots, a_{l,N} \in \mathbb{Z}_q$ introduced in the proof of Lemma 9.10. The problem is that we need that the lists $P_1, \ldots, P_N$ are linearly independent as vectors over $\mathbb{Z}_{\mathsf{Nf}}$, since Nf is the order of the group $\mathbb{Z}^*_{\mathsf{N}^2}$. The problem with this is that in most natural applications of the proof of a shuffle we must be able to extract a witness without knowledge of f. In other words we do not know how to invert elements in $\mathbb{Z}_{\mathsf{Nf}}$.

Despite this we can perform Gauss elimination over the integers and start with column l. This gives a list of integers $a_{l,1}, \ldots, a_{l,N}$ such that $\sum_{j=1}^{N} a_{l,j} P_j = (\delta_{l,1}, \ldots, \delta_{l,N})$ with $\delta_{l,j} = 0$ for $j \neq l$ and $\delta_{l,l} = p_{l,l}$ for some integer a. We apply these coefficients to Equation (9.73) and conclude that

$$\begin{aligned} \mathsf{g}_\mathsf{f}^{\sum_{j=1}^{N} \xi_{j,s} a_{l,j}} u_l^{p_{l,l}} &= \prod_{j=1}^{N} \left(g^{\xi_{j,s}} \prod_{i=1}^{N} u_i^{p_{ji}} \right)^{a_{l,j}} \\ &= \prod_{j=1}^{N} \left(\prod_{i=1}^{N} (u'_i)^{p_{j,\pi(i)}} \right)^{a_{l,j}} = (u'_{\pi^{-1}(l)})^{p_{l,l}} \ . \end{aligned}$$

This concludes the proof, since we can set $x_l = \sum_{j=1}^{N} \xi_{j,s} a_{l,j}$. □

Chapter 10

Secure Realizations of Two Ideal Functionalities

To complete the construction and analysis of the mix-net presented in Chapter 8 we must securely realize the ideal functionality, $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, of a proof of knowledge of a correct decryption-permutation, and the ideal functionality, $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$, for a proof of knowledge of the cleartext of an El Gamal cryptotext.

We have in fact done most of the work needed to solve the first problem in Chapter 9. The second problem is solved using ideas of Feldman [63] and Abe, Cramer, and Fehr [3].

Since the protocols in this chapter are used as subprotocols in the mix-net of Chapter 8, the adversary model we consider is taken from that chapter.

10.1 A Proof of Knowledge of a Cleartext

In this section we securely realize the ideal functionality for a zero-knowledge proof of knowledge of a cleartext of an El Gamal cryptotext.

We observe that we may view the verifiable secret sharing scheme (VSS) of Feldman [63] as a multi-verifier proof of knowledge of a logarithm. A similar approach is used by Abe, Cramer, and Fehr [3].

Intuitively, the protocol works as follows. A prover shares his witness to the relation R_{C}, and uses a semantically secure cryptosystem over the authenticated bulletin board $\mathcal{F}_{\mathrm{BB}}$ to distribute the shares. The verifiers check their shares, and write the result of their verification on the bulletin board. Each verifier then checks that all verifiers accepted their shares.

An alternative way to securely realize $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}}$ is to use the double-cryptotext trick of Naor and Yung [111]. The drawback of this approach is that the prover must interact with the verifiers. In Chapter 11 we solve the problem of extracting a Paillier cleartext in this way. Interaction can be eliminated in the random oracle model, but then security is only heuristic. We do not follow this path.

The protocol below has both provers and verifiers. In our applications the verifiers correspond to mix-servers. The provers on the other hand may be either senders or mix-servers depending on the application. Recall that $k' = \lceil (k+1)/2 \rceil$ denotes the number of mix-servers needed for majority.

Protocol 10.1 (Zero-Knowledge Proof of Knowledge of Cleartext). The zero-knowledge proof of knowledge of a cleartext protocol $\pi = (S_1, \ldots, S_N, M_1, \ldots, M_k)$ consists of provers S_i, and verifiers M_j.

PROVER S_i.

1. Wait until $(\cdot, M_j, \texttt{Keys}, y_{j,1}, \ldots, y_{j,N})$ appears on $\mathcal{F}_{\text{BB}}$ for $j = 1, \ldots, k$.
2. Wait for input $(\texttt{Prover}, (g, y, u_i, v_i), r_i)$, where $g, y, u_i, v_i \in G_q$ and $r_i \in \mathbb{Z}_q$. If $u_i \neq g^{r_i}$ set $r_i = 0$. Then proceed as follows.
   a) Choose $a_{i,l} \in \mathbb{Z}_q$ randomly, define $p_i(x) = r + \sum_{l=1}^{k'-1} a_{i,l} x^l$, and compute
   $$\begin{aligned} \alpha_{i,l} &= g^{a_{i,l}}, \quad \text{for } l = 1, \ldots, k'-1 \ , \\ s_{i,j} &= p_i(j), \quad \text{for } j = 1, \ldots, k \ , \quad \text{and} \\ C_{i,j} &= E_{y_{j,i}}(s_{i,j}), \quad \text{for } j = 1, \ldots, k \ . \end{aligned}$$
   b) Then hand $(\texttt{Write}, \texttt{Proof}, (g, y, u_i, v_i), (\alpha_{i,1}, \ldots, \alpha_{i,k'-1}, C_{i,1}, \ldots, C_{i,k}))$ to $\mathcal{F}_{\text{BB}}$.

VERIFIER M_j.

1. Generate El Gamal keys $(x_{j,i}, y_{j,i})$ for $i = 1, \ldots, N$, and hand $(\texttt{Write}, \texttt{Keys}, y_{j,1}, \ldots, y_{j,N})$ to $\mathcal{F}_{\text{BB}}$.
2. Repeatedly wait for an input $(\texttt{Question}, S_i, (g, y, u_i, v_i))$ and then do
   a) Wait until $(\cdot, S_i, \texttt{Proof}, (g, y, u_i, v_i), (\alpha_{i,1}, \ldots, \alpha_{i,k'-1}, C_{i,1}, \ldots, C_{i,k}))$ appears on $\mathcal{F}_{\text{BB}}$.
   b) Compute $s_{i,j} = D_{x_{j,i}}(C_{i,j})$, and verify that $g^{s_{i,j}} = u_i \prod_{l=1}^{k'-1} \alpha_{i,l}^{j^l}$. If so set $b_{j,i} = 1$, and otherwise $b_{j,i} = x_{j,i}$. Hand $(\texttt{Write}, \texttt{Judgement}, S_i, b_{j,i})$ to $\mathcal{F}_{\text{BB}}$.
   c) Wait until $(\cdot, M_l, \texttt{Judgement}, S_i, b_{l,i})$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$.
   d) Do the following for $l = 1, \ldots, k$:
      i. If $b_{l,i} = 1$, then set $b'_{l,i} = 1$.
      ii. If $b_{l,i} \neq 1$ then check if $y_{l,i} = g^{b_{l,i}}$. If not set $b'_{l,i} = 1$. If so compute $s'_{i,j} = D_{b_{l,i}}(C_{i,l})$, and verify that $g^{s'_{i,j}} = u_i \prod_{\iota=1}^{k'-1} \alpha_{i,\iota}^{l^\iota}$. If so set $b'_{l,i} = 1$ and otherwise set $b'_{l,i} = 0$.
   e) If $\sum_{l=1}^{k} b'_{l,i} = k$ set $b = 1$ and otherwise 0. Then output $(\texttt{Verifier}, S_i, (g, y, u_i, v_i), b)$.

Theorem 10.2. *Protocol 10.1 securely realizes the ideal functionality $\mathcal{F}_{\text{ZK}}^{R_{\text{C}}}$ in the $\mathcal{F}_{\text{BB}}$-hybrid model with respect to $\mathcal{M}_{k/2}$-adversaries under the DDH-assumption.*

The above protocol differs from the original protocol of Feldman [63] in that it does not require any interaction from the prover. To achieve this each verifier must generate an El Gamal key for each prover. The number of keys can be reduced to one key for each mix-server if we use a CCA2-secure cryptosystem, but then the mix-servers can not simply reveal their secret key to substantiate a complaint. Instead, a complaining mix-server would have to prove that its complaint was valid.

A CCA-secure cryptosystem that has the property that a decryptor can show directly to a third party the contents of a cryptotext without revealing its key would also solve the problem. Such a cryptosystem can be constructed under strong assumptions (cf. [39]). We do not use this solution, since we wish to use as weak assumptions as possible.

Proof of Theorem 10.2. We describe an ideal adversary $\mathcal{S}(\cdot)$ that runs any hybrid adversary $\mathcal{A}' = \mathcal{A}^{\mathcal{S}_{\text{BB}}}$ as a black-box. Then we show that if $\mathcal{S}$ does not imply that the protocol is secure, then we can break the DDH-assumption.

The Ideal Adversary $\mathcal{S}$. Let I_S and I_M be the set of indices of parties corrupted by $\mathcal{A}$ of the sender type and the mix-server type respectively. The ideal adversary $\mathcal{S}$ corrupts the dummy parties $\tilde{S}_i$ for which $i \in I_S$, and the dummy parties $\tilde{M}_j$ for which $j \in I_M$. The ideal adversary is best described by starting with a copy of the original hybrid ITM-graph

$$(V, E) = \mathcal{Z}'(\mathcal{H}(\mathcal{A}', \pi^{\tilde{\pi}^{\mathcal{F}_{\text{BB}}}}) \ ,$$

where we have replaced $\mathcal{Z}$ by a machine $\mathcal{Z}'$.

The adversary $\mathcal{S}$ simulates all machines in V except for those in $\mathcal{A}'$, and the corrupted machines S_i for $i \in I_S$ and M_i for $i \in I_M$ under $\mathcal{A}'$:s control.

The ideal adversary $\mathcal{S}$ simulates the ideal functionality $\mathcal{F}_{\text{BB}}$ honestly.

Simulation of Links $(\mathcal{Z}, \mathcal{A})$, $(\mathcal{Z}, S_i)$ for $i \in I_S$, and $(\mathcal{Z}, M_j)$ for $j \in I_M$. $\mathcal{S}$ simulates $\mathcal{Z}'$, $\tilde{S}_i$, for $i \in I_S$, and $\tilde{M}_j$ for $j \in I_M$, such that it appears as if $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ and S_i for $i \in I_S$, and $\mathcal{Z}$ and M_j for $j \in I_M$ are linked directly.

1. When $\mathcal{Z}'$ receives m from $\mathcal{A}$, m is written to $\mathcal{Z}$, by $\mathcal{S}$. When $\mathcal{S}$ receives m from $\mathcal{Z}$, m is written to $\mathcal{A}$ by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and $\mathcal{A}$ are linked directly.

2. When $\mathcal{Z}'$ receives m from S_i for $i \in I_S$, m is written to $\mathcal{Z}$ by $\tilde{S}_i$. When $\tilde{S}_i$, $i \in I_S$, receives m from $\mathcal{Z}$, m is written to S_i by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and S_i are linked directly for $i \in I_S$.

3. When $\mathcal{Z}'$ receives m from M_j for $j \in I_M$, m is written to $\mathcal{Z}$ by $\tilde{M}_j$. When $\tilde{M}_j$, $j \in I_M$, receives m from $\mathcal{Z}$, m is written to M_j by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and M_j are linked directly for $j \in I_M$.

Simulation of Honest Verifiers. When an honest verifier $\tilde{M}_j$, for $j \notin I_M$, receives $(\texttt{Question}, S_i, (g, y, u_i, v_i))$ from $\mathcal{Z}$, $\mathcal{S}$ must ensure that the simulated honest verifier M_j receives $(\texttt{Question}, S_i, (g, y, u_i, v_i))$ from $\mathcal{Z}'$. When the simulated honest verifier M_j hands $(\texttt{Verifier}, S_i, (g, y, u_i, v_i), b)$ to $\mathcal{Z}'$, $\mathcal{S}$ must ensure that $(\texttt{Verifier}, S_i, (g, y, u_i, v_i), b)$ is delivered to $\tilde{M}_j$. This is done as follows.

1. Let $j \notin I_M$. If $\mathcal{S}$ receives $((\mathcal{S}, \tilde{M}_j, \texttt{Verifier}, \tilde{S}_i, (g, y, u_i, v_i), b), (\tilde{M}_j, \tau_j))$ from $\mathcal{F}_{\mathrm{ZK}}^{R_C}$, $\mathcal{Z}'$ hands $(\texttt{Question}, S_i, (g, y, u_i, v_i))$ to M_j.
2. Let $j \notin I_M$. If $\mathcal{Z}'$ receives $(\texttt{Verifier}, S_i, (g, y, u_i, v_i), b)$ from M_j, $\mathcal{S}$ hands $(1, \tau_j)$ to $\mathcal{C}_{\mathcal{I}}$, i.e. $\mathcal{S}$ instructs $\mathcal{C}_{\mathcal{I}}$ to deliver $(\texttt{Verifier}, \tilde{S}_i, (g, y, u_i, v_i), b)$ to $\tilde{M}_j$.

Simulation of Honest Provers. If an honest dummy prover $\tilde{S}_i$, for $i \notin I_S$, receives a message $(\texttt{Prover}, (g, y, u_i, v_i), r_i)$, with $u_i = g^{r_i}$, from $\mathcal{Z}$, $\mathcal{S}$ must ensure that S_i constructs a simulated proof deemed valid by the verifiers M_j despite that $\mathcal{S}$ does not know r_i. To be able to do this $\mathcal{S}$ must ensure that the simulated honest mix-servers M_j, for $j \notin I_M$, do not complain. This is done as follows.

1. Let $j \notin I_M$. M_j follows its program except that if $i \notin I_S$ the ideal adversary $\mathcal{S}$ checks if it has received $(\mathcal{S}, \tilde{S}_i, \texttt{Prover}, (g, y, u_i, v_i), b)$. If so M_j sets $b_{j,i} = 1$ in Step 2b without decrypting anything. Otherwise it is simulated honestly.
2. Suppose that $\mathcal{S}$ receives $(\mathcal{S}, \tilde{S}_i, \texttt{Prover}, (g, y, u_i, v_i), b)$ from $\mathcal{F}_{\mathrm{ZK}}^{R_C}$ for $i \notin I_S$. If $b = 0$, then it hands $(\texttt{Prover}, (g, y, u_i, v_i), 0)$ to S_i which is simulated honestly.

   If $b = 1$, then it hands $(\texttt{Prover}, (g, y, u_i, v_i), \cdot)$ to S_i, where Step 2a in the program of S_i is replaced by the following.

   Without loss we assume that $I_M = \{1, \ldots, k_{\mathrm{corr}}\}$, where $k_{\mathrm{corr}} \leq k' - 1$. $\mathcal{S}$ chooses $s_{i,j} \in \mathbb{Z}_q$ randomly for $j = 1, \ldots, k' - 1$. We must now define $\alpha_{i,1}, \ldots, \alpha_{i,k'-1}$ such that $g^{s_{i,j}} = u_i \prod_{l=1}^{t} \alpha_{i,l}^{j^l}$ for $j \in I_M$. Set $s_{i,0} = r$, but note that r is not known by $\mathcal{S}$ and not used in the simulation. Using Lagrange's interpolation formula we define $a_{i,1}, \ldots, a_{i,k'-1}$ by $\sum_{l=0}^{t} a_{i,l} z^l = \sum_{j=0}^{k'-1} s_{i,j} \prod_{l \neq j} \frac{z-l}{j-l}$, where the product is over $l \in \{0, \ldots, k' - 1\}$. Thus $a_{i,l} = \sum_{l'=0}^{k'-1} b_{i,l'} s_{i,l'}$ for some $b_{i,l}$ and $\alpha_{i,l}$ can be computed by forming $\alpha_{i,l} = u^{b_{i,0}} g^{\sum_{l'=1}^{k'-1} b_{i,l'} s_{i,l'}}$ for $l = 1, \ldots, k' - 1$. To complete Step 2a, S_i computes

$$C_j = E_{y_{j,i}}(s_{i,j}), \quad \text{for } j = 1, \ldots, k' - 1 \ , \quad \text{and}$$
$$C_j = E_{y_{j,i}}(1), \quad \text{for } j = k', \ldots, k \ .$$

Note that all components of the (corrupt) proof of S_i above except C_j for $j \notin I_M$ are identically distributed to the proof of a prover following its program.

Extraction from Corrupt Provers. If a corrupt prover S_i, for $i \in I_S$, constructs a valid proof of knowledge, $\mathcal{S}$ must extract the knowledge and forward it to $\mathcal{F}_{\text{ZK}}^{R_C}$. $\mathcal{S}$ does this as follows.

1. Suppose that $(\cdot, S_i, \texttt{Proof}, (g, y, u_i, v_i), (\alpha_{i,1}, \ldots, \alpha_{i,t}, C_{i,1}, \ldots, C_{i,k}))$, for $i \in I_S$, appears on $\mathcal{F}_{\text{BB}}$. $\mathcal{S}$ interrupts the simulation of $\mathcal{F}_{\text{BB}}$ when a message on the form $(M_j, \texttt{Verifier}, S_i, (g, y, u_i, v_i), b_{j,i})$ appears on for $j = 1, \ldots, k$. Then $\mathcal{S}$ checks if honest verifiers will output 1. If so $\mathcal{S}$ does the following.
   a) It computes $s_{i,j} = D_{x_{j,i}}(C_{i,j})$ for $j \notin I_M$.
   b) It computes r_i using Lagrange interpolation $r_i = \sum_{j=1}^{k'} s_j \prod_{l \neq j} \frac{-l}{j-l}$.
   c) Finally $\mathcal{S}$ hands $(\texttt{Prover}, (g, y, u_i, v_i), r_i)$ to $\tilde{S}_i$ (who forwards it to $\mathcal{F}_{\text{ZK}}^{R_C}$). When $\mathcal{S}$ receives $(\mathcal{S}, \tilde{S}_i, \texttt{Prover}, (g, y, u_i, v_i), b)$ from $\mathcal{F}_{\text{ZK}}^{R_C}$ it continues the simulation of $\mathcal{F}_{\text{BB}}$.

Reaching a Contradiction. Next we show, using a hybrid argument, that if the ideal adversary $\mathcal{S}$ defined above does not imply that Protocol 10.1 is secure, then we can break the DDH-assumption.

Suppose that $\mathcal{S}$ does not imply the security of the protocol. Then there exists a hybrid adversary $\mathcal{A}' = \mathcal{A}^{\mathcal{S}_{\text{BB}}}$, an environment $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$, a constant $c > 0$ and an infinite index set $\mathcal{N} \subset \mathbb{N}$ such that for $n \in \mathcal{N}$

$$|\Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\text{ZK}}^{R_C}})) = 1] - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}', \pi^{\tilde{\pi}^{\mathcal{F}_{\text{BB}}}})) = 1]| \geq \frac{1}{n^c},$$

where $\mathcal{S}$ runs $\mathcal{A}'$ as a black-box as described above, i.e. $\mathcal{S} = \mathcal{S}(\mathcal{A}')$.

Defining the Hybrids. Without loss we assume that $\{1, \ldots, N\} \backslash I_S = \{1, \ldots, \eta\}$. We define $T_0 = \mathcal{Z}_z(\mathcal{I}(\mathcal{S}(\mathcal{A}'), \tilde{\pi}^{\mathcal{F}_{\text{ZK}}^{R_C}}))$, and then define T_s by the following modifications to T_0.

1. When $\mathcal{S}$ receives $(\texttt{Prover}, \tilde{S}_i, (g, y, u_i, v_i), b)$ for $i \notin I_M$, it checks if $i \in \{1, \ldots, s\}$. If so, $\mathcal{S}$ consults the internal storage of $\mathcal{F}_{\text{ZK}}^{R_C}$ and finds the r_i stored under the tag $(\tilde{S}_i, (g, y, u_i, v_i))$. Then it runs the original honest S_i following the protocol on input $(\texttt{Prover}, (g, y, u_i, v_i), r_i)$. If $i \notin \{1, \ldots, s\}$, then the simulation of S_i proceeds as outlined above.

By inspection of the constructions we see that T_η is identically distributed to $\mathcal{Z}_z(\mathcal{H}(\mathcal{A}', \pi^{\tilde{\pi}^{\mathcal{F}_{\text{BB}}}}))$, since the only essential difference is that honest verifiers do not verify the proofs of all honest provers, but this is never noticed by $\mathcal{A}'$ or $\mathcal{Z}$.

If we set $p_l = \Pr[T_l = 1]$, we have $\frac{1}{\kappa^c} \leq |p_0 - p_\eta| \leq \sum_{l=1}^{\eta} |p_{l-1} - p_l|$, which implies that there exists some fixed $0 < l \leq \eta$ such that $|p_{l-1} - p_l| \geq \frac{1}{\eta\kappa^c} \geq \frac{1}{N\kappa^c}$.

Defining a Distinguisher. Without loss we assume $\{1, \ldots, k\} \backslash I_M = \{1, \ldots, k_{\text{hon}}\}$. We are now finally ready to define a distinguisher D. D is confronted with the generalized indistinguishability experiment of Lemma 2.13. It receives $(y'_1, \ldots, y'_{k_{\text{hon}}})$ at the start.

D does the following. For $j \notin I_M$, it alters M_j such that it does not generate the El Gamal keys $(x_{j,l}, y_{j,l})$, but use y'_j instead (for which it does not know the corresponding private key x'_j). Then it simulates T_l until S_l has received $(\texttt{Prover}, (g, y, u_l, v_l), r_l)$ and computed $C_{l,1}, \ldots, C_{l,k}$ in the computation of Step 2a of the protocol.

D then defines the pair $[\{m_{0,j}\}_{j=1}^{k_{\text{hon}}}, \{m_{1,j}\}_{j=1}^{k_{\text{hon}}}]$ by

$$m_{0,j} = 1 \quad \text{and} \quad m_{1,j} = s_{l,j} \ ,$$

and sends this pair to the encryption oracle. The encryption oracle returns a list $(C_1, \ldots, C_{k_{\text{hon}}})$. S_l replaces $(C_{l,1}, \ldots, C_{l,k_{\text{hon}}})$ by $(C_1, \ldots, C_{k_{\text{hon}}})$ and continues its execution. Finally, D outputs as its guess the bit b' output by the simulation.

Note that if $b = 0$ the simulation is a simulation of T_{l-1}, and if $b = 1$ it is a simulation of T_l. It follows from Lemma 2.13 that this contradicts the polynomial indistinguishability of the El Gamal cryptosystem. From Lemma 3.9 we conclude that the DDH-assumption is broken. □

10.2 A Proof of Knowledge of a Shuffle of El Gamal Cryptotexts

In this section we transform the zero-knowledge proof of knowledge of correct decryption-permutation from Chapter 9 into a secure realization of $\mathcal{F}_{\text{ZK}}^{R_{\text{DP}}}$ in a $(\mathcal{F}_{\text{RSA}}, \mathcal{F}_{\text{ZK}}^{R_{\text{C}}}, \mathcal{F}_{\text{CF}}, \mathcal{F}_{\text{BB}})$-hybrid model, where $\mathcal{F}_{\text{RSA}}$ is the RSA common reference string functionality, and $\mathcal{F}_{\text{CF}}$ is the coin-flipping functionality defined in Section 5.3. In the protocol below one of the mix-servers play the role of the prover in the original protocol and all the other mix-servers implement the honest verifier.

Protocol 10.3 (Zero-Knowledge Proof of Decryption-Permutation). The protocol $\pi_{\text{DP}} = (M_1, \ldots, M_k)$ consists of mix-servers M_j and proceeds as follows.

MIX-SERVER M_j. Each mix-server M_j proceeds as follows.

1. Wait for $(\texttt{RSA}, \mathbf{N}, \mathbf{g}, \mathbf{h})$ from $\mathcal{F}_{\text{RSA}}$.
2. Hand $(\texttt{GenerateCoins}, N(\kappa + 1))$ to $\mathcal{F}_{\text{CF}}$ and wait until it returns $(\texttt{Coins}, (g_1, \ldots, g_N))$.
3. Repeatedly wait for inputs
   - On input $(\texttt{Prover}, (g, z, y, L, L'), (w, x))$ ignore further inputs on the form $(\texttt{Prover}, \cdot, \cdot)$ and do
     a) Check if $((g, z, y, L, L'), (w, x)) \in L_{R_{\text{DP}}}$. If not, set $w = 0$ and $x = 0$.
     b) Hand $(\texttt{Prover}, (g, z, 1, 1), w)$ to $\mathcal{F}_{\text{ZK}}^{R_{\text{C}}(z)}$ and hand $(\texttt{Prover}, (g, y, 1, 1), x)$ to $\mathcal{F}_{\text{ZK}}^{R_{\text{C}}(y)}$.

c) Denote by W the first message of the prover in Protocol 9.1. Then hand $(\texttt{Write}, \texttt{W}, W)$ to $\mathcal{F}_{\mathrm{BB}}$.

d) Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$. Wait until it returns $(\texttt{Coins}, s)$ and let $P = \mathsf{PGen}(\mathsf{PRG}(s))$ be the primes used by the prover in Protocol 9.1.

e) Denote by C the second message of the prover in Protocol 9.1. Hand $(\texttt{Write}, \texttt{C}, C)$ to $\mathcal{F}_{\mathrm{BB}}$. Then hand $(\texttt{GenerateCoins}, \kappa_c - 1)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c')$. Let $c = c' + 2^{\kappa_c - 1}$ be the final challenge in Protocol 9.1.

f) Denote by R the third message of the prover in Protocol 9.1. Hand $(\texttt{Write}, \texttt{R}, R)$ to $\mathcal{F}_{\mathrm{BB}}$.

- On input $(\texttt{Question}, M_l, (g, z, y, L, L'))$, where $L, L' \in G_q^{2N}$ and $(z, y) \in G_q$ do

a) Hand $(\texttt{Question}, M_l, (g, z, 1, 1))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}$ and wait until it returns $(\texttt{Verifier}, M_l, b_{z,l})$. Hand $(\texttt{Question}, M_l, (g, y, 1, 1))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}$ and wait until it returns $(\texttt{Verifier}, M_l, b_{y,l})$. If $b_{z,l} = 0$ or $b_{y,l} = 0$ output $(\texttt{Verifier}, M_l, 0)$.

b) Wait until $(M_l, \texttt{W}, W)$ appears on $\mathcal{F}_{\mathrm{BB}}$. Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, s)$. Let $P = \mathsf{PGen}(\mathsf{PRG}(s))$ be the primes used by the verifier in Protocol 9.1.

c) Wait until $(M_l, \texttt{C}, C)$ appears on $\mathcal{F}_{\mathrm{BB}}$. Then hand $(\texttt{GenerateCoins}, \kappa_c - 1)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c')$, and until $(M_l, \texttt{R}, R)$ appears on $\mathcal{F}_{\mathrm{BB}}$. Let $c = c' + 2^{\kappa_c - 1}$ be the final challenge in Protocol 9.1. Then verify (W, P, C, c, R) as in Protocol 9.1 and set $b_j = 1$ or $b_j = 0$ depending on the result. Finally, output $(\texttt{Verifier}, M_l, L, L', b_j)$.

Theorem 10.4. *The ideal functionality $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ is securely realized by π_{DP} in the $(\mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{CF}}, \mathcal{F}_{\mathrm{RSA}}, \mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}})$-hybrid model with respect to $\mathcal{M}_{k/2}$-adversaries under the DL-assumption and the strong RSA-assumption. If the generator $\mathsf{PGen}^{\mathsf{c}}$ is used instead, then the result holds in Cramér's model of the primes.*

Proof. We describe an ideal adversary $\mathcal{S}(\cdot)$ that runs any hybrid adversary $\mathcal{A}$ as a black-box. Then we show that if $\mathcal{S}$ does not imply that the protocol is secure, then we can break one of the assumptions.

The Ideal Adversary $\mathcal{S}$. Let I_M be the set of mix-servers corrupted by $\mathcal{A}$. The ideal adversary $\mathcal{S}$ corrupts the dummy parties $\tilde{M}_i$ for which $i \in I_M$. The ideal adversary is best described by starting with a copy of the original hybrid ITM-graph

$$(V, E) = \mathcal{Z}'(\mathcal{H}(\mathcal{A}, \pi_{\mathrm{DP}}^{(\tilde{\pi}_1^{\mathcal{F}_{\mathrm{BB}}}, \tilde{\pi}_2^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}}, \tilde{\pi}_3^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}}, \tilde{\pi}_4^{\mathcal{F}_{\mathrm{CF}}})})) \ ,$$

where $\mathcal{Z}$ is replaced by a machine $\mathcal{Z}'$. The adversary $\mathcal{S}$ simulates all ideal functionalities honestly except $\mathcal{F}_{\mathrm{CF}}$, $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}$, and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}$. The simulation of these functionalities and M_j for $j \notin I_M$ is described below.

Simulation of Links $(\mathcal{Z}, \mathcal{A})$*, and* $(\mathcal{Z}, M_j)$ *for* $j \in I_M$. $\mathcal{S}$ simulates $\mathcal{Z}'$, and $\tilde{M}_j$ for $j \in I_M$, such that it appears as if $\mathcal{Z}$ and $\mathcal{A}$, and $\mathcal{Z}$ and M_j for $j \in I_M$ are linked directly.

1. When $\mathcal{Z}'$ receives m from $\mathcal{A}$, m is written to $\mathcal{Z}$, by $\mathcal{S}$. When $\mathcal{S}$ receives m from $\mathcal{Z}$, m is written to $\mathcal{A}$ by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and $\mathcal{A}$ are linked directly.

2. When $\mathcal{Z}'$ receives m from M_j for $j \in I_M$, m is written to $\mathcal{Z}$ by $\tilde{M}_j$. When $\tilde{M}_j$, $j \in I_M$, receives m from $\mathcal{Z}$, m is written to M_j by $\mathcal{Z}'$. This is equivalent to that $\mathcal{Z}$ and M_j are linked directly for $j \in I_M$.

Extraction from Corrupt Provers. When a corrupt prover M_j, for $j \in I_M$, manages to convince the honest verifiers that it knows a witness, that witness must be forwarded to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$. To do that the witness must be extracted, but this is easy.

The ideal adversary waits until $(M_j, \mathtt{R}, R)$ appears on $\mathcal{F}_{\mathrm{BB}}$. Then it interrupts the simulation of $\mathcal{F}_{\mathrm{BB}}$ and checks if the honest verifiers would accept R as the final message of the prover in Protocol 9.1. If so, it looks at the internal tapes of $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}$ and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}$ to see if w and x have been stored under the tags (M_j, z) and (M_j, y) respectively. If this is the case it instructs $\tilde{M}_j$ to hand $(\mathtt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u_i', v_i')_{i=1}^N), (w, x))$ to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$. Then it waits until it receives $(M_j, \mathtt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u_i', v_i')_{i=1}^N), b)$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, at which point it resumes the simulation of $\mathcal{F}_{\mathrm{BB}}$.

Simulation of Honest Provers. When an honest dummy prover $\tilde{M}_j$ hands a witness to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, the ideal adversary must simulate a proof that convinces the corrupted verifiers. This must be done without knowledge of the witness.

When $\mathcal{S}$ receives $(M_j, \mathtt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u_i', v_i')_{i=1}^N), b)$ from $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, it inputs $(\mathtt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u_i', v_i')_{i=1}^N), (0, 0))$ to M_j if $b = 0$ and it inputs $(\mathtt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u_i', v_i')_{i=1}^N), (1, 1))$ to M_j if $b = 1$, and instructs it to ignore the fact that the witnesses are invalid. It also instructs $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}$ and $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}$ respectively to behave as if the submitted witnesses from M_j are correct when it is handed $(z, 1)$ and $(y, 1)$ respectively.

Then $c \in [2^{\kappa_c - 1}, 2^{\kappa_c} - 1]$ is chosen randomly and the simulator described in the proof of Proposition 9.7, i.e. the simulator of the honest verifier statistical zero-knowledge proof is invoked, to generate C. Then M_j is instructed to use this C, instead of computing it, and $\mathcal{F}_{\mathrm{CF}}$ is instructed to output c. This implies that all verifiers accept the proof.

Reaching a Contradiction. Next we show that if the ideal adversary $\mathcal{S}$ defined above does not imply the security of Protocol 10.4, then we can break the DL-assumption or the strong RSA-assumption.

Suppose that $\mathcal{S}$ does not imply the security of the protocol. Then there exists a hybrid adversary $\mathcal{A}$, an environment $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$, a constant $c > 0$ and an infinite index set $\mathcal{N} \subset \mathbb{N}$ such that for $\kappa \in \mathcal{N}$

$$\begin{aligned}&|\Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}})) = 1] \\ &\quad - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}, \pi_{\mathrm{DP}}^{(\tilde{\pi}_1^{\mathcal{F}_{\mathrm{BB}}}, \tilde{\pi}_2^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(z)}}, \tilde{\pi}_3^{\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{C}}(y)}}, \tilde{\pi}_4^{\mathcal{F}_{\mathrm{CF}}})})) = 1]| \geq \frac{1}{\kappa^c} \ ,\end{aligned}$$

where $\mathcal{S}$ runs $\mathcal{A}$ as a black-box as described above, i.e. $\mathcal{S} = \mathcal{S}(\mathcal{A})$.

Denote by T the machine that simulates $\mathcal{Z}_z(\mathcal{I}(\mathcal{S}(\mathcal{A}'), \tilde{\pi}^{\mathcal{F}_{\mathrm{MN}}}))$, except that instead of simulating honest provers M_j for $j \notin I_M$ as described above, it simply looks at the internal tapes of $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ to extract the message on the form

$$(M_j, \texttt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u'_i, v'_i)_{i=1}^N), (w, x))$$

handed to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$ by $\tilde{M}_j$. It inputs $(\texttt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u'_i, v'_i)_{i=1}^N), (w, x))$ to M_j, which then follows the protocol honestly.

From Proposition 9.7 we know that Protocol 9.1 is statistical zero-knowledge. Thus,

$$|\Pr[T = 1] - \Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}(\mathcal{A}'), \tilde{\pi}^{\mathcal{F}_{\mathrm{MN}}})) = 1]|$$

is negligible. For simplicity we ignore this negligible difference in the remainder of the proof.

The only remaining difference between the simulation carried out by T and an execution in the real model, is that it could happen that $\mathcal{S}$ hands

$$(\texttt{Prover}, (g, z, y, (u_i, v_i)_{i=1}^N, (u'_i, v'_i)_{i=1}^N), (w, x))$$

to $\mathcal{F}_{\mathrm{ZK}}^{R_{\mathrm{DP}}}$, despite that $((g, z, y, (u_i, v_i)_{i=1}^N, (u'_i, v'_i)_{i=1}^N), (w, x)) \notin R_{\mathrm{DP}}$.

Denote by B_l the event that this happens the lth time a proof is carried out. Each mix-servers proves at most one statement.

Since the simulation is statistically close to the real model in all respects except that some event B_l may occur, and we know that the environment can distinguish the hybrid model from the ideal model with notable probability, we conclude that

$$\Pr[B_1 \vee B_2 \vee \cdots \vee B_k] \geq \frac{1}{\kappa^c} \ .$$

An averaging argument implies that $\Pr[B_l] \geq \frac{1}{k\kappa^c}$ for some fixed l.

Next we argue that this contradicts the soundness of Protocol 9.1. Denote by A' the adversary that accepts $(\mathbf{\Gamma}, \overline{g}) = ((\mathbf{N}, \mathbf{g}, \mathbf{h}), (g_i)_{i=1}^N)$ as input and simulates simulates T, except for the following changes. It uses its input, instead of generating these parameters during the simulation of $\mathcal{F}_{\mathrm{RSA}}$ and $\mathcal{F}_{\mathrm{CF}}$. The simulation is continued until the lth proof is about to be executed. Then it waits for an input $s \in [0, 2^\kappa - 1]$ and instructs $\mathcal{F}_{\mathrm{CF}}$ to output s in the generation of the primes, i.e.,

$P_j = \mathsf{PGen}^{\mathsf{c}}(\mathsf{PRG}(s))$. Then it waits for another input $c \in [2^{\kappa_c - 1}, 2^{\kappa_c} - 1]$ and instructs $\mathcal{F}_{\mathrm{CF}}$ to output $c' = c - 2^{\kappa_c - 1}$ in the generation of the challenge. It follows that

$$\Pr_{\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}}}[\mathrm{Acc}_V(\mathrm{view}^V_{A'}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{A'}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \notin L_{R_{\mathrm{DP}}}] \geq \frac{1}{k\kappa^c} \ ,$$

where r_{p} denotes the randomness of all machines in the simulation of T, and r_{v} denotes the randomness of the honest verifier. Recall from Proposition 9.8 that Protocol 9.1 is a computationally convincing proof of knowledge. Proposition 2.32 then implies that $\Pr_{\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}}}[\mathrm{Acc}_V(\mathrm{view}^V_{A'}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{A'}(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}}) \notin L_{R_{\mathrm{DP}}}]$ is negligible. Thus, we have reached a contradiction. □

Chapter 11

An Adaptively Secure Mix-Net Without Erasures

Many mix-net constructions are proposed in the literature, but we are not aware of any construction that is claimed to be secure against an adaptive adversary in any model, even with erasures. Recall that in a model with erasures some parties must erase parts of their computational history to ensure security, whereas in a model without erasures it is assumed that every state transition is stored on a special history tape that is handed to the adversary upon corruption. Adaptive adversaries are more realistic than static adversaries, in particular for protocols that execute over a long period of time such as electronic election protocols. We provide the first mix-net that is secure against an adaptive adversary that corrupts any minority of the mix-servers and any set of senders. We formalize security in the UC-framework without erasures. Our solution is based on the Paillier cryptosystem, but we also consider to what extent our method carries over to El Gamal based mix-nets. As our two solutions with the El Gamal cryptosystem have less desirable properties, we only sketch these. The first is provably secure, but impractical. The second is only secure in a random oracle model with erasures, but it is practical. Our analysis is novel in that we show that a mix-net can be proved UC-secure even if the zero-knowledge proofs of knowledge of correct re-encryption-permutations computed by the mix-servers are not zero-knowledge against adaptive adversaries and not straight-line extractable.

11.1 Adversary Model

The adversarial model considered in this chapter is the same as that considered in Chapter 8, except that the adversary may choose which parties to corrupt adaptively. Thus, the comments in Section 8.1 on alternative adversary models are applicable to this chapter as well.

Definition 11.1 (Adaptive Mix-Net Adversaries). We define $\mathcal{M}_l^*$ to be the set of adaptive adversaries that corrupt less than l out of k parties of the mix-server type, and arbitrarily many parties of the sender type.

11.2 Distributed Paillier

We use a combination of two threshold versions of the Paillier [121] cryptosystem introduced by Lysyanskaya and Peikert [100] and Damgård et al. [57]. We concentrate on the fixed-size cleartext case. A detailed introduction to the Paillier cryptosystem is given in Section 3.12.

The scheme is turned into a distributed cryptosystem with k parties of which k' are needed for decryption in the following way.

The key generator chooses two random generators g and h of a subgroup G_q of prime order q of $\mathbb{Z}^*_{2q+1}$ for a random prime $2q+1$ such that $\log_2 q > 2\kappa + \kappa_r$, where κ_r is an additional security parameter.

We also let v be a generator of the group of squares $\mathrm{QR}_{\mathsf{N}^2} = G_{\mathsf{N}} \times G_{\mathsf{f}}$. Then we assign to each party M_j a random element $\mathsf{d}_j \in [0, 2^{2\kappa+\kappa_r} - 1]$ under the restriction that $\mathsf{d} = \sum_{j=1}^{k} \mathsf{d}_j \bmod \mathsf{Nf}$, and define $\mathsf{v}_j = \mathsf{v}^{\mathsf{d}_j} \mod \mathsf{N}^2$. We also compute a Shamir-secret sharing [136] of each d_j to allow reconstruction of this value. More precisely we choose for each j a random $(k'-1)$-degree polynomial f_j over $\mathbb{Z}_q$ under the restriction that $f_j(0) = \mathsf{d}_j$, and define $\mathsf{d}_{j,l} = f_j(l) \bmod q$. A Pedersen [123] commitment $F_{j,l} = g^{\mathsf{d}_{j,l}} h^{\mathsf{t}_{j,l}}$ of each $\mathsf{d}_{j,l}$ is also computed, where $\mathsf{t}_{j,l} \in \mathbb{Z}_q$ is randomly chosen. The joint public key consists of $(\mathsf{N}, \mathsf{v}, (\mathsf{v}_j)_{j=1}^k, (F_{j,l})_{j,l\in\{1,\ldots,k\}})$. The private key of M_j consists of $(\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k)$.

To jointly decrypt a cryptotext u, the jth share-holder computes $u_j = u^{\mathsf{d}_j} \bmod \mathsf{N}^2$ and proves in zero-knowledge that $\log_u u_j = \log_{\mathsf{v}} \mathsf{v}_j$. If the proof fails, each M_l publishes $(\mathsf{d}_{j,l}, \mathsf{t}_{j,l})$. This allows each honest party to find a set of $(\mathsf{d}_{j,l}, \mathsf{t}_{j,l})$ such that $F_{j,l} = g^{\mathsf{d}_{j,l}} h^{\mathsf{t}_{j,l}}$, recover d_j using Lagrange interpolation, and compute $u_j = u^{\mathsf{d}_j} \bmod \mathsf{N}^2$. Finally, the values are combined to yield the cleartext by $L(\prod_{j=1}^k u_j) = m$.

11.3 Key Generation

The mix-servers generate a joint Paillier public key N. The corresponding private key is verifiably and secretly shared among the mix-servers as described above. In our formulation this is intertwined with the main protocol. The public key N is the main public key in the mix-net, but we do need additional keys.

We need an additional Paillier cryptosystem, but this need not be distributed. We denote by $(\mathsf{N}', \mathsf{g}', \mathsf{d}')$ Paillier parameters generated as above but such that $\mathsf{N}' > \mathsf{N}$. We also need an RSA modulus $\mathbf{N}$ that is chosen exactly as the Paillier moduli N and N'. Finally, we need two Paillier cryptotexts $K_0 = E_{\mathsf{N}}(0, R_0)$ and $K_1 = E_{\mathsf{N}}(1, R_1)$ of 0 and 1 respectively. Below we summarize key generation as an ideal functionality.

Functionality 11.2 (Key Generation). The ideal key generation functionality, $\mathcal{F}_{\mathrm{PKG}}$, running with mix-servers $M_1, \ldots, M_k$, senders $S_1, \ldots, S_N$, and ideal adversary $\mathcal{S}$ proceeds as follows. It generates keys as described above and hands $((\mathcal{S}, \mathtt{PublicKeys}, (\mathsf{N}, g, h, \mathsf{N}', \mathsf{N}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}}))$, $\{(M_j, \mathtt{Keys}, (\mathsf{N}, g, h, \mathsf{N}', \mathsf{N}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}}),$ $(\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k))\}_{j=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$.

Recall that $\mathcal{C}_{\mathcal{I}}$ acts as a router between the parties and the ideal functionality. The ideal functionality can be securely realized using general methods [44] in the common random string model.

11.4 The Adaptively Secure Mix-Net

In this section we first describe the basic structure of our mix-net. Then we explain how we modify this to accommodate adaptive adversaries. We also discuss how and why our construction differs from previous constructions in the literature. This is followed by subsections introducing the subprotocols invoked in an execution of the mix-net. Finally, we give a detailed description of the mix-net.

11.4.1 The Overall Structure

Our mix-net is based on the re-encryption-permutation paradigm, which is explained as follows. Let $L_0 = \{u_{0,i}\}_{i=1}^N$ be the list of cryptotexts submitted by senders. For $l = 1, \ldots, k$ the lth mix-server M_l re-encrypts each element in $L_{l-1} = \{u_{l-1,i}\}_{i=1}^N$ as explained in Section 3.12, sorts the resulting list and publishes the result as L_l. Then it proves, in zero-knowledge, knowledge of a witness that L_{l-1} and L_l are related in this way. The mix-servers then jointly and verifiably re-encrypt the cryptotexts in L_k. Note that no permutation takes place in latter step. The result is denoted by L_{k+1}. Finally, the mix-servers jointly and verifiably decrypt each cryptotext in L_{k+1} and sort the resulting list of cleartexts to form the output. In short, the overall structure is similar to previous constructions with the exception of the joint re-encryption step, which is needed for technical reasons discussed below.

To avoid relation attacks, where a sender manages to submit a cryptotext of a message that is related to the encrypted message of another sender, each sender proves knowledge of the message it submits. Each sender forms two cryptotexts $u_{0,i}$ and $u'_{0,i}$ using different public keys and proves that they hold the same cleartext. Naor and Yung's [110] double-cryptotext trick then allows straight-line extraction of all cleartexts submitted by corrupted parties.

Correctness should hold due to the proofs computed by the mix-servers. Anonymity for the senders should hold due to the semantic security and homomorphic property of Paillier's cryptosystem, and the fact that all proofs are zero-knowledge.

11.4.2 Accommodating Adaptive Adversaries

In this section we point at some key modifications needed for our mix-net to be secure against *adaptive* adversaries, and briefly explain how they are used by the ideal adversary.

As in the static case considered in [147] and in Chapter 8 the ideal adversary must simulate the submissions of honest senders without knowing which message they actually hand to $\mathcal{F}_{\text{RMN}}$. The first new problem encountered when considering adaptive adversaries is that the adversary may decide to corrupt a simulated honest sender S_i that has already computed fake cryptotexts $u_{0,i}$ and $u'_{0,i}$. The ideal adversary can of course corrupt the corresponding dummy party $\tilde{S}_i$ and retrieve the true value m_i it handed to $\mathcal{F}_{\text{RMN}}$ as in the static case. The problem is that it must provide S_i with a plausible history tape that convinces the adversary that S_i sent m_i already from the beginning. To solve this problem we use an idea of Damgård and Nielsen [58]. We have two public keys $K_0 = E_{\mathsf{N}}(0, R_0) = R_0^{2\mathsf{N}} \bmod \mathsf{N}^2$ and $K_1 = E_{\mathsf{N}}(1, R_1) = \mathsf{g}R_1^{2\mathsf{N}} \bmod \mathsf{N}^2$ and each sender is given a unique key $K'_i = E_{\mathsf{N}'}(a_i)$ for a randomly chosen $a_i \in \mathbb{Z}_{\mathsf{N}'}$. The sender of a message m_i chooses $b_i \in \mathbb{Z}_{\mathsf{N}}$, $r_i \in \mathbb{Z}^*_{\mathsf{N}}$ and $r'_i \in \mathbb{Z}^*_{\mathsf{N}'}$ randomly, and computes its two cryptotexts as follows

$$u_i = E_{K_1,\mathsf{N}}(m_i, r_i) \quad \text{and} \quad u'_i = E_{\mathsf{g}^{b_i}K'_i,\mathsf{N}'}(m_i, r'_i) \ .$$

Then it submits (b_i, u_i, u'_i) and proves in zero-knowledge that the same message m_i is encrypted in both cryptotexts. Note that $D_{\mathsf{d}}(u_i) = m_i$ and $D_{\mathsf{d}'}(u'_i) = (a_i + b_i)m_i$ due to the homomorphic property of the cryptosystem.

During simulation we instead define $K_0 = E_{\mathsf{N}}(1, R_0) = \mathsf{g}R_0^{2\mathsf{N}} \bmod \mathsf{N}^2$ and $K_1 = E_{\mathsf{N}}(0, R_1) = R_1^{2\mathsf{N}} \bmod \mathsf{N}^2$. This means that u_i becomes an encryption of 0 for all senders. Furthermore, simulated senders choose $b_i = -a_i \bmod \mathsf{N}'$ which implies that also u'_i is an encryption of 0. The important property of the simulation is that given m_i and R_1 we can define $\bar{r}_i = r_i/R_1^{m_i}$ such that $u_i = E_{K_1,\mathsf{N}}(m_i, \bar{r}_i)$, i.e., we can open a simulated cryptotext as an encryption of an arbitrary message m_i. The cryptotext u'_i can also be opened as an encryption of m_i in a similar way when $b_i + a_i = 0 \bmod \mathsf{N}'$. Finally, it is possible to construct the proof of equality such that it also can be opened in a convincing way. This allows the simulator to simulate honest senders and produce plausible history tapes as required.

Corrupt senders on the other hand have negligible probability of guessing a_i, so the simulator can extract the message submitted by corrupt senders using only the private key d' by computing $m_i = D_{\mathsf{d}'}(u'_i)/(a_i + b_i) \bmod \mathsf{N}'$ as b_i is sent explicitly.

Before the mix-net simulated by the ideal adversary starts to process the input cryptotexts the ideal mix-net $\mathcal{F}_{\text{RMN}}$ has handed the ideal adversary the list of cleartexts that should be output by the simulation. All cleartexts equal zero in the simulation and must be replaced. In the static case considered in [147] and in Chapter 8 this is done by one of the honest simulated mix-servers M_j when it re-encrypts and permutes L_j. Then it invokes the simulator instead of executing its proof of knowledge. With an adaptive adversary this is no longer possible. The

problem is that the adversary may corrupt M_j after it has output L_j, and it is then impossible to compute a plausible history tape for M_j.

Instead, the correct messages are introduced in the joint re-encryption phase. Thus, all mix-servers are simulated honestly during the re-encryption-permutation phase and the decryption phase is also simulated honestly using the private key d.

The joint re-encryption is defined as follows. Before the start of the mixing each mix-server is given a random cryptotext $\bar{K}'_j$ using the public key N'. Each mix-server M_j chooses random elements $m_{j,i} \in \mathbb{Z}_{\mathsf{N}}$ and commits to these by choosing $\bar{b}_j \in \mathbb{Z}_{\mathsf{N}'}$ and $s'_{j,i} \in \mathbb{Z}^*_{\mathsf{N}'}$ randomly and computing $w'_{j,i} = E_{\mathsf{g}^{\bar{b}_j}\bar{K}'_j,\mathsf{N}'}(m_{j,i}, s'_{j,i})$. When all mix-servers have published their commitments, it chooses $s_{j,i} \in \mathbb{Z}^*_{\mathsf{N}}$ randomly and computes $w_{j,i} = E_{K_0,\mathsf{N}}(m_{j,i}, s_{j,i})$. It also proves in zero-knowledge that the same random element $m_{j,i}$ is encrypted in both cryptotexts. The jointly re-encrypted elements $u_{k+1,i}$ are then formed as

$$u_{k+1,i} = u_{k,i} \prod_{l \in I} w_{l,i}^{\prod_{l' \neq l} \frac{l'}{l'-l}} \quad ,$$

where I is the lexicographically first set of k' indices j such that the proof of M_j is valid.

In the real execution this is an elaborate way to re-encrypt $u_{k,i}$, since K_0 is an encryption of 0. In the simulation on the other hand the ideal adversary chooses $\bar{b}_j = -\bar{a}_j \bmod \mathsf{N}'$ and sets $m_{j,i} = 0$ for simulated mix-servers and extracts the $m_{j,i}$ values of corrupt mix-servers from their commitments. It then redefines the $m_{j,i}$ values of simulated honest mix-servers such that $f_i(j) = m_{j,i}$ for a $(k'-1)$-degree polynomial f_i over $\mathbb{Z}_{\mathsf{N}}$ such that $f_i(0)$ equals $m_{\pi(i)}$ for some random permutation $\pi \in \Sigma_N$. Since $\bar{b}_j + \bar{a}_j = 0 \bmod \mathsf{N}'$ it can compute $\bar{s}'_{j,i}$ such that $w'_{j,i} = E_{\mathsf{g}^{\bar{b}_j}\bar{K}'_j,\mathsf{N}'}(m_{j,i}, \bar{s}'_{j,i})$. In the simulation K_0 is an encryption of 1 and each $u_{k,i}$ is an encryption of zero, which implies that $u_{k+1,i}$ becomes an encryption of $m_{\pi(i)}$ as required. Intuitively, the adversary can not tell the difference since it can never corrupt k' mix-servers and get its hands on their $m_{j,i}$ values directly, and the semantic security of the cryptosystem prevents it from knowing these values otherwise.

11.4.3 Some Intuition Behind Our Analysis

The soundness of the proof of equal exponents used during decryption, and the proof of knowledge of correct re-encryption-permutation ensures that the output messages are identical in the ideal model and the real execution.

It may seem that the zero-knowledge simulator of the proof of equal exponents used during decryption and the proof of knowledge of correct re-encryption-permutation respectively play no role in the protocol, since they are not invoked by the ideal adversary sketched above. It may also appear that the knowledge extraction property of the proof of knowledge of a correct re-encryption-permutation is never exploited, but they play an essential role in the analysis of the ideal adversary.

To see this note that the private key d corresponding to the Paillier modulus N is needed both in the ideal model and in the real protocol execution. Thus, even if the adversary can distinguish the ideal model from the execution of the protocol, we can not use the adversary directly to reach a contradiction to the semantic security of Paillier.

To reach a contradiction we must first replace the ideal model and the real protocol execution by simulations that do not use the private key, but that are indistinguishable from the ideal model and the real execution respectively. Then we use the two simulations to break the semantic security of the Paillier cryptosystem. To do this we use the single-honest-player proof strategy, namely we guess the identity j of a mix-server M_j that the adversary will not corrupt. Our guess is correct with probability at least $1/2$, since the majority of the mix-servers are honest and our guess is independent of all other variables in the simulation. Then we instruct M_j to simulate its proof of knowledge of correct re-encryption-permutation and its proof of correct decryption. Since we know the set of messages that should be output by the protocol, and M_j simulates its proof, we can instruct M_j to output his parts of the decryption procedure without using his share of the private key d. This allows us to hand simulated shares of the private key to all other mix-servers.

The only remaining problem is that although we have instructed M_j to output the correct set of messages, we have no guarantees that they appear in the correct order. Indeed if the last mix-server M_k is corrupted and we do not have d, the semantic security of Paillier prohibits us from knowing the correct order. Thus, decryption is not simulated correctly. To solve this problem we exploit the knowledge extractor of the proof of knowledge of correct re-encryption-permutation for each corrupted mix-server M_j. This allows us to extract in which order the messages should be output and M_j can simulate decryption correctly. The reason we need the zero-knowledge simulator of the proof of knowledge of correct re-encryption-permutation is intuitively obvious, but to describe precisely the role it plays in the security analysis is difficult, and we can only refer the reader to the proof in Section 11.7.

11.4.4 Differences with Previous Constructions

Our mix-net differs from the two previously considered statically UC-secure mix-nets, [147] and Chapter 8 in several ways.

Our construction is based on the Paillier cryptosystem, whereas the previous are based on the El Gamal cryptosystem. We need to use the Paillier cryptosystem to allow adaptive corruption of the senders in the way explained above.

In previous work each mix-server partially decrypts its input. Thus, L_k is already the list of cleartexts, which reduces the number of rounds and the communication complexity of the protocol. It also simplifies the analysis compared to having a joint decryption step. We use the more cumbersome paradigm outlined above, since we are not aware of any method to partially decrypt Paillier cryptotexts securely.

We have a joint re-encryption step which no previous construction has. This is needed since we are not aware of any other way to insert the correct messages in an adaptively secure way in the simulation.

The analyses of the mix-nets in [147] and in Chapter 8 are modular. This means that the mix-net is given in a hybrid model with access to ideal zero-knowledge proof of knowledge functionalities. These functionalities are then securely realized, and the composition theorem of the UC-framework invoked. The modular approach simplifies the analysis considerably, but the strong demands on subprotocols make them hard to securely realize efficiently. In particular, the proof of knowledge of a correct re-encryption-permutation is difficult to realize, since the size of the witness is large. We remark that this problem is avoided in Chapter 8 by a trick specific to the El Gamal cryptosystem. It may seem that it would suffice with a proof of correctness instead of a proof of knowledge, but as explained in Chapter 8, if a mix-net is based on the re-encryption-permutation paradigm, then each mix-server must prove knowledge of a witness of a correct re-encryption-permutation. In other words a "proof of correctness" or a "proof of membership" [116] in the UC-framework is not enough. In the adaptive setting the problem becomes even harder since any realization must be secure with regards to an adaptive adversary.

We avoid this problem by showing that a zero-knowledge proof of knowledge of correct re-encryption-permutation in the classical sense is sufficient, i.e., the protocol can *not be simulated to an adaptive adversary* and extraction is *not straight-line*.

11.5 Subprotocols Invoked by the Main Protocol

Before we give the details of the main protocol, we detail the two subprotocols that are needed in addition to the zero-knowledge proof of knowledge of correct re-encryption of Section 9.7.

11.5.1 Proof of Knowledge of Re-encryption-Permutation

Denote by $\pi_{\text{prp}} = (P_{\text{prp}}, V_{\text{prp}})$ the 5-move protocol for proving knowledge of a witness of a re-encryption and permutation of a list of Paillier cryptotexts we construct in Section 9.7. We assume for simplicity that the protocol invokes PGen during generation of the prime vectors. Both the prover and the verifier accepts as special parameters an RSA-modulus $\mathbf{N}$ and random $\mathbf{g}, \mathbf{h} \in \text{QR}_{\mathbf{N}}$, and random $g_1, \ldots, g_N \in G_q$. For convenience we restate the definition of the re-encryption-permutation relation $R_{\text{RP}}^{\mathsf{N}}$ and the security properties of π_{prp} below.

Definition 9.20 (Knowledge of Correct Re-encryption-Permutation). Define for each N and N a relation $R_{\text{RP}}^{\mathsf{N}} \subset ((\mathbb{Z}_{\mathsf{N}}^*)^N \times (\mathbb{Z}_{\mathsf{N}}^*)^N) \times [0, 2^{\kappa+\kappa_r} - 1]^N$, by

$$((\{u_i\}_{i=1}^N, \{u_i'\}_{i=1}^N), (a, (x_i)_{i=1}^N)) \in R_{\text{RP}}^{\mathsf{N}}$$

precisely when $a < \sqrt{\mathsf{N}}/4$ equals one or is prime and $(u'_i)^a = \mathsf{g}_\mathsf{f}^{x_{\pi(i)}} u_{\pi(i)}^a \mod \mathsf{N}^2$ for $i = 1, \ldots, N$ and some permutation $\pi \in \Sigma_N$ such that the list $\{\bar{u}_i\}_{i=1}^N$ is sorted lexicographically.

Proposition 9.22. *The protocol π_{prp} is honest verifier statistical zero-knowledge.*

Proposition 9.23. *The protocol π_{prp} is a computationally convincing proof of knowledge for the relation $R_{\mathrm{RP}}^{\mathsf{N}}$ with regards to the distribution of $(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(g_1, \ldots, g_N)$.*

Proposition 9.24 (Simplified). *Protocol 9.21 has overwhelming completeness.*

11.5.2 Proof of Equality of Cleartexts

When a sender submits its cryptotexts u_i and u'_i it must prove that they are encryptions of the same message under two distinct public keys. The protocol $\pi_{eq} = (P_{\mathrm{eq}}, V_{\mathrm{eq}})$ used to do this is given below. The security parameters κ_c and κ_r decide the soundness and statistical zero-knowledge property of the protocol.

Protocol 11.3 (Proof of Equal Cleartexts Using Distinct Moduli).
COMMON INPUT: $\mathsf{N} \in \mathbb{Z}$, $K, u \in \mathbb{Z}^*_{\mathsf{N}^2}$, $\mathsf{N}' \in \mathbb{Z}$, $K', u' \in \mathbb{Z}^*_{\mathsf{N}'^2}$, $\mathbf{N} \in \mathbb{N}$, generators $\mathbf{g}$ and $\mathbf{h}$ of $\mathrm{QR}_{\mathbf{N}}$.
PRIVATE INPUT: $m \in \{0,1\}^{\kappa - \kappa_r}$, $r \in \mathbb{Z}^*_{\mathsf{N}}$, and $r' \in \mathbb{Z}^*_{\mathsf{N}'}$ such that $u = E_{K,\mathsf{N}}(m, r)$ and $u' = E_{K',\mathsf{N}'}(m, r')$.

1. The prover chooses $r'' \in [0, 2^{2\kappa+\kappa_r} - 1]$, $s_0 \in \mathbb{Z}^*_{\mathsf{N}^2}$ and $s_1 \in \mathbb{Z}^*_{(\mathsf{N}')^2}$, and $t \in [0, 2^{\kappa+\kappa_c} - 1]$ and $s_2, \in [0, 2^{2\kappa+\kappa_c+2\kappa_r} - 1]$ randomly. Then it computes

$$\begin{aligned} C &= \mathbf{g}^m \mathbf{h}^{r''} \bmod \mathbf{N} \ , \text{ and} \\ (\alpha_0, \alpha_1, \alpha_2) &= (K^t s_0^{2\mathsf{N}} \bmod \mathsf{N}^2, (K')^t s_1^{2\mathsf{N}'} \bmod (\mathsf{N}')^2, \mathbf{g}^t \mathbf{h}^{s_2} \bmod \mathbf{N}) \ , \end{aligned}$$

   and hands $(C, \alpha_1, \alpha_2, \alpha_3)$ to the verifier.

2. The verifier chooses $c \in [2^{\kappa_c - 1}, 2^{\kappa_c} - 1]$ and hands c to the prover.

3. The prover computes

$$\begin{aligned} (e_0, e_1) &= (r^c s_0 \bmod \mathsf{N}, (r')^c s_1 \bmod \mathsf{N}') \ , \text{ and} \\ (e_2, e_3) &= (cr'' + s_2 \bmod 2^{2\kappa+\kappa_c+2\kappa_r}, cm + t \bmod 2^{\kappa+\kappa_c}) \ , \end{aligned}$$

   and hands (e_0, e_1, e_2, e_3) to the verifier.

4. The verifier checks that

$$\begin{aligned} (u^c \alpha_0, (u')^c \alpha_1) &= (K^{e_3} e_0^{2\mathsf{N}} \bmod \mathsf{N}, (K')^{e_3} e_1^{2\mathsf{N}'} \bmod \mathsf{N}') \\ C^c \alpha_2 &= \mathbf{g}^{e_3} \mathbf{h}^{e_2} \bmod \mathbf{N} \ . \end{aligned}$$

Although the protocol is statistical zero-knowledge this is not the property we need in the simulation. Recall that the simulator must be able to compute a plausible history tape if a simulated sender is corrupted.

Damgård and Nielsen [58] introduce the notion of non-erasure zero-knowledge proofs of knowledge. Our protocol could be said to be a "non-erasure statistical zero-knowledge computationally convincing proof", but we do not define this notion explicitly. Instead we prove the following propositions.

Proposition 11.4 ("Zero-Knowledge"). *Let* $K = R^{2\mathsf{N}} \bmod \mathsf{N}^2$ *and* $K' = R'^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$ *for some* $R \in \mathbb{Z}^*_{\mathsf{N}}$ *and* $R' \in \mathbb{Z}^*_{\mathsf{N}'}$. *Let* $\mathbf{h}$ *be a generator of* $\mathrm{QR}_{\mathbf{N}}$ *and* $\mathbf{g} = \mathbf{h}^{\mathbf{x}}$.

Let r, r', *and* (r'', s_0, s_1, t, s_2) *be randomly distributed in the domains described in the protocol, and denote by* $I(m) = (\mathsf{N}, K, u, \mathsf{N}', K', u', \mathbf{N}, \mathbf{g}, \mathbf{h})$ *the common input corresponding to the private input* (m, r, r'). *Denote by* c *the random challenge from the verifier and let* $T(m) = (\alpha, c, e)$ *be the proof transcript induced by* (m, r, r'), c, *and* (r'', s_0, s_1, t, s_2).

There exists a deterministic polynomial-time algorithm His *such that for every* $m \in \{0,1\}^{\kappa - \kappa_r}$ *it holds that if we define*

$$(\bar{r}, \bar{r}', \bar{r}'', \bar{s}_0, \bar{s}_1, \bar{t}, \bar{s}_2) = \mathsf{His}(R, R', \mathbf{x}, m, r, r', r'', s_0, s_1, t, s_2, c)$$

then the distributions of

$$[I(m), T(m), (m, r, r'), (r'', s_0, s_1, t, s_2)] \quad \textit{and}$$
$$[I(0), T(0), (m, \bar{r}, \bar{r}'), (\bar{r}'', \bar{s}_0, \bar{s}_1, \bar{t}, \bar{s}_2))]$$

are statistically close.

Note that the proposition in itself does not imply that the protocol is zero-knowledge. The protocol is only zero-knowledge for common inputs on a certain form, namely if K and K' are both encryptions of zero.

Proof of Proposition 11.4. The algorithm His first defines

$$\begin{aligned} (\bar{r}, \bar{r}') &= (rR^{-m} \bmod \mathsf{N},\ r'(R')^{-m} \bmod \mathsf{N}) \ , \quad \text{and} \\ \bar{r}'' &= r'' - \mathbf{x}m \bmod 2^{\kappa+\kappa_c+2\kappa_r} \ . \end{aligned}$$

Then it defines

$$\begin{aligned} (\bar{s}_0, \bar{s}_1) &= (e_0/\bar{r}^c \bmod \mathsf{N},\ e_1/(\bar{r}')^c \bmod \mathsf{N}) \ , \\ \bar{s}_2 &= e_2 - c\bar{r}'' \bmod 2^{\kappa+\kappa_c+2\kappa_r} \ , \quad \text{and} \\ \bar{t} &= e_3 - cm \bmod 2^{\kappa+\kappa_c} \ . \end{aligned}$$

It is easy to see that the distribution of the resulting elements is statistically close to the distribution of the corresponding elements in a real execution of the proof.

To see why the elements are defined this way, note that

$$\begin{aligned}
u &= r^{2\mathsf{N}} = (\bar{r}R^m)^{2\mathsf{N}} = \bar{r}^{2\mathsf{N}}K^m \bmod \mathsf{N}^2 \\
u' &= (r')^{2\mathsf{N}} = (\bar{r}'R^m)^{2\mathsf{N}} = (\bar{r}')^{2\mathsf{N}}K^m \bmod (\mathsf{N}')^2 \\
C &= \mathbf{h}^{r''} = \mathbf{h}^{\bar{r}''+\mathbf{x}m} = \mathbf{g}^m\mathbf{h}^{\bar{r}''} \bmod \mathbf{N} \\
\alpha_0 &= s_0^{2\mathsf{N}}K^t = (e_0/r^c)^{2\mathsf{N}}K^{e_3} = \bar{s}_0^{2\mathsf{N}}(\bar{r}^c/r^c)^{2\mathsf{N}}K^{\bar{t}+cm} \\
&= \bar{s}_0^{2\mathsf{N}}(r^cR^{-cm}/r^c)^{2\mathsf{N}}K^{\bar{t}+cm} = \bar{s}_0^{2\mathsf{N}}K^{\bar{t}} \bmod \mathsf{N}^2 \\
\alpha_1 &= s_1^{2\mathsf{N}'}(K')^t = (e_1/(r')^c)^{2\mathsf{N}'}(K')^{e_3} = \bar{s}_1^{2\mathsf{N}'}((\bar{r}')^c/(r')^c)^{2\mathsf{N}'}(K')^{\bar{t}+cm} \\
&= \bar{s}_1^{2\mathsf{N}}((r')^cR'^{-cm}/(r')^c)^{2\mathsf{N}'}(K')^{\bar{t}+cm} = \bar{s}_1^{2\mathsf{N}'}(K')^{\bar{t}} \bmod (\mathsf{N}')^2 \\
\alpha_2 &= \mathbf{g}^t\mathbf{h}^{s_2} = \mathbf{g}^{e_3}\mathbf{h}^{e_2-cr''} = \mathbf{g}^{\bar{t}+cm}\mathbf{h}^{e_2-c\bar{r}''-\mathbf{x}cm} = \mathbf{g}^{\bar{t}}\mathbf{h}^{\bar{s}_2} \ .
\end{aligned}$$

□

The protocol must have overwhelming completeness and the adversary must have negligible probability of proving a false statement. Denote by $R_{\mathrm{EC},\mathsf{N},\mathsf{N}'} \subset \mathbb{Z}^*_{\mathsf{N}^2} \times \mathbb{Z}^*_{\mathsf{N}'^2}$ the relation defined as the set of $(u,u') \in R_{\mathrm{EC},\mathsf{N},\mathsf{N}'}$ such that $u = E_{\mathsf{N}}(m,r)$ and $u' = E_{\mathsf{N}'}(m,r')$ for some $m \in \{0,1\}^{\kappa-\kappa_r}$, $r \in \mathbb{Z}^*_{\mathsf{N}}$, and $r' \in \mathbb{Z}^*_{\mathsf{N}'}$.

The adversary is given as input Paillier parameters (N,N'), RSA-parameters $\mathbf{\Gamma} = (\mathbf{N},\mathbf{g},\mathbf{h})$ and an internal random string r_{p} as input and outputs an instance $I_{P^*_{\mathrm{eq}}}(\mathsf{N},\mathsf{N}',\mathbf{\Gamma},r_{\mathrm{p}})$, i.e., some pair $(u,u') \in \mathbb{Z}^*_{\mathsf{N}^2} \times \mathbb{Z}^*_{\mathsf{N}'^2}$. Then it executes the protocol above with an honest verifier. As in the previous section we denote by $\delta_{P^*_{\mathrm{eq}}}(\mathsf{N},\mathsf{N}',\mathbf{\Gamma},r_{\mathrm{p}})$ the probability over the random choices of the honest verifier that it accepts the instance $I_{P^*_{\mathrm{eq}}}(\mathsf{N},\mathsf{N}',\mathbf{\Gamma},r_{\mathrm{p}})$.

Proposition 11.5 (Completeness). *The protocol has overwhelming completeness.*

Proof. The protocol may fail if and only if a modular reduction is performed in the computation of e_2 or e_3 in Step 3. This happens with negligible probability over the random choice of t and s_2. □

Proposition 11.6 (Soundness). *The protocol is a computationally convincing proof with regards to the distribution of* $(\mathbf{N},\mathbf{g},\mathbf{h})$.

Proof. Suppose that we are given two accepting transcripts

$$((C,\alpha_0,\alpha_1,\alpha_2),c,(e_0,e_1,e_2,e_3)) \ , \quad \text{and} \quad ((C,\alpha_0,\alpha_1,\alpha_2),c',(e'_0,e'_1,e'_2,e'_3))$$

such that $c \neq c'$. From the transcripts we have

$$\begin{aligned}
C^{c-c'} &= \mathbf{g}^{e_3-e'_3}\mathbf{h}^{e_2-e'_2} \bmod \mathbf{N} \\
u^{c-c'} &= K^{e_3-e'_3}(e_0/e'_0)^{2\mathsf{N}} \bmod \mathsf{N}^2 \\
(u')^{c-c'} &= (K')^{e_3-e'_3}(e_1/e'_1)^{2\mathsf{N}'} \bmod (\mathsf{N}')^2 \ .
\end{aligned}$$

First note that $c - c'$ is invertible modulo $|\mathrm{QR}_{\mathsf{N}^2}|$ and $|\mathrm{QR}_{(\mathsf{N}')^2}|$. Thus, there exists r and r' such that $r^{c-c'} = e_0/e_0' \bmod \mathsf{N}$ and $(r')^{c-c'} = e_1/e_1' \bmod \mathsf{N}$. If $(c - c')$ divides $(e_3 - e_3')$ over $\mathbb{Z}$ we define $m = (e_3 - e_3')/(c - c')$ and have

$$u = K^m r^{2\mathsf{N}} \bmod \mathsf{N}^2 \quad \text{and} \quad u' = (K')^m (r')^{2\mathsf{N}'} \bmod (\mathsf{N}')^2 \ .$$

By definition m contains less bits than N and N'. Thus, $I_{P^*_{\mathrm{eq}}}(\mathsf{N}, \mathsf{N}', \mathbf{\Gamma}, r_\mathrm{p}) \in R_{\mathrm{EC},\mathsf{N},\mathsf{N}'}$.

If on the other hand $c - c'$ does not divide $e_3 - e_3'$ over $\mathbb{Z}$, then we have found an element $\mathbf{b} \in \mathbb{Z}^*_{\mathbf{N}}$ and integers $\eta_0 \neq 0$, η_1, and η_2, where η_0 does not divide both η_1 and η_2, such that $\mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{h}^{\eta_2}$.

We conclude that the protocol is special-sound according to Definition 2.36. Thus, by Lemma 2.40 it is a proof of knowledge so there exists an extractor $\mathcal{X}$. Unfortunately it is not a proof of knowledge for the relation $R_{\mathrm{EC},\mathsf{N},\mathsf{N}'}$, but for the relation $R_{\mathrm{EC},\mathsf{N},\mathsf{N}'} \vee R_{\mathrm{SRSA}}$. We must show that $\mathcal{X}$ is an extractor for the relation $R_{\mathrm{EC},\mathsf{N},\mathsf{N}'}$.

The first property of the extractor $\mathcal{X}$ required by Definition 2.28 is satisfied trivially. Suppose that the second property does not hold. Then there exists a prover P^*, a constant c, and an infinite index set $\mathcal{N}$ such that $\Pr_{\mathbf{\Gamma},r_\mathrm{p}}[\delta_{P^*,V}(\mathbf{\Gamma}, r_\mathrm{p}) \geq \kappa^{-c}] \geq \kappa^{-c}$ and

$$\Pr[(I_{P^*}(\mathbf{\Gamma}, r_\mathrm{p}), \mathcal{X}^{P^*}(\mathbf{\Gamma}, r_\mathrm{p})) \notin R_{\mathrm{EC},\mathsf{N},\mathsf{N}'} \mid \delta_{P^*,V}(\mathbf{\Gamma}, r_\mathrm{p}) \geq \kappa^{-c}] \geq \kappa^{-c} \ ,$$

for $\kappa \in \mathcal{N}$. Note that no generality is lost by using the same constant c to bound both probabilities. On the other hand we know that

$$\Pr[(I_{P^*}(\mathbf{\Gamma}, r_\mathrm{p}), \mathcal{X}^{P^*}(\mathbf{\Gamma}, r_\mathrm{p})) \in R_{\mathrm{EC},\mathsf{N},\mathsf{N}'} \vee R_{\mathrm{SRSA}} \mid \delta_{P^*,V}(\mathbf{\Gamma}, r_\mathrm{p}) \geq \kappa^{-c}]$$

is overwhelming. Thus, we conclude that

$$\Pr[(I_{P^*}(\mathbf{\Gamma}, r_\mathrm{p}), \mathcal{X}^{P^*}(\mathbf{\Gamma}, r_\mathrm{p})) \in R_{\mathrm{SRSA}} \mid \delta_{P^*,V}(\mathbf{\Gamma}, r_\mathrm{p}) \geq \kappa^{-c}] \geq \frac{1}{2\kappa^c} \ ,$$

and we have $\Pr[(I_{P^*}(\kappa, \mathbf{\Gamma}, r_\mathrm{p}), \mathcal{X}^{P^*}(\mathbf{\Gamma}, r_\mathrm{p})) \in R_{\mathrm{SRSA}}] \geq \frac{1}{2\kappa^{2c}}$. This contradicts the variant strong RSA-assumption, i.e., Lemma 3.12. Thus, the protocol is a computationally convincing proof of knowledge with regards to the distribution of $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$, and we conclude from Proposition 2.32 that it is a computationally convincing proof with regards to the distribution of $\mathbf{\Gamma}$. □

It is easy to see that multiple instances of the protocol can be run in parallel using the same RSA-parameters and same challenge without any decrease in the security. Thus, we use the protocol also for common inputs on the form $(\mathsf{N}, K, \{u_i\}_{i=1}^N, \mathsf{N}', K', \{u_i'\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ and with corresponding private input $(\{m_i\}_{i=1}^N, \{r_i\}_{i=1}^N, \{r_i'\}_{i=1}^N)$.

Remark 11.7. The reason we use the relaxed definition of an ideal mix-net, Functionality 7.2, is that although the protocol is sound it does not imply that the message m is contained in $\{0,1\}^{\kappa-\kappa_r}$. The problem with this is that an adversary may sacrifice the zero-knowledge property of its proof and submit a message that has say $\kappa - \frac{1}{2}\kappa_r$ bits. It seems that the only way to avoid this problem is to use an interval proof of Boudot and Traoré [30]. However, we need the protocol to have a similar "zero-knowledge" property as that above to allow the simulator to produce plausible history tapes. Since the analysis in [30] does not give an algorithm that produces plausible history tapes, and the relaxation of the definition of an ideal mix-net seems reasonable, we do not follow this path.

11.5.3 Proof of Equality of Exponents

During joint decryption of a cryptotext u each mix-server computes $u^{\mathsf{d}_j} \bmod \mathsf{N}^2$ using its part d_j of the private key, and proves correctness relative $\mathsf{v}_j = \mathsf{v}^{\mathsf{d}_j} \bmod \mathsf{N}^2$. The protocol $\pi_{\text{exp}} = (P_{\text{exp}}, V_{\text{exp}})$ below and the propositions are taken from [57].

Protocol 11.8 (Proof of Equal Exponents).
COMMON INPUT: $\mathsf{N} \in \mathbb{Z}$, $u, u', v, v' \in \text{QR}_{\mathsf{N}^2}$.
PRIVATE INPUT: $d \in [0, 2^{2\kappa+\kappa_r} - 1]$ such that $u' = u^d$ and $v' = v^d$.

1. The prover chooses $r \in [0, 2^{2\kappa+\kappa_c+2\kappa_r} - 1]$ randomly, computes $\alpha_0 = u^r$ and $\alpha_1 = v^r$, and hands (α_0, α_1) to the verifier.
2. The verifier chooses $c \in [0, 2^{\kappa_c} - 1]$ and hands c to the prover.
3. The prover computes $e = cd + r \bmod 2^{2\kappa+\kappa_c+2\kappa_r}$ and hands e to the verifier.
4. The verifier checks that $(u')^c\alpha_0 = u^e \bmod \mathsf{N}^2$ and $(v')^c\alpha_1 = v^e \bmod \mathsf{N}^2$.

Proposition 11.9 (Zero-Knowledge). *The protocol is honest verifier statistical zero-knowledge.*

Proof. The zero-knowledge simulator chooses $e \in [0, 2^{2\kappa+\kappa_c+\kappa_r} - 1]$ and $c \in [0, 2^{\kappa_c} - 1]$ randomly and defines $\alpha_0 = u^d/(u')^c \bmod \mathsf{N}^2$ and $\alpha_1 = v^d/(v')^c \bmod \mathsf{N}^2$. It follows that the distribution of the simulated view is statistically close to that in the real protocol. □

Proposition 11.10 (Proof). *The protocol is a proof with overwhelming completeness.*

Proof. Completeness follows since the verifier accepts as long as there is no reduction in the computation of e and this happens with exponentially small probability.

Given two accepting transcripts (α, β, c, e) and (α, β, c', e') such that $c \neq c'$ we have

$$(u')^{c-c'} = u^{e-e'} \quad \text{and} \quad (v')^{c-c'} = v^{e-e'} \ .$$

Our choice of parameters ensure that $c - c'$ is smaller than all divisors of the order Nf of the group $\mathrm{QR}_{\mathsf{N}^2}$ of squares modulo N^2. Thus, $c - c'$ is invertible modulo Nf and we may define $d = (e - e')/(c - c') \bmod \mathsf{Nf}$ with $u' = u^d$ and $v' = v^d$. Thus, $((c - c'), (e - e'))$ is a witness.

We conclude that the protocol is special sound according to Definition 2.36. Thus, by Lemma 2.40 it is a proof of knowledge and by Proposition 2.32 it is a computationally convincing proof. □

11.6 The Mix-Net

We are now ready to give a detailed description of the mix-net. Recall that $k' = \lceil (k+1)/2 \rceil$ denotes the number of mix-servers needed for majority.

Protocol 11.11 (Mix-Net). The mix-net $\pi_{\mathrm{RMN}} = (S_1, \ldots, S_N, M_1, \ldots, M_k)$ consists of senders S_i, and mix-servers M_j.

Sender S_i. Each sender S_i proceeds as follows.

1. Wait until $(M_l, \mathsf{N}, K_1, \mathsf{N}', \{K'_i\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ appears on $\mathcal{F}_{\mathrm{BB}}$ for k' distinct indices l.

2. Wait for an input $(\mathtt{Send}, m_i)$, $m_i \in \{0,1\}^{\kappa - \kappa_r}$. Choose $r_i \in \mathbb{Z}^*_{\mathsf{N}}$, $b_i \in \mathbb{Z}_{\mathsf{N}'}$ and $r'_i \in \mathbb{Z}^*_{\mathsf{N}'}$ randomly and compute

$$u_i = E_{K_1,\mathsf{N}}(m_i, r_i) \ , \quad u'_i = E_{(\mathsf{g}')^{b_i} K'_i, \mathsf{N}'}(m_i, r'_i) \ , \text{ and}$$
$$(\alpha_i, \mathrm{state}_i) = P_{\mathrm{eq}}(\mathsf{N}, K_1, u_i, \mathsf{N}', (\mathsf{g}')^{b_i} K'_i, u'_i, \mathbf{N}, \mathbf{g}, \mathbf{h}, m_i, r_i, r'_i) \ .$$

   Then hand $(\mathtt{Write}, \mathtt{Submit}, (b_i, u_i, u'_i), \mathtt{Commit}, \alpha_i)$ to $\mathcal{F}_{\mathrm{BB}}$.

3. Wait until $(M_j, \mathtt{Challenge}, S_i, c_i)$ appears on $\mathcal{F}_{\mathrm{BB}}$ for k' distinct j with identical c_i. Then compute $e_i = P_{\mathrm{eq}}(\mathrm{state}_i, c_i)$ and hand $(\mathtt{Write}, \mathtt{Reply}, e_i)$ to $\mathcal{F}_{\mathrm{BB}}$.

Mix-Server M_j. Each mix-server M_j proceeds as follows.

Preliminaries

1. Wait for a message on the form

$$(\mathtt{Keys}, (\mathbf{N}, g, h, \mathsf{N}', \mathsf{N}, K_0, K_1, \mathsf{v}, (\mathsf{v}_l)_{l=1}^k, (F_{l,l'})_{l,l' \in \{1,\ldots,k\}}), (\mathsf{d}_j, (\mathsf{d}_{l,j}, \mathsf{t}_{l,j})_{l=1}^k))$$

   from $\mathcal{F}_{\mathrm{PKG}}$.

2. Hand $(\mathtt{GenerateCoins}, (N+k)(\kappa+\kappa_r) + (\kappa+\kappa_r) + 2(\kappa+\kappa_r) + N(\kappa+\kappa_r))$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\mathtt{Coins}, \{K'_i\}_{i=1}^N, \{\bar{K}'_j\}_{j=1}^k, \mathsf{g_f}, \mathbf{g}, \mathbf{h}, g_1, \ldots, g_N)$. Then hand $(\mathtt{Write}, \mathsf{N}, K_1, \mathsf{N}', \{K'_i\}_{i=1}^N, \mathbf{N}, \mathbf{g}, \mathbf{h})$ to $\mathcal{F}_{\mathrm{BB}}$.

Reception of Inputs

3. Initialize $L_0 = \emptyset$, $I = \emptyset$ and $J = \emptyset$.
4. Repeat
   a) When given input (`Run`) hand (`Write`, `Run`) to $\mathcal{F}_{\mathrm{BB}}$.
   b) When a new entry $(T, M_l, \texttt{Run})$ appears on $\mathcal{F}_{\mathrm{BB}}$ set $J \leftarrow J \cup \{l\}$ and if $|J| \geq k'$ set $T_{\mathrm{run}} = T$ and go to Step 5.
   c) When a new entry $(S_i, \texttt{Submit}, (b_i, u_i, u_i'), \texttt{Commit}, \alpha_i)$ appears on $\mathcal{F}_{\mathrm{BB}}$ such that $i \notin I$, set $I \leftarrow I \cup \{i\}$ and hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_i)$. Hand $(\texttt{Write}, \texttt{Challenge}, S_i, c_i)$ to $\mathcal{F}_{\mathrm{BB}}$.
5. Request the contents on $\mathcal{F}_{\mathrm{BB}}$ with index less than T_{run}. Find for each i the first occurrences of entries on the forms $(T_i, \texttt{Submit}, (b_i, u_i, u_i'), \texttt{Commit}, \alpha_i)$, $(T_{j,i}', M_j, \texttt{Challenge}, S_i, c_i)$, and $(T_i'', S_i, \texttt{Reply}, e_i)$. Then form a list L_0 of all cryptotexts $u_i^2 \bmod \mathsf{N}^2$ such that $T_i < T_{j,i}' < T_i'' < T_{\mathrm{run}}$ for at least k' distinct indices j and $V_{\mathrm{eq}}(\mathsf{N}, K_1, u_i, \mathsf{N}', (\mathsf{g}')^{b_i} K_i', u_i', \mathbf{N}, \mathbf{g}, \mathbf{h}, \alpha_i, c_i, e_i) = 1$.

Re-encryption and Permutation

6. Write $L_0 = \{u_{0,i}\}_{i=1}^{N'}$ for some N'. Then for $l = 1, \ldots, k$ do
   a) If $l = j$, then do
      i. Choose $r_{j,i} \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly, compute

$$\begin{aligned} L_j &= \{u_{j,i}\}_{i=1}^{N'} = \mathsf{Sort}(\{\mathsf{g}_{\mathsf{f}}^{r_{j,i}} u_{j-1,i}^2 \bmod \mathsf{N}^2\}_{i=1}^{N'}) \ , \text{ and} \\ (\alpha_j, \mathrm{state}_j) &= P_{\mathrm{prp}}(\mathsf{N}, \mathsf{g}_{\mathsf{f}}, L_{l-1}^4, L_l^2, \mathbf{N}, \mathbf{g}, \mathbf{h}, \\ &\qquad g, g_1, \ldots, g_{N'}, \{2r_{j,i}\}_{i=1}^{N'}) \ , \end{aligned}$$

      and hand $(\texttt{Write}, \texttt{List}, L_j, \texttt{Commit1}, \alpha_j)$ to $\mathcal{F}_{\mathrm{BB}}$. The exponentiations L_{l-1}^4 and L_l^2 should be interpreted term-wise.
      ii. Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_j)$. Then compute $(\alpha_j', \mathrm{state}_j') = P_{\mathrm{prp}}(\mathrm{state}_j, c_j)$ and hand $(\texttt{Write}, \texttt{Commit2}, \alpha_j')$ to $\mathcal{F}_{\mathrm{BB}}$.
      iii. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_j')$. Then compute $e_j = P_{\mathrm{prp}}(\mathrm{state}_j', c_j')$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\mathrm{BB}}$.
   b) If $l \neq j$, then do
      i. Wait until an entry $(M_l, \texttt{List}, L_l, \texttt{Commit1}, \alpha_l)$ appears on $\mathcal{F}_{\mathrm{BB}}$.
      ii. Hand $(\texttt{GenerateCoins}, \kappa)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_l)$.
      iii. Wait for a new entry $(M_l, \texttt{Commit2}, \alpha_l')$ on $\mathcal{F}_{\mathrm{BB}}$. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\mathrm{CF}}$ and wait until it returns $(\texttt{Coins}, c_l')$.

iv. Wait for a new entry $(M_l, \texttt{Reply}, e_l)$ on $\mathcal{F}_{\text{BB}}$ and compute

$$b_l = V_{\text{prp}}(\mathsf{N}, \mathsf{g}_\mathsf{f}, L_{l-1}^4, L_l^2, \mathbf{N}, \mathbf{g}, \mathbf{h}, g, g_1, \ldots, g_{N'}, \alpha_l, c_l, \alpha_l', c_l', e_l) \ .$$

v. If $b_l = 0$, then set $L_l = L_{l-1}^2$.

Joint Re-encryption

7. Choose $\bar{b}_j \in \mathbb{Z}_{\mathsf{N}'}$, $m_{j,i} \in \mathbb{Z}_{\mathsf{N}'}$ and $s'_{j,i} \in \mathbb{Z}_{\mathsf{N}'}^*$ randomly and compute $W_j' = \{w'_{j,i}\}_{i=1}^{N'} = \{E_{\mathsf{g}'^{\bar{b}_j}\bar{K}'_j,\mathsf{N}'}(m_{j,i}, s'_{j,i})\}_{i=1}^{N'}$. Then hand $(\texttt{Write}, \texttt{RandExp}, \bar{b}_j, W_j')$ to $\mathcal{F}_{\text{BB}}$.

8. Wait until $(\texttt{RandExp}, \bar{b}_l, W_l')$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$. Then choose $s_{j,i} \in \mathbb{Z}_{\mathsf{N}}^*$ randomly and compute

$$\begin{aligned} W_j &= \{w_{j,i}\}_{i=1}^{N'} = \{E_{K_0,\mathsf{N}}(m_{j,i}, s_{j,i})\}_{i=1}^{N'} \ , \quad \text{and} \\ (\alpha_j, \text{state}_j) &= P_{\text{eq}}(\mathsf{N}, K_0, W_j, \mathsf{N}', K', W_j', \mathbf{N}, \mathbf{g}, \mathbf{h}, \\ &\qquad \{m_{j,i}\}_{i=1}^{N'}, \{s_{j,i}\}_{i=1}^{N'}, \{s'_{j,i}\}_{i=1}^{N'}) \ . \end{aligned}$$

Then hand $(\texttt{Write}, \texttt{RandExp}, W_j, \texttt{Commit}, \alpha_j)$ to $\mathcal{F}_{\text{BB}}$.

9. Wait until $(\texttt{RandExp}, W_l, \texttt{Commit}, \alpha_l)$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$. Hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\text{CF}}$ and wait until it returns $(\texttt{Coins}, c)$. Compute $e_j = P_{\text{eq}}(\text{state}_j, c)$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\text{BB}}$.

10. Wait until $(\texttt{Reply}, e_l)$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$. Let I be the lexicographically first set of k' indices such that

$$V_{\text{eq}}(\mathsf{N}, K_0, W_l, \mathsf{N}', K', W_l', \mathbf{N}, \mathbf{g}, \mathbf{h}, \alpha_l, c, e_l) = 1 \ .$$

11. Compute

$$L_{k+1} = \{u_{k+1,i}\}_{i=1}^{N'} = \left\{ u_{k,i} \prod_{l \in I} w_{l,i}^{\prod_{l' \neq l} \frac{l'}{l'-l}} \right\}_{i=1}^{N'} \ .$$

Joint Decryption

12. Compute $\Gamma_j = \{v_{j,i}\}_{i=1}^{N'} = \{u_{k+1,i}^{2\mathsf{d}_j}\}_{i=1}^{N}$ using d_j and a proof $(\alpha_j, \text{state}_j) = P_{\text{exp}}(\mathsf{N}, \mathsf{v}, \mathsf{v}_j, L_{k+1}^2, \Gamma_j)$. Then hand $(\texttt{Write}, \texttt{Decrypt}, \Gamma_j, \texttt{Commit}, \alpha_j)$ to $\mathcal{F}_{\text{BB}}$, where exponentiation is interpreted element-wise.

13. Wait until $(M_l, \texttt{Decrypt}, \Gamma_l, \texttt{Commit}, \alpha_l)$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$. Then hand $(\texttt{GenerateCoins}, \kappa_c)$ to $\mathcal{F}_{\text{CF}}$ and wait until it returns $(\texttt{Coins}, c)$.

14. Compute $e_j = P_{\text{exp}}(\text{state}_j, c)$ and hand $(\texttt{Write}, \texttt{Reply}, e_j)$ to $\mathcal{F}_{\text{BB}}$.

15. Wait until $(\texttt{Reply}, e_l)$ appears on $\mathcal{F}_{\text{BB}}$ for $l = 1, \ldots, k$. For $l = 1, \ldots, k$ do the following. If $V_{\text{exp}}(\mathsf{N}, \mathsf{v}, v_l, L_{k+1}^2, \Gamma_l, \alpha_l, c, e_l) = 0$ do

a) Hand $(\texttt{Write}, \texttt{Recover}, M_l, \mathsf{d}_{l,j}, \mathsf{t}_{l,j})$ to $\mathcal{F}_{\mathrm{BB}}$.

b) Wait until $(M_{l'}, \texttt{Recover}, M_l, \mathsf{d}_{l,l'}, \mathsf{t}_{l,l'})$ appears on $\mathcal{F}_{\mathrm{BB}}$ for $l' = 1, \ldots, k$. Then find a subset I of k' indices l' such that $F_{l,l'} = g^{\mathsf{d}_{l,l'}} h^{\mathsf{t}_{l,l'}}$ and Lagrange interpolate d_l

$$\mathsf{d}_l = \sum_{l' \in I} \mathsf{d}_{l,l'} \prod_{l'' \neq l'} \frac{l''}{l'' - l'} \bmod q \ .$$

c) Compute $\Gamma_l = \{v_{l,i}\}_{i=1}^{N'} = \{u_{k,i}^{2\mathsf{d}_l}\}_{i=1}^{N}$.

16. Let L_{out} be the list of strings in $\{L(\prod_{l=1}^{k} v_{l,i})/2^{k+2}\}_{i=1}^{N'}$ truncated to $\kappa - \kappa_r$ bits. The mix-server outputs $(\texttt{Output}, \mathsf{Sort}(L_{\mathrm{out}}))$.

Remark 11.12. For simplicity the mix-servers generate a separate challenge for each sender in the protocol. It is obviously possible to let a set of senders execute their proofs in parallel and use the same challenge for these senders to improve efficiency. The squaring of elements is required to keep cryptotexts in $\mathrm{QR}_{\mathsf{N}^2}$. All interactive proofs in the protocol can be made non-interactive in the random oracle model. Using ideas from Damgård and Groth [56] there is a natural way to avoid the generation of individual keys for each sender and mix-server in the random oracle model. Since the penalty of generating these keys is small in our setting, and the random oracle model is needed to avoid it, we do not make this solution explicit.

11.7 Security Analysis

The following theorem captures the security of Protocol 11.11 above.

Theorem 11.13. *The protocol π_{RMN} above securely realizes $\mathcal{F}_{\mathrm{RMN}}$ in the $(\mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{PKG}}, \mathcal{F}_{\mathrm{CF}})$-hybrid model for $\mathcal{M}^*_{k/2}$-adversaries under the DCR-assumption, the strong RSA-assumption, and the DL-assumption.*

11.7.1 Proof of Theorem 11.13

To prove Theorem 11.13 we describe an ideal adversary $\mathcal{S}(\cdot)$ that runs any hybrid adversary $\mathcal{A}$ as a black-box. Then we show that if $\mathcal{S}$ does not imply the security of the protocol, we can break the DCR-assumption, the DL-assumption or the strong RSA-assumption. The proof follows common practice and goes through a number of games, where each game is a slight modification of the previous game. This proof technique is sometimes called "hybrid proof" or "game hopping".

The proof consists of three parts. The first part describes the ideal adversary $\mathcal{S}$. To simplify game hopping we mark clearly the parts of $\mathcal{S}$ that are later replaced as numbered "switches". The second part gives a sequence of games leading from the ideal model to the hybrid model in which the protocol executes. For all except one of the hops we prove that the output distributions of neighboring games are

negligibly close. The remaining hop requires rewinding techniques and is treated in the third part of the proof.

We write $X \approx Y$ for two ensembles, $X = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa\in\mathbb{N}}$, of binary random variables if $|\Pr[X = 1] - \Pr[Y = 1]|$ is negligible.

The Ideal Adversary

Let I_S and I_M be the set of indices of parties corrupted by $\mathcal{A}$ so far of the sender type and the mix-server type respectively. At the start of the execution $I_S = I_M = \emptyset$.

Switch 1 (Switching Keys). The ideal key generator $\mathcal{F}_{\mathrm{PKG}}$ is simulated honestly, except that K_0 and K_1 are defined by $K_0 = E_{\mathsf{N}}(1, R_0) = \mathsf{g}R_0^{2\mathsf{N}} \bmod \mathsf{N}^2$ and $K_1 = E_{\mathsf{N}}(0, R_1) = R_1^{2\mathsf{N}} \bmod \mathsf{N}^2$. Thus, K_0 is now an encryption of 1 and K_1 is an encryption of 0 instead of vice versa.

Simulation of Honest Senders. Whenever $\mathcal{S}$ receives $(\mathcal{S}, \tilde{S}_i, \mathtt{Input}, c')$ from $\mathcal{C}_{\mathcal{I}}$ it stores (i, c') and instructs S_i to compute a submission in the simulated execution. This has to be done without knowledge of the message handed by $\tilde{S}_i$ to $\mathcal{F}_{\mathrm{RMN}}$. The simulated sender S_i computes (b_i, u_i, u_i') and α_i honestly except for the following.

Switch 2 (Submission of Fake Inputs). It sets $m_i = 0$. Note that this most likely is not the message submitted by $\tilde{S}_i$.

Switch 3 (Submission of Fake Inputs). It sets $b_i = -a_i \bmod \mathsf{N}'$. This is to make the submission equivocable.

The simulation is interrupted whenever $\mathcal{F}_{\mathrm{BB}}$ is about to hand a tuple $(\mathcal{A}, \tilde{S}_i, \mathtt{Input}, c, S_i, \mathtt{Reply}, e_i)$ to $\mathcal{C}_{\mathcal{I}}$ for some $i \notin I_S$ (that may be different from the one in the previous paragraph). If this tuple would imply that $u_i^2 \bmod \mathsf{N}^2$ is added to L_0 according to the protocol if it appears on $\mathcal{F}_{\mathrm{BB}}$, then $\mathcal{S}$ finds the pair (i, c') for some c', stores (c, c') and continues the simulation.

Extraction From Corrupted Senders. When the submission of a corrupted sender S_i, with $i \in I_S$, is accepted its cryptotext u_i^2 is added to the list L_0, and the ideal adversary must extract the submitted cleartext and hand it to $\mathcal{F}_{\mathrm{RMN}}$. The message is extracted as follows.

When $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, \mathtt{Input}, c, S_i, \mathtt{Reply}, e_i)$ to $\mathcal{C}_{\mathcal{I}}$ the simulation is interrupted and $\mathcal{S}$ checks if $u_i^2 \bmod \mathsf{N}^2$ would be added to L_0 if the tuple appears on $\mathcal{F}_{\mathrm{BB}}$. If so it does the following.

Switch 4 (Extracting Messages). It computes $m_i' = D_{\mathsf{d}'}(u_i')/(b_i + a_i) \bmod \mathsf{N}'$ unless $b_i + a_i = 0 \bmod \mathsf{N}'$ in which case it sets $m_i' = 0$. Since a_i is encrypted the latter event should happen with negligible probability.

Then it instructs $\tilde{S}_i$ to hand $(\mathtt{Send}, m_i')$ to $\mathcal{F}_{\mathrm{MN}}$ and waits until it receives $(\mathcal{S}, \tilde{S}_i, \mathtt{Input}, c')$ from $\mathcal{C}_{\mathcal{I}}$. Then it stores (c, c') and the simulation is continued.

Accepting Inputs. The simulation is interrupted whenever $\mathcal{F}_{\mathrm{BB}}$ is about to hand tuple $(\mathcal{A}, \tilde{S}_i, \texttt{AcceptInput}, c)$ to $\mathcal{C}_{\mathcal{I}}$ for some $\tilde{S}_i$ which is either honest or corrupted. If there is a c' such that a pair (c, c') is stored, then $\mathcal{S}$ hands $(\mathcal{F}_{\mathrm{MN}}, \texttt{AcceptInput}, c')$ to $\mathcal{C}_{\mathcal{I}}$ and waits for $(\mathcal{S}, S_i, \texttt{Send})$ from $\mathcal{C}_{\mathcal{I}}$. Then the simulation is continued.

Corruption of Sender. If $\mathcal{A}$ corrupts S_i, $\mathcal{S}$ must construct a plausible history tape for S_i that is consistent with any previous communication involving S_i before handing over the state and control of S_i to $\mathcal{A}$. At the start of the simulation the ideal adversary $\mathcal{S}$ defines $\mathbf{g} = \mathbf{h}^{\mathbf{x}}$ for a random $\mathbf{x} \in [0, 2^{\kappa+\kappa_r} - 1]$ and instructs $\mathcal{F}_{\mathrm{CF}}$ to use these values. Then corruptions are treated as follows.

Switch 5 (Correcting the History On Corruption). The ideal adversary first corrupts $\tilde{S}_i$ to extract the message m_i handed by $\tilde{S}_i$ to $\mathcal{F}_{\mathrm{RMN}}$. Denote by $(0, r_i, r_i')$, $(r_i'', s_{i,0}, s_{i,1}, t_i, s_{i,2})$, and c_i the private input, the random tape, and the challenge used in the simulation of S_i. Denote by I_i the common input and T_i the proof transcript. The ideal adversary $\mathcal{S}$ computes

$$(\bar{r}_i, \bar{r}_i', \bar{r}_i'', \bar{s}_{i,0}, \bar{s}_{i,1}, \bar{t}_i, \bar{s}_{i,2}) = \mathsf{His}(R_1, R', \mathbf{x}, m_i, r_i, r_i', r_i'', s_{i,0}, s_{i,1}, t_i, s_{i,2}, c_i) \ ,$$

using the algorithm His from Lemma 11.4, and restarts the simulation of S_i using the new values. It continues the new simulation of S_i up to the point in the protocol where it was corrupted. Finally, the resulting state of S_i, and control over S_i, is handed to the adversary $\mathcal{A}$.

Simulation of Honest Mix-Servers and Extraction from Corrupted Mix-Servers. When a corrupt mix-server M_j, for $j \in I_M$, writes $\texttt{Run}$ on $\mathcal{F}_{\mathrm{BB}}$, $\mathcal{S}$ must make sure that $\tilde{M}_j$ sends $(\texttt{Run})$ to $\mathcal{F}_{\mathrm{MN}}$. Otherwise it may not be possible to deliver an output to honest mix-servers at a later stage. If an honest dummy mix-server $\tilde{M}_j$, for $j \notin I_M$, receives $(\texttt{Run})$ from $\mathcal{Z}$, $\mathcal{S}$ must make sure that M_j receives $(\texttt{Run})$ from $\mathcal{Z}'$. In both instances $\mathcal{S}$ must do this in two steps, first sending and then instructing $\mathcal{F}_{\mathrm{MN}}$ or $\mathcal{F}_{\mathrm{BB}}$ respectively to accept the submitted message. If an honest mix-server M_j, for $j \notin I_M$, outputs $(\texttt{Output}, L')$, $\mathcal{S}$ must make sure that $\tilde{M}_j$ does the same. This is done as follows.

1. Let $j \in I_M$. If $\mathcal{F}_{\mathrm{BB}}$ receives $(M_j, \texttt{Write}, \texttt{Run})$, then $\mathcal{S}$ continues the simulation until $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, \texttt{Input}, c, \tilde{M}_j, \texttt{Run})$ to $\mathcal{C}_{\mathcal{I}}$. Then the simulation of $\mathcal{F}_{\mathrm{BB}}$ is interrupted and $\tilde{M}_j$ hands $(\texttt{Run})$ to $\mathcal{F}_{\mathrm{MN}}$. When $\mathcal{S}$ receives $(M_j, \texttt{Input}, c')$ from $\mathcal{C}_{\mathcal{I}}$ it stores the pair (c, c') and continues the simulation of $\mathcal{F}_{\mathrm{BB}}$.

2. Let $j \notin I_M$. If $\mathcal{S}$ receives $(\mathcal{S}, \texttt{Input}, c', \tilde{M}_j)$ from $\mathcal{C}_{\mathcal{I}}$, then $\mathcal{S}$ hands $(\texttt{Run})$ to M_j and continues the simulation until $\mathcal{F}_{\mathrm{BB}}$ is about to hand $(\mathcal{A}, \texttt{Input}, c, \tilde{M}_j, \texttt{Run})$ to $\mathcal{C}_{\mathcal{I}}$. Then the pair (c, c') is stored and the simulation is continued.

3. If $\mathcal{F}_{\mathrm{BB}}$ receives $(\mathcal{A}, \texttt{AcceptInput}, c)$ the simulation of $\mathcal{F}_{\mathrm{BB}}$ is interrupted. If there is a pair (c, c') for some c', then $\mathcal{S}$ hands $(\mathcal{F}_{\mathrm{MN}}, \texttt{AcceptInput}, c')$ to $\mathcal{C}_{\mathcal{I}}$

and waits until it receives $(\mathcal{S}, \tilde{M}_j, \texttt{Run})$ or $((\mathcal{S}, \tilde{M}_j, \texttt{Output}, L'), \{(\tilde{M}_l, \tau_l)\}_{l=1}^k)$ from $\mathcal{C}_\mathcal{I}$. Then the simulation of $\mathcal{F}_{\text{BB}}$ is continued.

4. For the outputting of messages we need a "switch".

   Switch 6 (Outputting Messages). Let $j \notin I_M$. If M_j outputs $(\texttt{Output}, L')$, $\mathcal{S}$ sends $(1, \tau_j)$ to $\mathcal{C}_\mathcal{I}$, i.e. $\mathcal{S}$ instructs $\mathcal{C}_\mathcal{I}$ to deliver $(\texttt{Output}, L')$ to $\tilde{M}_j$.

   Note that it does not matter if the adversary manages to replace or modify some of the messages in the simulated protocol. The output handed to honest mix-servers still consists of the messages handed to $\mathcal{F}_{\text{RMN}}$.

Consider now the simulation of the computations of the honest mix-servers. These are simulated honestly except in the re-encryption phase, i.e., Steps 7-11. Note that when these steps are simulated, $\mathcal{F}_{\text{RMN}}$ has already handed $(\texttt{Output}, L')$ to $\mathcal{S}$.

Unless $\mathcal{S}$ somehow introduces the correct cleartexts from L' in the simulation, all messages are zero.

Switch 7 (Replacing the Zero Cleartexts With Correct Ones). Step 7 of each simulated mix-server M_j, with $j \notin I_M$, is simulated honestly except that $\mathcal{S}$ sets $\bar{b}_j = -\bar{a}_j \bmod \mathsf{N}'$ for $j \notin I_M$. After all mix-servers M_l have published $w'_{l,i}$, $\mathcal{S}$ computes

$$m_{l,i} = D_{\mathsf{d}'}(w'_{l,i})/(\bar{b}_l + \bar{a}_l) \bmod \mathsf{N}' \ ,$$

for $l \in I_M$. It then defines $m_{0,i} = m_{\pi(i)} 2^{k+1} \bmod \mathsf{N}$ for a random permutation $\pi \in \Sigma_{N'}$ and defines $f_i(x) \in \mathbb{Z}_\mathsf{N}[x]$ as a random $(k'-1)$-degree polynomial such that $f_i(j) = m_{j,i}$ for $j \in I_M \cup \{0\}$. This should be possible since $|I_M| < k'$, unless some non-invertible element $a \in \mathbb{Z}_\mathsf{N}$ is encountered.

Finally, it defines $\bar{m}_{j,i} = f_i(j)$ and $\bar{s}'_{j,i} = s'_{j,i} R'^{\,m_{j,i} - \bar{m}_{j,i}}_j \bmod (\mathsf{N}')^2$ for $j \notin I_M$. Then it restarts each honest M_j with a modified random string to give the new values. This is to give M_j a proper history tape that can be handed to the adversary $\mathcal{A}$.

Corruption of Mix-Servers. Corruption of a mix-server is straightforward. The ideal adversary $\mathcal{S}$ simply hands over the current state of the simulated machine and the control over it to $\mathcal{A}$.

Extraction From Corrupted Mix-Servers. When M_j for $j \in I_M$ hands $(\texttt{Write}, \texttt{Run})$ to $\mathcal{F}_{\text{BB}}$, $\mathcal{S}$ interrupts the simulation and instructs the corresponding dummy mix-server $\tilde{M}_j$ to hand $(\texttt{Run})$ to $\mathcal{F}_{\text{RMN}}$. Then $\mathcal{S}$ waits until it receives $(\tilde{M}_j, \texttt{Run})$ or $((\tilde{M}_j, \texttt{Output}, L'), \{(\tilde{M}_l, \tau_l)\}_{l=1}^k)$ from $\mathcal{C}_\mathcal{I}$ and the simulation of $\mathcal{F}_{\text{BB}}$ is continued.

The Main Hybrid Argument

Suppose that $\mathcal{S}$ does not imply the security of the protocol. Then there exists a hybrid adversary $\mathcal{A}$, an environment $\mathcal{Z}$ with auxiliary input $z = \{z_\kappa\}$, a constant

$c > 0$ and an infinite index set $\mathcal{N} \subset \mathbb{N}$ such that for $\kappa \in \mathcal{N}$

$$|\Pr[\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\text{RMN}}})) = 1] - \Pr[\mathcal{Z}_z(\mathcal{H}(\mathcal{A}, \pi_{\text{RMN}}^{(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{PKG}}, \mathcal{F}_{\text{CF}})})) = 1]| \geq \frac{1}{\kappa^c}$$

where $\mathcal{S}$ runs $\mathcal{A}$ as a black-box as described above, i.e. $\mathcal{S} = \mathcal{S}(\mathcal{A})$.

We show that this gives a contradiction by a hybrid argument which gradually turns the ideal model execution into the hybrid model execution of the protocol. Denote by H the machine that simulates all machines in the ideal model $\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\text{RMN}}}))$. We use the convention of adding as a subscript the index of the "switch" that is replaced in the game previously considered. In this part of the proof we prove that

$$H \approx H_{2,5} \text{ and } H_{2,5,1} \approx H_{2,5,1,4} \approx H_{2,5,1,4,3} \approx H_{2,5,1,4,3,7} \approx H_{2,5,1,4,3,7,6} \ .$$

The proof that $H_{2,5} \approx H_{2,5,1}$ is postponed for Section 11.7.1.

Denote by $H_{2,5}$ the machine that simulates H except for the following changes. Instead of executing Switch 2 it extracts the message m_i handed by $\tilde{S}_i$ to $\mathcal{F}_{\text{RMN}}$ from the simulation of $\mathcal{F}_{\text{RMN}}$ and use it when simulating the submission of a message from S_i. Instead of executing Switch 5 to correct the history of S_i it simply hands the state of and control over S_i to $\mathcal{A}$. Thus, although the resulting cleartexts are still all zero, the senders compute their cryptotexts as in the real protocol.

Claim 1. $H \approx H_{2,5}$.

Proof. The modifications of H only change the distribution negligibly, since by Proposition 11.4 the distribution of the tuple produced by the algorithm His is statistically close to the corresponding tuple in a real execution. □

Denote by $H_{2,5,1}$ the machine identical to $H_{2,5}$ except that instead of executing Switch 1 it simulates $\mathcal{F}_{\text{PKG}}$ honestly. Note that this means that the cryptotexts computed by the senders become valid encryptions of their respective messages, and at the same time the joint re-encryption step no longer alters the cleartexts.

Claim 2. Under the DCR-assumption $H_{2,5} \approx H_{2,5,1}$.

The proof of this claim is involved and postponed for Section 11.7.1 to keep the main track of the proof simple.

Denote by $H_{2,5,1,4}$ the machine identical to $H_{2,5,1}$ except that instead of executing Switch 4 it defines m_i' by $m_i' = D_{\mathsf{d}}(u_i)$. If this changes the distribution we must have $D_{\mathsf{d}}(u_i) \neq D_{\mathsf{d}'}(u_i')/(b_i + a_i)$ for some i. Since each sender proves that it uses the same exponent m_i when constructing both u_i and u_i', this can only happen if $b_i = -a_i \bmod \mathsf{N}'$. However, a_i is randomly chosen so this means that the adversary can break the semantic security of the Paillier cryptosystem. More precisely we have the following claim.

Claim 3. Under the CR-assumption and the strong RSA-assumption $H_{2,5,1} \approx H_{2,5,1,4}$.

Proof. Suppose first that $\Pr[b_i + a_i = 0 \bmod \mathsf{N}' \text{ for some } i \in I_S]$ is non-negligible. An averaging argument implies that there exists a fixed i such that $\Pr[b_i + a_i = 0 \bmod \mathsf{N}']$ is non-negligible. Next we show that this allow us to break the CR-assumption.

We define D to be the machine identical to $H_{2,5,1,4}$ except for the following changes. It waits for a modulus N' and a random element $u \in \mathbb{Z}^*_{\mathsf{N}'^2}$. It instructs $\mathcal{F}_{\mathrm{PKG}}$ to use N' and it instructs $\mathcal{F}_{\mathrm{CF}}$ to use $K_i' = u^2 \bmod (\mathsf{N}')^2$ and continues the simulation. Then it waits until S_i has handed $(\mathtt{Write}, (b_i, u_i, u_i'), \mathtt{Commit}, \alpha_i)$ to $\mathcal{F}_{\mathrm{BB}}$ and outputs $b_i/2$. If S_i never does so it outputs $\perp$. It follows that D outputs the residuosity class of the challenge cryptotext with non-negligible probability. This contradicts the CR-assumption, and we conclude that $\Pr[b_i + a_i \neq 0 \bmod \mathsf{N}' \text{ for all } i \in I_S]$ with overwhelming probability.

Denote by E_i^{distinct} the event that

$$D_{\mathsf{d}}(u_i) \neq D_{\mathsf{d}'}(u_i')/(b_i + a_i) \quad \text{and}$$
$$V_{\mathrm{eq}}(\mathsf{N}, K_1, u_i, \mathsf{N}', (\mathsf{g}')^{b_i} K_i', u_i', \mathbf{N}, \mathbf{g}, \mathbf{h}, \alpha_i, c_i, e_i) = 1 \ .$$

The only way the simulation carried out by $H_{2,5,1,4}$ can differ from that of $H_{2,5,1}$ is if the event E_i^{distinct} occurs for some i. We assume that $\Pr[E_i^{\mathrm{distinct}} \text{ for some } i \in I_S]$ is non-negligible and show that this gives a contradiction.

An averaging argument implies that $\Pr[E_i^{\mathrm{distinct}}]$ is non-negligible for some fixed i. We define a malicious prover P^*_{eq} that contradicts the soundness of the protocol, i.e., Proposition 11.6. It waits for public parameters $(\mathbf{N}, \mathbf{g}, \mathbf{h})$. Then it simulates $H_{2,5,1}$, and instructs $\mathcal{F}_{\mathrm{PKG}}$ to use the RSA-modulus and it instructs $\mathcal{F}_{\mathrm{CF}}$ to output the RSA-generators $\mathbf{g}$ and $\mathbf{h}$. The simulation is interrupted when S_i hands $(\mathtt{Write}, (b_i, u_i, u_i'), \mathtt{Commit}, \alpha_i)$ to $\mathcal{F}_{\mathrm{BB}}$. Then P^*_{eq} outputs (u_i, u_i', α_i) and waits for a challenge c_i. When it receives the challenge it instructs $\mathcal{F}_{\mathrm{CF}}$ to output c_i instead of generating a new challenge in the simulation. Finally, it continues the simulation until S_i hands $(\mathtt{Write}, \mathtt{Reply}, e_i)$ to $\mathcal{F}_{\mathrm{BB}}$, at which point it outputs e_i. We have established that $\Pr[b_i + a_i = 0 \bmod \mathsf{N}']$ is negligible and $\Pr[E_i^{\mathrm{distinct}}]$ is non-negligible. The union bound implies that $\Pr[E_i^{\mathrm{distinct}} \wedge b_i + a_i \neq 0 \bmod \mathsf{N}']$ is non-negligible. Thus, the existence of P^*_{eq} contradicts Proposition 11.6 and we conclude that the claim holds. □

Denote by $H_{2,5,1,4,3}$ the machine identical to $H_{2,5,1,4}$ except that instead of executing Switch 3 it chooses $b_i \in \mathbb{Z}_{\mathsf{N}'}$ randomly for all $i \notin I_S$. Note that the simulator no longer uses the fact that $b_i = -a_i \bmod \mathsf{N}'$ and in the real execution b_i is chosen randomly. The change should only influence the output distribution negligibly unless the adversary can guess a randomly chosen encrypted cleartext a_i correctly. More precisely the following claim holds.

Claim 4. Under the DCR-assumption $H_{2,5,1,4} \approx H_{2,5,1,4,3}$.

Proof. The proof is by contradiction. Assume without loss that $\Pr[H_{2,5,1,4} = 1] - \Pr[H_{2,5,1,4,3} = 1]$ is non-negligible.

We describe a distinguisher D that breaks the semantic security of the Paillier cryptosystem. Denote by D the machine that simulates $H_{2,5,1,4}$ except that it accepts a modulus N' as input. Then it outputs the messages $(0,1)$ to the experiment. The experiment chooses a random bit b and computes a cryptotext $u = E_{\mathsf{N}'}(b)$ which it hands to D. D chooses $a_i, b_i \in \mathbb{Z}_{\mathsf{N}'}$ and $t_i \in \mathbb{Z}^*_{\mathsf{N}'}$ randomly and defines $K'_i = u^{-b_i}((\mathsf{g}')^1/u)^{a_i} t_i^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$ for $i \in \{1,\ldots,N\}$. These values are then used in the continued simulation of $\mathcal{F}_{\mathrm{CF}}$. Note that if u is on the form $E_{\mathsf{N}'}(1,r) = (\mathsf{g}')^1 r^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$, then K'_i is on the form $(\mathsf{g}')^{-b_i}(t'_i)^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$ for a randomly distributed $t'_i \in \mathbb{Z}^*_{\mathsf{N}'}$, i.e., D is identically distributed to $H_{2,5,1,4}$. On the other hand, if u is on the form $E_{\mathsf{N}'}(0,r) = r^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$, then K'_i is on the form $(\mathsf{g}')^{a_i}(t'_i)^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$ for a randomly distributed $t'_i \in \mathbb{Z}^*_{\mathsf{N}'}$, i.e., the output of D is identically distributed to $H_{2,5,1,4,3}$.

It follows that D breaks the semantic security of the Paillier cryptosystem, which is equivalent to breaking the DCR-assumption. □

Denote by $H_{2,5,1,4,3,7}$ the machine identical to $H_{2,5,1,4,3}$ except that instead of executing Switch 7 it simulates Steps 7-11 honestly. This does not alter the cleartexts, since now K_0 is an encryption of zero. The output distribution should only change negligibly, since the adversary can not tell if $\bar{b}_j = -\bar{a}_j \bmod \mathsf{N}'$, or not when the cryptosystem is semantically secure.

Claim 5. Under the DCR-assumption $H_{2,5,1,4,3} \approx H_{2,5,1,4,3,7}$.

Proof. Firstly, note that we may assume that $\mathcal{S}$ chooses $m_{l,i}$ for $l \notin I_M$ randomly in $H_{2,5,1,4,3}$, since K_0 is on the form $R_0^{2\mathsf{N}} \bmod \mathsf{N}^2$ and $(\mathsf{g}')^{b_i} K'_i$ is on the form $R'^{2\mathsf{N}'} \bmod (\mathsf{N}')^2$. More precisely, the distribution of $w'_{l,i}$ and $w_{l,i}$ does not change by this modification. Secondly, Proposition 11.4 implies that executing the real proof of equal cleartexts from the beginning instead of invoking the history algorithm His only changes the distribution negligibly.

This leaves us with the problem of changing the definition of $\bar{b}_j$ for $j \notin I_M$ from $\bar{b}_j = -\bar{a}_j \bmod \mathsf{N}'$ to random. That this only changes the output distribution of $H_{2,5,1,4,3}$ follows by a slight variation of the proof of Claim 4 and is left to the reader. □

Denote by $H_{2,5,1,4,3,7,6}$ the machine identical to $H_{2,5,1,4,3,7}$ except that it modifies Switch 6 as follows. Instead of sending $(1,\tau_j)$ to $\mathcal{C}_{\mathcal{I}}$ when it receives $(\texttt{Output}, L')$ from M_j it instructs $\mathcal{C}_{\mathcal{I}}$ to deliver the output L'. In principle it could be the case that L' is different from the true output of $\mathcal{F}_{\mathrm{RMN}}$, but this should only happen with negligible probability, since each mix-server proves in zero-knowledge that it behaves correctly.

Claim 6. Under the DCR-assumption, the strong RSA-assumption, and the DL-assumption $H_{2,5,1,4,3,7} \approx H_{2,5,1,4,3,7,6}$.

Proof. Denote by E_{distinct} the event that the set of cleartexts in L' does not correspond to the set of messages encrypted in L_0 in the simulation of $H_{2,5,1,4,3,7,6}$.

Denote by $E_{\text{shuffle},l}$ the event that the messages encrypted in L_l are distinct from those encrypted in L_{l-1}^2. Denote by $E_{\text{reenc},l}$ the event that W_l is not a list of encryptions of 0. Denote by $E_{\text{decr},l}$ the event that $v_{l,i} \neq u_{k,i}^{\mathsf{d}_l}$ for some i. Then it follows that one of the events $E_{\text{shuffle},l}$, $E_{\text{reenc},l}$ or $E_{\text{decr},l}$ must occur if the event E_{distinct} occurs. Thus, if each of these events occur with negligible probability the claim follows.

We prove that the probability of the event $E_{\text{shuffle},l}$ is negligible for all l. For each $l = 1, \ldots, k$ we construct an adversary $P^*_{\text{prp}_l}$ to the experiment considered in Proposition 9.23 as follows. It takes as input RSA parameters $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ and discrete logarithm parameters $\overline{g} = (g, g_1, \ldots, g_N)$ and then instructs $\mathcal{F}_{\text{PKG}}$ and $\mathcal{F}_{\text{CF}}$ to use these values in the simulation of $H_{2,5,1,4,3,7,6}$. It interrupts the simulation when M_l hands $(\texttt{Write}, \texttt{List}, L_l, \texttt{Commit1}, \alpha_l)$ to $\mathcal{F}_{\text{BB}}$, and outputs $(\mathsf{N}, L_{l-1}^4, L_l^2)$. Then it outputs α_l as the first commitment in the proof. A challenge value c_l is then accepted as input and $\mathcal{F}_{\text{CF}}$ is instructed to use this value as challenge to M_l. The simulation is continued until M_l hands $(\texttt{Write}, \texttt{Commit2}, \alpha'_l)$ to $\mathcal{F}_{\text{BB}}$ and $P^*_{\text{prp}_l}$ outputs α'_l as the second commitment of the proof. Finally, it accepts a second challenge value c'_l as input and instructs $\mathcal{F}_{\text{CF}}$ to use this value as challenge to M_l. The simulation is then continued until M_l hands $(\texttt{Write}, \texttt{Reply}, e_l)$ to $\mathcal{F}_{\text{BB}}$ at which point $P^*_{\text{prp}_l}$ outputs e_l as the final reply.

Denote by $T_{P^*_{\text{prp}}}(\kappa)$ a polynomial upper bound on the running time of any adversary $P^*_{\text{prp}_l}$. The adversaries are polynomial since $H_{2,5,1,4,3,7,6}$ is polynomial. Denote by r_{p_0} the random tapes of all machines in the simulation, except those parts needed to generate $\mathbf{\Gamma}$, $\overline{g}$ and $c_1, c'_1, \ldots, c_k, c'_k$. Let r_{p_l} be the list $(r_{\text{p}_{l-1}}, c_{l-1}, c'_{l-1})$. It follows that we may consider $P^*_{\text{prp}_l}$ as an adversary in the sense of Propositions 9.23 taking as input $(\mathbf{\Gamma}, \overline{g}, r_{\text{p}_l})$.

If $E_{\text{shuffle},l}$ is non-negligible, the adversary $P^*_{\text{prp}_l}$ breaks the soundness of the proof of knowledge of a shuffle and contradicts Proposition 9.23, which holds under the strong RSA-assumption and the DL-assumption.

If $\Pr[E_{\text{decr},l}]$ is non-negligible there are two cases to consider. Either M_l manages to output a faulty $v_{l,i}$ and still convince the other mix-servers that it was correct, or M_l outputs an invalid proof and some other $M_{l'}$ manages to open $F_{l,l'}$ in a different way than the way it was constructed leading to a faulty reconstruction of d_l, and thus a faulty $v_{l,i}$. We consider the second event first. Denote by $E_{\text{comm},l,l'}$ the event that $M_{l'}$ opens one of its commitments $F_{l,l'}$ using $(d_{l,l'}, t_{l,l'}) \neq (d'_{l,l'}, t'_{l,l'})$. If $\Pr[E_{\text{comm},l,l'}]$ is non-negligible for any pair (l, l') we define a machine B the breaks the DL-assumption as follows. It takes $g, h \in G_q$ as input instructs $\mathcal{F}_{\text{CF}}$ to use these values in the simulation of $H_{2,5,1,4,3,7,6}$ and if $M_{l'}$ uses valid $(d'_{l,l'}, t'_{l,l'}) \neq (d_{l,l'}, t_{l,l'})$ it outputs $(d'_{l,l'} - d_{l,l'})/(t_{l,l'} - t'_{l,l'}) \bmod q$. This is the logarithm of h in the basis g. Thus, B contradicts the DL-assumption.

Denote by $E_{\text{share},l}$ the event that M_l outputs a faulty $v_{l,i}$ and still convince the other mix-servers that it was correct. Consider now a malicious prover P^*_{exp} for Protocol 11.8 defined as follows. It simulates $H_{2,5,1,4,3,7,6}$ except for the randomness used to generate the random challenge c produced by $\mathcal{F}_{\text{CF}}$ in Step 13, which

it replaces by a challenge handed to it by the honest verifier V_{exp}. If $\Pr[E_{\text{share},l}]$ is non-negligible, a simple averaging argument implies that there exists a deterministic variant of P^*_{exp} which breaks the soundness of Protocol 11.8 and contradicts Proposition 11.10. Note that here the argument is quite easy, since the protocol π_{exp} is a proof system and not merely a computationally convincing proof.

If $\Pr[E_{\text{reenc},l}]$ is non-negligible there exists a malicious prover P^*_{eq} for Protocol 11.3. It takes as input RSA-parameters $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h})$ and simulates $H_{2,5,1,4,3,7,6}$ until Step 8. Then it outputs (W'_l, W_l) and α_l and waits for a random challenge c. When given a random challenge it instructs $\mathcal{F}_{\text{CF}}$ to use this value in the simulation. After Step 15 has been executed, it outputs the reply e_l. It follows that P^*_{eq} breaks the soundness of the proof of equal cleartexts and contradicts Proposition 11.6. □

We now conclude the proof of the theorem using the above claims. It follows by inspection that the distribution of $H_{2,5,1,4,3,7,6}$ is identical to the distribution of $\mathcal{Z}_z(\mathcal{H}(\mathcal{A}, \pi_{\text{RMN}}^{(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{PKG}}, \mathcal{F}_{\text{CF}})}))$. Claim 1-6 then imply that $\mathcal{Z}_z(\mathcal{I}(\mathcal{S}, \tilde{\pi}^{\mathcal{F}_{\text{RMN}}})) \approx \mathcal{Z}_z(\mathcal{H}(\mathcal{A}, \pi_{\text{RMN}}^{(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{PKG}}, \mathcal{F}_{\text{CF}})}))$, which contradicts the assumption about the adversary $\mathcal{A}$, and the theorem is true. It remains to prove Claim 2.

Proof of Claim 2

The proof is by contradiction. Suppose that there exists a constant c_0 and an infinite index set $\mathcal{N}$ such that

$$\Pr[H_{2,5,1} = 1] - \Pr[H_{2,5} = 1] \geq \frac{1}{\kappa^{c_0}} \ ,$$

for all $\kappa \in \mathcal{N}$. We show that we can exploit the non-negligible gap to break the semantic security of the Paillier cryptosystem. This is a contradiction and the claim must be true. In all claims below we implicitly assume the strong RSA-assumption, the DL-assumption, and the DCR-assumption.

More precisely we describe an adversary for the following special case of the polynomial indistinguishability experiment. A public key N and a random bit $b \in \{0, 1\}$ is generated and cryptotexts $K_0 = E_{\mathsf{N}}(b)$ and $K_1 = E_{\mathsf{N}}(1 - b)$ defined. Then the adversary D is given (N, K_0, K_1) and must guess b.

A naive first attempt is to define an adversary D that simulates $H_{2,5,1}$ except that $\mathcal{F}_{\text{PKG}}$ uses the modulus N and cryptotexts K_0 and K_1 given by the experiment. The output of D should then be distributed as the output of $H_{2,5,1}$ if $b = 1$ and as the output of $H_{2,5}$ if $b = 0$, and D would contradict the polynomial indistinguishability of the Paillier cryptosystem.

Unfortunately, the naive attempt does not work since the private key d is needed by the ideal adversary to simulate the decryption phase. Indeed, $\mathcal{F}_{\text{PKG}}$ must output correct private keys d_j to all mix-servers, since any mix-server may be adaptively corrupted. Let us modify both $H_{2,5}$ and $H_{2,5,1}$ such that they do not use the private key.

Denote by $H^c_{2,5}$ the machine that chooses a random index $j \in \{1, \ldots, k\}$ and simulates $H_{2,5}$ except that if the adversary corrupts M_j it outputs $\perp$. Define $H^c_{2,5,1}$ correspondingly.

Claim 7. $\Pr[H_{2,5} = 1] = \Pr[H^c_{2,5} = 1 \mid H^c_{2,5} \neq \perp]$ and
$\Pr[H_{2,5,1} = 1] = \Pr[H^c_{2,5,1} = 1 \mid H^c_{2,5,1} \neq \perp]$.

Proof. This follows since j is chosen randomly and independently. □

Denote by $H^{c,s}_{2,5}$ the machine that simulates $H^c_{2,5}$ with the following changes. In Step 6a it simulates the proof of knowledge of correct re-encryption-permutation of L_{j-1} to L_j using the statistical zero-knowledge simulator guaranteed by Proposition 9.22 instead of letting M_j compute it honestly. Similarly, in Steps 12-15 it simulates the proof of correctness of $v_{j,i}$ using the simulator guaranteed by Proposition 11.9 of M_j instead of letting it compute it honestly. Define $H^{c,s}_{2,5,1}$ correspondingly.

Claim 8. $|\Pr[H^c_{2,5} = 1 \mid H^c_{2,5} \neq \perp] - \Pr[H^{c,s}_{2,5} = 1 \mid H^{c,s}_{2,5} \neq \perp]|$ and
$|\Pr[H^c_{2,5,1} = 1 \mid H^c_{2,5,1} \neq \perp] - \Pr[H^{c,s}_{2,5,1} = 1 \mid H^{c,s}_{2,5,1} \neq \perp]|$ are negligible.

Proof. The claim follows from Proposition 9.22 and Proposition 11.9 which states that the simulations are statistically close to real executions of the protocols. □

The natural next step is to replace decryption using the private key d with simulated decryption where d is not used, since we have already made sure that d is not needed when proving correctness of decryption. The problem with this plan is that in the simulation of $H^{c,s}_{2,5,1}$ we do not know the correspondence between the cleartexts m_i and the cryptotexts $u_{k+1,i}$ to be decrypted, unless we use the private key d to actually decrypt these cryptotexts.

We overcome this problem by invoking the knowledge extractor for each corrupted mix-server M_l, for $l \in I_M$, and extract the permutation π_l used when forming L_l from L_{l-1}. Since the simulator knows the message m_i sent by each sender S_i, this allows the simulator to simulate decryption correctly without the private key. Below we make this idea precise.

Denote by $P^*_{\mathrm{prp}_l}$ the malicious prover defined in Claim 6. Denote by $H^{c,s,e,*}_{2,5}$ the machine that simulates $H^{c,s}_{2,5}$ except that it interrupts the execution before Step 12, i.e., before starting the decryption phase. Then for each l such that M_l was corrupt when executing its proof of a shuffle, the extractor $\mathcal{X}_{P^*_{\mathrm{prp}_l}}$ is invoked on input $(\mathbf{\Gamma}, \overline{g}, r_{\mathrm{p}_l})$, and the result is denoted by $(f_l, \{r_{l,i}\}_{i=1}^N, \pi_l)$. Finally, π' is defined as the product $\pi_k \pi_{k-1} \cdots \pi_1$, and the simulation is continued using $\pi = \pi'$ in the simulation of the joint re-encryption instead of choosing π randomly. We transform the machine into a polynomial time machine $H^{c,s,e}_{2,5}$ by bounding its running time by a polynomial $p_e(\kappa) = 4p_s^2(\kappa) T_{P^*_{\mathrm{prp}}}(\kappa) k^2$, where $p_s(\kappa)$ is a polynomial to be defined later. If the simulation is not finished, it outputs $*$. Define $H^{c,s,e,*}_{2,5,1}$ and $H^{c,s,e}_{2,5,1}$ correspondingly.

Before we analyze these machines we need a technical observation. Consider a sequence $Z_1, Z_2, \ldots, Z_k$ of families $Z_l = \{Z_{l,\kappa}\}_{\kappa\in\mathbb{N}}$ of binary random variables. Denote by $S_{\text{good}}(f(\kappa))$ the set of families of outcomes $(z_1, \ldots, z_k)$ such that

$$\Pr[Z_l = 1 \mid (Z_1, \ldots, Z_{l-1}) = (z_1, \ldots, z_{l-1})] \geq \frac{1}{f(\kappa)}$$

for all l such that $z_l = 1$ and κ in some infinite index set $\mathcal{N}$. Then we have the following claim.

Claim 9. There exists an infinite index set $\mathcal{N}$ such that for every polynomial $p(\kappa)$, and $\kappa \in \mathcal{N}$, we have

$$\Pr[(Z_1, Z_2, \ldots, Z_k) \in S_{\text{good}}(kp(\kappa))] \geq 1 - \frac{1}{p(\kappa)} \ .$$

Proof. Suppose the claim is false and write $Z = (Z_1, Z_2, \ldots, Z_k)$. Then for every infinite index set $\mathcal{N}$ exists a polynomial $p(\kappa)$ and an $\kappa \in \mathcal{N}$ such that

$$1 - \frac{1}{p(\kappa)} \quad > \quad \Pr[Z \in S_{\text{good}}(kp(\kappa))] = 1 - \Pr[Z \notin S_{\text{good}}(kp(\kappa))] \ .$$

Thus, $\Pr[Z \notin S_{\text{good}}(kp(\kappa))] > \frac{1}{p(\kappa)}$, but the union bound implies that $\Pr[Z \notin S_{\text{good}}(p(\kappa))]$ is bounded from above by $k\frac{1}{p(\kappa)k} = \frac{1}{p(\kappa)}$. This is a contradiction and the claim follows. □

Claim 10.

$$\Pr[H_{2,5}^{c,s,e} = *] \quad \leq \quad \frac{1}{p_s(\kappa)} \quad \text{and} \quad \Pr[H_{2,5,1}^{c,s,e} = *] \leq \frac{1}{p_s(\kappa)} \ .$$

Proof. We prove the first inequality. The proof of the second inequality is almost identical. Denote by Z_l an indicator variable for the event that M_l produces a valid proof in the simulation. Then we can define $T_{\text{good}}(f(n))$ as the set of lists $(\mathbf{\Gamma}, \overline{g}, r_{\text{p}_k}, c_k, c'_k)$ such that the outcome $(z_1, \ldots, z_k)$ of $(Z_1, \ldots, Z_k)$ is contained in $S_{\text{good}}(f(n))$.

Proposition 9.23 implies that the expected running time of $H_{2,5}^{c,s,e,*}$ is bounded by $2p_s(\kappa)T_{P^*_{\text{prp}}}(\kappa)k^2$ whenever $(\mathbf{\Gamma}, \overline{g}, r_{\text{p}_k}) \in T_{\text{good}}(2kp_s(\kappa))$. Markov's inequality, Proposition 9.23 and the union bound implies that

$$\Pr[H_{2,5}^{c,s,e} \neq * \mid (\mathbf{\Gamma}, \overline{g}, r_{\text{p}_k}) \in T_{\text{good}}(2kp_s(\kappa))] \geq 1 - \frac{1}{2p_s(\kappa)} - \epsilon(\kappa) \ ,$$

for some negligible function $\epsilon(\kappa)$. It follows from Claim 9 that $\Pr[(\mathbf{\Gamma}, \overline{g}, r_{\text{p}_k}) \in T_{\text{good}}(2kp_s(\kappa))] \geq 1 - \frac{1}{2p_s(\kappa)}$. Thus, we have $\Pr[H_{2,5}^{c,s,e} \neq *] \leq \frac{1}{p_s(\kappa)}$ as required. □

Claim 11.

$$|\Pr[H_{2,5}^{c,s} = 1 \mid H_{2,5}^{c,s} \neq \perp] - \Pr[H_{2,5}^{c,s,e} = 1 \mid H_{2,5}^{c,s,e} \neq \perp]| \leq \frac{2}{p_s(\kappa)}$$

$$|\Pr[H_{2,5,1}^{c,s} = 1 \mid H_{2,5,1}^{c,s} \neq \perp] - \Pr[H_{2,5,1}^{c,s,e} = 1 \mid H_{2,5,1}^{c,s,e} \neq \perp]| \leq \frac{2}{p_s(\kappa)}$$

Proof. Consider the simulation of $H_{2,5}^{c,s,e}$. During the re-encryption-permutation phase all cleartexts are zero. The method used by M_j for re-encryption gives cryptotexts with distribution statistically close the the distribution of cryptotexts re-encrypted using the standard method. Note that there exists for each permutation $\pi_j \in \Sigma_{N'}$ values $r_{j,i} \in [0, 2^{\kappa+\kappa_r} - 1]$ such that

$$L_j \quad = \quad \{u_{j,i}\}_{i=1}^{N'} = \mathsf{Sort}(\{\mathsf{g}^{r_{j,i}} u_{j-1,i}^2 \bmod \mathsf{N}^2\}_{i=1}^{N'}) \ .$$

Thus, the distribution of the cryptotexts for any pair of fixed permutations only differ negligibly. Furthermore, the proof of M_j is simulated and therefore contains no additional information on the permutation π_j used by M_j. It is at this point the zero-knowledge simulator of the proof of knowledge of correct re-encryption-permutation is essential.

Thus, the distribution of $H_{2,5}^{c,s,e}$ conditioned on not outputting $*$ is statistically close to the distribution of $H_{2,5}^{c,s}$. The first inequality then follows from Claim 10 and the union bound.

Consider now the simulation of $H_{2,5,1}^{c,s,e}$. Note that since K_0 and $(\mathsf{g}')^{\bar{b}_l} \bar{K}'_l$ are encryptions of 0 for $l \notin I_M$, the cryptotexts $w'_{l,i}$ and $w_{l,i}$ contain no information on π' for $l \notin I_M$. Thus, each transcript of a simulation of $H_{2,5,1}^{c,s,e}$ that does not result in the output $*$ is also the transcript of a simulation of $H_{2,5,1}^{c,s}$. The second inequality then follows from Claim 10 and the union bound. □

Denote by $H_{2,5}^{c,s,e,d}$ the machine that simulates $H_{2,5}^{c,s,e}$ except for the following changes. It defines $\mathsf{v} = \mathsf{g}^{m'} r^{2\mathsf{N}}$ using a random $m' \in \mathbb{Z}_{\mathsf{N}}$, it never defines d_j, and it defines

$$\mathsf{v}_j = \mathsf{g}^{m'} / \prod_{l=1}^{k} \mathsf{v}_l \ .$$

The reason this is a good idea is that $\mathsf{v}^{\mathsf{d}} = \mathsf{g}^{m'}$ and $\mathsf{v}^{\mathsf{d}_j} = \mathsf{v}_j$, where $\mathsf{d} = \sum_{l=1}^{k} \mathsf{d}_l \bmod \mathsf{Nf}$. Thus, we have made sure that v and v_j are distributed exactly as in $H_{2,5}^{c,s,e}$. It also chooses $\mathsf{d}_{j,l} \in \mathbb{Z}_q$ randomly instead of evaluating a polynomial. The final change takes place in the decryption phase, Steps 12-15. $\mathcal{S}$ instructs M_j to compute $v_{j,i}$ by

$$v_{j,i} = \mathsf{g}^{m_{\pi'(i)}} / \prod_{l \neq j} v_{l,i} \ .$$

Define $H_{2,5,1}^{c,s,e,d}$ correspondingly.

Claim 12. $\Pr[H_{2,5}^{c,s,e,d} = 1 \mid H_{2,5}^{c,s,e,d} \neq \perp] = \Pr[H_{2,5}^{c,s,e} = 1 \mid H_{2,5}^{c,s,e} \neq \perp]$, and $\Pr[H_{2,5,1}^{c,s,e,d} = 1 \mid H_{2,5,1}^{c,s,e,d} \neq \perp] = \Pr[H_{2,5,1}^{c,s,e} = 1 \mid H_{2,5,1}^{c,s,e} \neq \perp]$.

Proof. It suffices to note that although the private key d is not used anymore the values of v, v_j or $\mathsf{v}_{j,i}$ in any simulation are identically distributed to the values that would have resulted if d would have been used. Recall that the proof of correctness is already simulated. □

We are now ready to conclude the proof of Claim 2. Suppose first that

$$|\Pr[H_{2,5}^{c,s,e,d} \neq \perp] - \Pr[H_{2,5,1}^{c,s,e,d} \neq \perp]|$$

is non-negligible. Then we define a distinguisher D that is the naive distinguisher except that it simulates $H_{2,5,1}^{c,s,e,d}$ instead of $H_{2,5,1}$ and outputs 1 if $H_{2,5,1}^{c,s,e,d}$ outputs $\perp$ and otherwise 0. It follows that D breaks the polynomial indistinguishability of the Paillier cryptosystem, which is equivalent to breaking the DCR-assumption. Thus, $|\Pr[H_{2,5}^{c,s,e,d} \neq \perp] - \Pr[H_{2,5,1}^{c,s,e,d} \neq \perp]|$ is negligible.

Next we note that $\Pr[H_{2,5}^{c,s,e,d} \neq \perp] \leq \frac{1}{2} + \epsilon(\kappa)$, for some negligible function $\epsilon(\kappa)$, since M_j is chosen randomly, and the only parts of the simulation that contains any information on the identity of M_j are the simulated proofs in the re-encryption-permutation phase and decryption-phase, and these are statistically close to real executions of these protocols by Proposition 9.22 and Proposition 11.9. Thus, $\Pr[H_{2,5}^{c,s,e,d} \neq \perp], \Pr[H_{2,5,1}^{c,s,e,d} \neq \perp] \leq \frac{1}{4}$

Finally, if we define $p_s(\kappa) = 8\kappa^{c_0}$ we have from Claim 7, 8, 11, and 12 that

$$\Pr[H_{2,5,1}^{c,s,e,d} = 1 \mid H_{2,5,1}^{c,s,e,d} \neq \perp] - \Pr[H_{2,5}^{c,s,e,d} = 1 \mid H_{2,5}^{c,s,e,d} \neq \perp] \geq \frac{1}{2\kappa^{c_0}} \ .$$

Combined with the above two observations this means that $\Pr[H_{2,5,1}^{c,s,e,d} = 1] - \Pr[H_{2,5}^{c,s,e,d} = 1]$ is non-negligible. We define a distinguisher D that is identical to the naive distinguisher except that it simulates $H_{2,5,1}^{c,s,e,d}$ instead of $H_{2,5,1}$. It follows that D breaks the polynomial indistinguishability of the Paillier cryptosystem, which is equivalent to breaking the DCR-assumption. This is a contradiction and Claim 2 must be true.

11.8 On Adaptively Secure El Gamal Based Mix-Nets

The underlying complexity assumption of the Paillier cryptosystem has not undergone the same scrutiny as the decision Diffie-Hellman (DDH) assumption on which the security of the El Gamal cryptosystem rests. Furthermore, distributed key generation is simpler when using El Gamal. Thus, there are good reasons to investigate if one can form an adaptively secure mix-net based on El Gamal.

Recall that El Gamal is employed in a group G_q of prime order with generator g. A private key is an element $x \in \mathbb{Z}_q$ and the corresponding public key consists

of g and $y = g^x$. To encrypt a message $m \in G_q$ a random exponent $r \in \mathbb{Z}_q$ is chosen and $(u, v) = E_{(g,y)}(m, r) = (g^r, y^r m)$ is computed. To decrypt (u, v) one computes $D_x(u, v) = vu^{-x} = m$. A cryptotext (u, v) can be randomly re-encrypted by computing $(g^s u, y^s v)$ for a random s.

On a very high level, two problems must be dealt with to translate our approach to the El Gamal cryptosystem. Firstly, there must be a way for the senders to submit their cryptotexts in such a way that simulated honest senders can later open their cryptotexts to any cleartext. Secondly, there must be a way to jointly insert the correct outputs before the final shuffled list of cryptotexts is decrypted.

11.8.1 An Impractical Construction

A simple method to solve the first problem is to change the encoding of messages. Instead of encoding a message as an element $m \in G_q$, the message is encoded as an element $m \in \mathbb{Z}_q$ and then $m' = g^m$ is encrypted. The key generator chooses a joint public key $y = g^x$, where $x = \sum_{j=1}^{k} x_j$ and $y_j = g^{x_j}$, and each x_j is secretly shared in exactly the same way as in the Paillier setting. An additional joint public key $y' = g^{x'}$ is also generated. The keys y and y' play the same roles as N and N' in the Paillier setting. Two cryptotexts $(g_0, y_0) = E_{(g,y)}(g^0, R_0) = (g^{R_0}, y^{R_0})$ and $(g_1, y_1) = E_{(g,y)}(g^1, R_0) = (g^{R_1}, gy^{R_1})$ for random $R_0, R_1 \in \mathbb{Z}_q$ are also generated that play the same role as K_0 and K_1 in the Paillier setting.

Each sender is given an encryption $(g'_i, y'_i) = (g^{R'_i}, (y')^{R'_i} a_i)$ of a random element $a_i \in G_q$. This plays the role of K'_i in the Paillier setting. It chooses $b_i \in G_q$ and $r_i, r'_i \in \mathbb{Z}_q$ randomly and computes

$$\begin{aligned}(u_i, v_i) &= (g_1^{m_i} g^{r_i}, y_1^{m_i} y^{r_i}) = (g^{R_0 m_i + r_i}, y^{R_0 m_i + r_i} g^{m_i}) \ , \quad \text{and} \\ (u'_i, v'_i) &= ((g'_i)^{m_i} g^{r'_i}, (y'_i / b_i)^{m_i} (y')^{r'_i}) \ .\end{aligned}$$

Then it submits $(b_i, (u_i, v_i), (u'_i, v'_i))$ and proves that it uses the same m_i when forming (u_i, v_i) and (u'_i, v'_i).

The distributions of the keys (g_0, y_0) and (g_1, y_1) are indistinguishable to the adversary. During simulation we switch the definitions of these keys. Furthermore, simulated senders chooses $b_i = a_i$ and $m_i = 0$, which allows the ideal adversary to later compute a plausible history tape by setting $\bar{r}_i = r_i - R_1 m_i \bmod q$ and $\bar{r}'_i = r'_i - R'_i m_i \bmod q$, and opening the proof similarly.

The messages m_i submitted by corrupted senders are extracted by computing $m_i = \log_{b_i/a_i}(D_{x'}(u'_i))$. By the semantic security of the cryptosystem the adversary can not guess a_i, so $b_i/a_i \neq 1$ and the extracted message is well defined and equal to the message encrypted in u_i with overwhelming probability. To compute the logarithm $m_i = \log_{b_i/a_i}(D_{x'}(u'_i))$ only a small number of values of m_i can be allowed. One way to restrict the set of values is to let the prover prove that m has small absolute value when viewed as an integer. Another way to restrict the set of values is to let the prover prove explicitly that m is one of a fixed number of different values using standard Schnorr-style proofs.

The correct messages can be replaced in a joint re-encryption step as follows. Two $(k'-1)$-degree polynomials f_0 and f_1 are defined by the key generator such that $f_0(0) = 0$ and $f_1(0) = 1$. Then each mix-server M_l is given $f_0(l)$ and $f_1(l)$ and Pedersen commitments $H_{j,0} = g^{f_0(l)}h^{z_{j,0}}$ and $H_{j,1} = g^{f_1(l)}h^{z_{j,1}}$ are made public by the key generator. To insert the correct messages each mix-server secret shares random $(w'_{j,i,1}, w'_{j,i,2}) \in G_q^2$ using an adaptively UC-secure verifiable secret sharing scheme [4]. All random pairs are recovered. Each mix-server computes $(w_{j,i,1}, w_{j,i,2}) = (g^{s_{j,i}}u_{k,i}^{f_1(j)}(\prod_{j=1}^{k} w'_{j,i,1})^{f_0(j)}, y^{s_{j,i}}v_{k,i}^{f_1(j)}(\prod_{j=1}^{k} w'_{j,i,2})^{f_0(j)})$ and proves that it did so for some $s_{j,i} \in \mathbb{Z}_q$ relative $H_{j,0}$ and $H_{j,1}$. Finally, all mix-servers compute $(u_{k+1,i}, v_{k+1,i}) = (\prod_{j\in I} w_{j,i,1}^{\prod_{l\neq j}\frac{l}{l-j}}, \prod_{j\in I} w_{j,i,2}^{\prod_{l\neq j}\frac{l}{l-j}})$, where I is the lexicographically first set of indices j such that the proof of M_j is valid.

In the real protocol this amounts to a elaborate way to re-encrypt the list of cryptotexts. In the simulation the ideal adversary defines the polynomials such that $f_0(0) = 1$ and $f_1(0) = 0$ instead. After all corrupted mix-servers have handed their pairs $(w'_{j,i,1}, w'_{j,i,2})$ to the ideal secret sharing functionality, the secret sharing functionality is instructed to redefine the pairs $(w'_{j,i,1}, w'_{j,i,2})$ of the simulated honest mix-servers such that they are random, but under the restriction that $(\prod_{j=1}^{k} w'_{j,i,1}, \prod_{j=1}^{k} w'_{j,i,2})$ is an encryption of $g^{m_{\pi(i)}}$ for a randomly chosen permutation $\pi \in \Sigma_N$ using the public key (g, y).

To decrypt the cryptotexts in L_{k+1} each mix-server computes $u_{k+1,i}^{x_j}$ and proves that it does this correctly relative y_j. If not, its part x_j of the private key is recovered and the computation is done openly.

The obvious problem with the outlined solution is that only very short messages can be submitted. Longer messages can be handled by parallel mixing of several cryptotexts, but this is not efficient.

11.8.2 A Practical Construction

A more practical "solution" to the submission problem is to let the senders compute the proof needed using a random oracle, and to stipulate that they delete the randomness used in encryption and proving before submitting their input. In other words the senders would *force erasures* and the problem of constructing plausible history tapes for senders disappears.

More precisely, a sender encodes its message as an element $m_i \in G_q$, chooses $r_i, r'_i \in \mathbb{Z}_q$, forms two cryptotexts

$$(u_i, v_i) = (g^{r_i}, y^{r_i}m_i) \quad \text{and} \quad (u'_i, v'_i) = (g^{r'_i}, (y')^{r'_i}m_i) \ ,$$

and computes a proof σ that it uses the same message in both cryptotexts using a standard Σ-proof in the random oracle model. Then it erases its history of computation and it submits $(u_i, v_i, u'_i, v'_i, \sigma)$.

Honest senders are simulated by setting $m_i = 1$. The adversary can only corrupt a sender before it has started the computation of its submission or after it has

completed it and erased the randomness used in this process. Extraction from corrupted senders S_i is done by computing $m_i = v'_i(u'_i)^{-x'}$.

The problem of correcting the input is solved in the same way as is outlined above except that $(w'_{j,i,1}, w'_{j,i,2})$ is chosen randomly for simulated mix-servers under the restriction that $(\prod_{j=1}^{k} w'_{j,i,1}, \prod_{j=1}^{k} w'_{j,i,2})$ is an encryption of $m_{\pi(i)}$.

Although the sketched solution is practical it is only heuristically secure due to the dependence on the random oracle model. Furthermore, it is questionable if the assumption that all senders manage to erase their randomness is realistic.

Chapter 12

Conclusions of Part II

The need for a rigorous treatment of mix-nets has been illustrated by several practical attacks on a construction in the literature. To rectify this the first definition of an ideal mix-net functionality in the universally composable security framework has been presented. The functionality is simple and corresponds to a natural trusted party. We have also presented the first efficient mix-nets with complete security analyses, in any model. This shows that our definition of security of a mix-net is not only natural, but it can be satisfied in an efficient way. In doing this we have introduced a new method to construct a proof of a shuffle that seems as efficient as the two previously known approaches. There are at least two interesting open problems.

The adaptively secure mix-net we describe is based on the security of the Paillier cryptosystem. This cryptosystem has not undergone the same scrutiny as the older El Gamal cryptosystem. Therefore it is interesting to investigate if it is possible to construct an efficient El Gamal based adaptively secure mix-net without erasures.

We have illustrated two methods of extracting the cleartexts during submission. The first is non-interactive, but the work of the sender is linear in the number of mix-servers. The second is interactive, or non-interactive in the random oracle model, but very efficient. An interesting open problem is if it is possible to use a distributed CCA2-secure cryptosystem for extraction in the submission phase. This would give a non-interactive submission phase in the plain model. We believe that this is possible.

Part III

Hierarchical Group Signatures

Chapter 13

Introduction of the New Notion and Definitions

In this chapter we introduce the notion of a hierarchical group signatures. There exists no previous work on this notion, but it is a strict generalization of group signatures which have been studied extensively. Therefore, we give a detailed account on previous work on group signatures and other related notions. Then we describe the parties of a hierarchical group signature system and give formal definitions. We also discuss informally alternative definitions and the difficulties involved in constructing a hierarchical group signature schemes. Finally, we prove a characterization of anonymous cryptosystems that is used in the next chapter. This chapter is based on the paper by Trolin and Wikström [141].

13.1 Related Work

The concept of group signatures was introduced by Chaum and van Heyst [48] in 1991. Recall from Section 1.4.2 that group signature schemes generalize digital signature schemes. There is a group manager and several signers. Each signer holds a private signing key and the group manager holds a private opening key. There is also a joint public key. Anybody holding the public key can verify the correctness of a signature, but without the group manager private key it is infeasible to determine who computed a given group signature. Thus, any outsider can verify that some signer belonging to the group computed the signature, but nothing else. The group manager on the other hand can use its private key to open any valid signature and discover the identity of the signer that produced it.

The original scheme in [48] and the group signature schemes that followed [49, 32] all have the property that the complexity of the scheme grows with the number of parties. In [38] Camenisch and Stadler presented a system where the key does not grow with the number of parties. This system, however, relies on a non-standard number-theoretic assumption. The assumption was actually found to

be incorrect and was modified in [8]. An efficient system whose security rests on the strong RSA-assumption and the decision Diffie-Hellman assumption was presented by Camenisch and Michels in 1998 [35]. This system was improved in [7]. The currently most efficient scheme that is secure under standard assumptions was given by Camenisch and Groth [33]. More efficient schemes do exist [27, 34], but they are based on bilinear maps and thus relies on less well-studied assumptions for security.

A related notion is traceable signatures introduced by Kiayias et al. [93], where signatures belonging to a member can be opened, or traced, in a distributed way without revealing the group secret.

Bellare et al. [17] give a definitional framework for group signatures for static groups, i.e., when the set of members cannot be changed after the initial setup. They also present a scheme that is secure according to their definitions under general assumption. Kiayias and Yung [94] define security for dynamic groups and prove that a modification of [7] is secure under these definitions. Independently, Bellare et al. [19] extend the definitions of [17] in a similar way to handle dynamic groups, and present a scheme that is secure under general assumptions.

The first of our constructions which is secure under general assumptions can be seen as a generalization of the construction in [17].

In [8] the concepts of multi-group signatures and subgroup signatures are described, and in [96] a system for hierarchical multi-groups is given. It is worthwhile to consider the differences between these concepts and hierarchical group signatures introduced here.

Subgroup signatures make it possible for an arbitrary number i of signers to produce a joint signature which can be verified to stem from precisely i distinct group members, without disclosing the identity of the individual signers.

Multi-group signature schemes allow a signer who is a member of two groups to produce a signature that shows membership of either both groups or just one of them. In hierarchical multi-groups a signer who is a member of a supergroup with subgroups can produce a signature that reveals membership either of the supergroup or of a subgroup of his choice. Thus, the signer decides to some extent the amount of information about its identity that is made public.

Recall from Section 1.4.4 that in a hierarchical group signature scheme the parties are organized in a tree with group managers as internal nodes and signers as leaves. As for group signatures no outsider can determine from a signature which signer produced it. A group manager can from a signature determine if the signer that produced the signature belongs to the subtree of which it is the root, and if so determine to which of its immediate subtrees the signer belongs, but nothing else.

In both subgroup signatures and multi-group signatures there are several group managers, but any group manager that opens a valid signature learns the identity of the signer. In hierarchical group signatures on the other hand the opening procedure is hierarchical. Both the subgroup property and the multi-group property are independent from the hierarchical property we study.

The connection between group signatures and anonymous payment schemes is quite natural and has been studied before. In [101] a system for electronic cash

based on group signatures is given by Lysyanskaya and Ramzan.

Group signatures, and especially hierarchical group signatures, should not be confused with zero-knowledge sets as described in [104]. Zero-knowledge sets enables a prover to commit to a set S. Given x he can then prove $x \in S$ or $x \notin S$, whichever is true, without disclosing anything else about S. For zero-knowledge sets the prover has the necessary information to produce a proof of membership for any element in the set. With group signatures on the other hand the set of members may be public, and the signer proves that it belongs to this set.

13.2 The Parties

There are two types of parties: signers denoted S_α for α in some index set $\mathcal{I}$, and group managers denoted M_α for indices α described below. The parties form a tree T, where the signers are leaves and the group managers are inner nodes. We denote by $\mathcal{L}(T)$ its set of leaves and by $\mathcal{V}(T)$ the set of all vertices. The indices of the group managers are formed as follows. If a group manager manages a set of signers S_α for $\alpha \in \beta \subset \mathcal{I}$ we denote it by M_β. This corresponds to M_β having S_α for $\alpha \in \beta$ as children. If a group manager manages a set of group managers $\{M_{\beta_1}, \ldots, M_{\beta_l}\}$ we denote it by M_γ where γ is the set of sets $\{\beta_1, \ldots, \beta_l\}$. This corresponds to M_γ having M_{β_i} for $i = 1, \ldots, l$ as children. Let M_ρ denote the root group manager. We define the root group manager to be at depth 0 and assume that all leaves in the tree are at the same depth. This is illustrated in Figure 1.3 in the introduction. We reproduce this figure below for convenience.

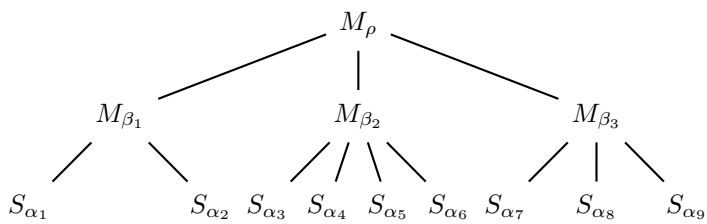

Figure 1.3: A tree of group managers and signers, where $\rho = \{\beta_1, \beta_2, \beta_3\}$, $\beta_1 = \{\alpha_1, \alpha_2\}$, $\beta_2 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6\}$, and $\beta_3 = \{\alpha_7, \alpha_8, \alpha_9\}$.

Note that standard group signatures correspond to having a single group manager $M_{[1,l]}$ that manages all signers $S_1, \ldots, S_l$.

13.3 The Definition of Security

Bellare et al. [17] give a definition of a group signature scheme, but more importantly they argue that two properties of group signatures, full anonymity and full traceability, imply any reasonable security requirements one can expect from a group signature scheme.

We follow their definitional approach closely and develop definitions that are proper generalizations of the original.

The idea is that the managers and signers are organized in a tree T, and we wish to associate with each node and leaf α a public value $hpk(\alpha)$ and a private value $hsk(\alpha)$.

Definition 13.1 (Hierarchical Group Signature). A *hierarchical group signature scheme* $\mathcal{HGS} = (\mathsf{HKg}, \mathsf{HSig}, \mathsf{HVf}, \mathsf{HOpen})$ consists of four polynomial-time algorithms

1. The probabilistic *key generation algorithm* HKg takes as input $(1^\kappa, T)$, where T is a tree of size polynomially bounded in κ with all leaves at the same depth, and outputs a pair of maps $hpk, hsk : \mathcal{V}(T) \to \{0,1\}^*$.

2. The probabilistic *signature algorithm* HSig takes as input a message m, a tree T, a public map hpk, and a private signing key $hsk(\alpha)$, and returns a signature of m.

3. The deterministic *signature verification algorithm* HVf takes as input a tree T, a public key map hpk, a message m and a candidate signature σ of m and returns either 1 or 0.

4. The deterministic *opening algorithm* HOpen takes as input a tree T, a public map hpk, a private opening key $hsk(\beta)$, a message m, and a candidate signature σ. It outputs an index $\alpha \in \beta$ or $\perp$.

In the definition of HSig above, it is assumed that it is possible to verify in polynomial time given the public tree hpk, a private key $hsk(\alpha)$ and an index α', if $\alpha = \alpha'$. This is the case for the construction in [17]. We assume that hpk and hsk map any input that is not a node of T to $\perp$ and that $\mathsf{HOpen}(\cdot, \cdot, \perp, \cdot, \cdot) = \perp$.

We need to define what we mean by security for a hierarchical group signature scheme. We begin with anonymity. Consider Figure 13.1, where two signers $S_{\alpha^{(0)}}$ and $S_{\alpha^{(1)}}$ are marked. Assume that a signature σ of a message m is given and that it is computed by either $S_{\alpha^{(0)}}$ or $S_{\alpha^{(1)}}$. Then any group manager on the path leading from $S_{\alpha^{(0)}}$ or $S_{\alpha^{(1)}}$ to their first common ancestor can determine who produced the signature. In the figure those group managers are marked black. In the definition of anonymity we capture the property that unless the adversary corrupts one of these group managers, it cannot determine whether $S_{\alpha^{(0)}}$ or $S_{\alpha^{(1)}}$ signed the message, even if the adversary is given the private keys of all signers and is allowed to select $\alpha^{(0)}$, $\alpha^{(1)}$ and the message m that is signed.

We define Experiment 13.2 to formalize these ideas. Throughout the experiment the adversary has access to an $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle. At the start of the experiment the adversary is given the public keys of all parties and the private keys of all signers. Then it can adaptively ask for the private keys of the group managers. At some point it outputs the indices $\alpha^{(0)}$ and $\alpha^{(1)}$ of two leaves and a message m. The $\mathsf{HSig}(\cdot, T, hpk, hsk(\alpha^{(b)}))$ oracle then computes the signature of m

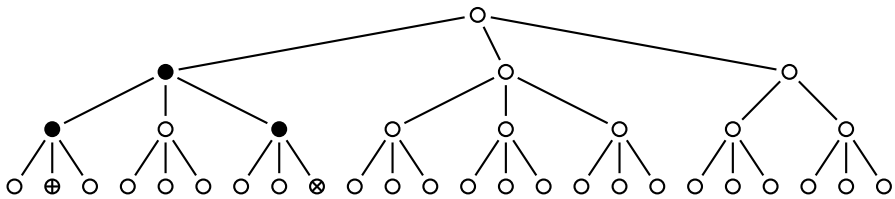

Figure 13.1: Nodes in black represent group managers able to distinguish between signatures by $S_{\alpha^{(0)}}$ and $S_{\alpha^{(1)}}$, the two leaves marked $\oplus$ and $\otimes$ respectively.

and hands it to the adversary. The adversary finally outputs a guess d of the value of b. If the scheme is anonymous the probability that $b = d$ should be negligibly close to $1/2$ when b is a randomly chosen bit. The labels corrupt, choose and guess below allows the adversary to distinguish between the phases of the experiment.

Experiment 13.2 (Hierarchical Anonymity, $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$).

$(hpk, hsk) \leftarrow \mathsf{HKg}(1^\kappa, T)$; state $\leftarrow (hpk, hsk(\mathcal{L}(T)))$; $\mathcal{C} \leftarrow \emptyset$; $\alpha \leftarrow \emptyset$;
Do
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
  $(\text{state}, \alpha) \leftarrow A^{\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)}(\mathsf{corrupt}, \text{state}, hsk(\alpha))$
While $(\alpha \in \mathcal{V}(T) \setminus \mathcal{C})$
$(\text{state}, \alpha^{(0)}, \alpha^{(1)}, m) \leftarrow A^{\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)}(\mathsf{choose}, \text{state})$
$\sigma \leftarrow \mathsf{HSig}(m, T, hpk, hsk(\alpha^{(b)}))$
$d \leftarrow A^{\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)}(\mathsf{guess}, \text{state}, \sigma)$

Let B be the set of nodes on the paths from $\alpha^{(0)}$ and $\alpha^{(1)}$ up to their first common ancestor α_t excluding $\alpha^{(0)}$ and $\alpha^{(1)}$ but including α_t, i.e., the set of nodes $\alpha^{(0)}_l$, $\alpha^{(1)}_l$, $l = t, \ldots, \delta - 1$, such that

$$\alpha^{(0)} \in \alpha^{(0)}_{\delta-1} \in \alpha^{(0)}_{\delta-2} \in \ldots \in \alpha^{(0)}_{t+1} \in \alpha_t \ni \alpha^{(1)}_{t+1} \ni \ldots \ni \alpha^{(1)}_{\delta-2} \ni \alpha^{(1)}_{\delta-1} \ni \alpha^{(1)} \ .$$

If $B \cap \mathcal{C} \neq \emptyset$ or if A asked its $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle a query $(\alpha^{(0)}_l, m, \sigma)$ or $(\alpha^{(1)}_l, m, \sigma)$ return 0. Otherwise return d.

No generality is lost by having a corrupt phase only before σ is computed. The reason for this is that before A receives σ, it has decided on $\alpha^{(0)}$ and $\alpha^{(1)}$ and can corrupt any group manager not on the path from $\alpha^{(0)}$ or $\alpha^{(1)}$ respectively.

Consider the above experiment with a depth one tree T with root ρ. In that case we may assume that $hsk(\rho)$ is never handed to the adversary, since the adversary fails in that case anyway. Similarly the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle reduces to the Open oracle in [17]. Thus, our experiment reduces to the experiment for full anonymity given in [17] where the adversary gets the private keys of all signers, but only the public key of the group manager.

Next we consider how the notion of full traceability can be defined in our setting. Full traceability as defined in [17] is similar to security against chosen message attacks as defined by Goldwasser, Micali and Rivest [78] for signatures. Their definition is given in Section 2.6.1.

The only essential difference is that the group manager must always be able to open a signature and identify the signer. In our setting this amounts to the following. Given a signature deemed valid by the HVf algorithm, the root should always be able to identify the child directly below it of which the signer is a descendant. The child should have the same ability for the subtree of which it is a root and so on until the child itself is a signer.

Again we define an experiment consisting of two phases. The adversary is given the private keys of all group managers and has access to a signature oracle, and adaptively chooses a set of signers to corrupt. Then in a second phase the adversary outputs a message m and a signature σ. If σ is a valid signature of m and the signer cannot be traced, or if the signature is traced to a non-corrupted signer S_α and the adversary has not queried its signature oracle $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$ on (m, α), the adversary has succeeded and the experiment outputs 1. The other way the adversary can succeed is by constructing a signature that does trace correctly, but has the property that some group manager not belonging to the path also gets a valid index corresponding to one of its children if it opens the signature. If none of the above is the case it outputs 0. Thus, the distribution of the experiment should be negligibly close to 0 for all adversaries if the scheme is secure.

Experiment 13.3 (Hierarchical Traceability, $\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$).

$(hpk, hsk) \leftarrow \mathsf{HKg}(1^\kappa, T)$; state $\leftarrow (hpk, hsk(\mathcal{V}(T) \backslash \mathcal{L}(T)))$; $\mathcal{C} \leftarrow \emptyset$; $\alpha \leftarrow \emptyset$;
Do
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
  $(\text{state}, \alpha) \leftarrow A^{\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))}(\mathsf{corrupt}, \text{state}, hsk(\alpha))$
While $(\alpha \in \mathcal{L}(T) \setminus \mathcal{C})$
$(m, \sigma) \leftarrow A^{\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))}(\mathsf{guess}, \text{state})$

If $\mathsf{HVf}(T, hpk, m, \sigma) = 0$ return 0. Define $\alpha_0 = \rho$ and define α_l for $l = 1, \ldots, \delta$ by $\alpha_l = \mathsf{HOpen}(T, hpk, hsk(\alpha_{l-1}), m, \sigma)$. If $\alpha_l = \perp$ for some $0 < l \leq \delta$ return 1. If $\alpha_\delta \notin \mathcal{C}$ and the $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$ oracle did not get a query (m, α_δ) return 1. If there exists an index $\alpha \in \mathcal{V}(T)$ such that $\alpha \neq \alpha_l$ for $l = 1, \ldots, \delta$ and $\alpha' = \mathsf{HOpen}(T, hpk, hsk(\alpha), m, \sigma)$ and $\alpha' \in \alpha$, then return 1. Otherwise return 0.

Remark 13.4. The above definition differs from the one in [141] in that the requirement on the special index α "outside" the path has been added. The original definition guarantees that the group managers along the path to the producer of a signature can open their part of the signature. Unfortunately, it does not prohibit the construction of signatures such that if two distinct group managers M_α and M_β on the same level open a signature they both get indices $\alpha' \in \alpha$ and $\beta' \in \beta$. Naturally, we expect that only one of α' and β' can be different from $\perp$. Thus,

although the original definition guarantees that the signer can be identified, a group manager can not fully trust the result of the opening algorithm unless it communicates with all group managers on the path from itself to the root. This goes against the non-interactivity of signature schemes.

The definition above on the other hand not only requires that the group managers along the path to the signer can open a signature and recover the index of the sub-group manager to which the signer belongs, but also that if any group manager that is not on the path to the signer opens the signature then the result is $\perp$.

Consider the experiment above with a depth one tree. This corresponds to giving the adversary the private key of the group manager, and letting it adaptively choose additional signing keys. Furthermore, the $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$ oracle reduces to the GSig oracle in [17]. Thus, the definition reduces to the definition of full traceability in [17].

The advantages of the adversary in the experiments are defined by

$$\begin{aligned}\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa,T) &= |\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},A}(\kappa,T)=1]-\Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},A}(\kappa,T)=1]|\ , \text{ and}\\ \mathsf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa,T) &= \mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa,T)\ .\end{aligned}$$

Definition 13.5 (Security of Hierarchical Group Signatures). A hierarchical group signature scheme $\mathcal{HGS} = (\mathsf{HKg}, \mathsf{HSig}, \mathsf{HVf}, \mathsf{HOpen})$ is secure if for all trees T of polynomial size in κ with all leaves at the same depth, and all adversaries $A \in \mathrm{PPT}^*$ the sum $\mathsf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa,T) + \mathsf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa,T)$ is negligible.

An ordinary signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$, with key generator Kg, signature algorithm Sig, and verification algorithm Vf, can be viewed as a hierarchical group signature scheme $(\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{HOpen})$ of depth 0. Definition 13.3 reduces to the definition of security against chosen message attacks as defined by Goldwasser, Micali, and Rivest [78].

Remark 13.6. Formally, only the private key of a corrupted group manager or signer is handed to the adversary in the definitions above. Thus, the definition captures a model where the signers always erase the randomness that is used to construct signatures.

13.4 Alternative Definitions

Above we define a hierarchical group signature scheme such that the group managers are organized in a tree where all leaves are at the same depth. Furthermore, a group manager can by looking at a signature decide whether the signer belongs to it or not without any interaction with other group managers. Several other variants are possible. Below we discuss some of these variants informally.

Trees with leaves on different depths could be considered. Any such tree can clearly be replaced by a tree with all leaves at the same depth by inserting dummy

group managers in between signers and their immediate parents until all signers are at the same depth.

We could let group managers sign on behalf of its group. If this is needed a dummy signer that corresponds to the group manager is added. Depending on if the parent of the group manager should be able to distinguish between a signature of the group manager itself and its children or not, the signer is added as a child to the group manager's parent or itself. This may give a tree with leaves on different depths, in which case the transformation described above is applied.

We could consider a forest of trees, i.e., there could be several roots. Such a scheme can be simulated in our definition by first joining the trees into a single tree by adding a root and then disposing of the private root key.

The group managers could be organized in a directed acyclic graph (DAG), e.g., two group managers could share a common subtree. This would give alternative paths to some signers. There may be situations where this is advantageous, but the semantics of such a scheme is complex and involves many subtle issues, e.g., should all group managers of a signer get information on its identity, or should the signer decide on a path from a root and only reveal information to group managers along this path? Although we believe that the techniques we use for our constructions would be useful also for this type of scheme we do not investigate such schemes further.

Another interesting variation is to require that a group manager needs the admission and help of its ancestor to open a signature, or to help any of its children to open a signature. We believe that it is not hard to solve this problem using our methods, but we have not investigated this in detail.

13.5 The Main Difficulties

All modern group signatures are based on the idea that the signer encrypts a secret of some sort using the group manager's public key, and then proves that the resulting cryptotext is on this special form. The security of the cryptosystem used implies anonymity, since no adversary can distinguish cryptotexts of two distinct messages if they are encrypted using the same public key. We generalize this approach.

First we consider the problem of forwarding partial information on the identity of the signer to group managers without leaking information. Each group manager M_β is given a private key sk_β and a public key pk_β of a cryptosystem. We also give each signer S_α a public key pk_α that is used to identify the signer. Each signer is associated in the natural way with the path $\alpha_0, \alpha_1, \ldots, \alpha_\delta$ from the root $\rho = \alpha_0$ to the leaf $\alpha = \alpha_\delta$ in the tree T of group managers and signers. To compute a signature, the signer computes as part of the signature a chain

$$(C_0, C_1, \ldots, C_{\delta-1}) = \left(\mathsf{Enc}_{pk_{\alpha_0}}(pk_{\alpha_1}), \mathsf{Enc}_{pk_{\alpha_1}}(pk_{\alpha_2}), \ldots, \mathsf{Enc}_{pk_{\alpha_{\delta-1}}}(pk_{\alpha_\delta})\right) \ .$$

Note that each cryptotext C_l in the list encrypts the public key $pk_{\alpha_{l+1}}$ used to form the next cryptotext. The particular structure of the chain and the fact that all leaves

are on the same depth in the tree ensures that a group manager M_β on depth l can try to open a signature by decrypting C_l, i.e., it computes $pk = D_{sk_\beta}(C_l)$.

If $\alpha_l = \beta$, then $pk = pk_{\alpha_{l+1}}$. Thus, if M_β manages signers, it learns the identity of the signer S_α, and if it manages other group managers it learns the identity of the group manager below it in the tree which, perhaps indirectly, manages the signer S_α.

Now suppose that $\alpha_l \neq \beta$, so $pk \neq pk_{\alpha_{l+1}}$. What does M_β, or indeed any outsider, learn about the identity of the signer S_α? It clearly does not learn anything from a cryptotext C_l about the encrypted cleartext, as long as the cryptosystem is semantically secure. There may be another way to deduce the content of C_l though. If the cryptotext C_{l+1} somehow indicate which public key was used to form it, M_β, or any outsider, can simply look at C_{l+1} and recover the cleartext of C_l. This means that it can look at the chain of cryptotexts and extract information on the identity of the signer. We conclude that using the approach above, we need a cryptosystem which not only hides the cleartext, but also hides the public key used to form the cryptotext. A cryptosystem with this property is said to be *anonymous*. We give a definition in Section 2.5.2. The property of anonymity was discussed in [1] and studied extensively by Bellare et al. in [14].

Next we consider the problem of ensuring hierarchical traceability. This problem consists of two parts. We must ensure chosen message security to avoid that an illegitimate signer is able compute a valid signature at all. The difficult problem is to ensure that the signer S_α not only formed $(C_0, \ldots, C_{\delta-1})$ as described above for some public keys $pk_{\alpha_0}, \ldots, pk_{\alpha_\delta}$, but also that the public keys used correspond to the unique path $\alpha_0, \alpha_1, \ldots, \alpha_\delta$ from the root $\rho = \alpha_0$ to the leaf $\alpha = \alpha_\delta$ corresponding to the signer S_α. This is the main obstacle to construct an efficient hierarchical group signature scheme.

13.6 A Characterization of Anonymous Cryptosystems

As explained above we need an anonymous cryptosystem to construct a hierarchical group signature scheme using our approach. The lemma below characterizes the set of cryptosystems which are both polynomially indistinguishable and anonymous.

Denote by $\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$ Experiment 2.11, but with the challenge cryptotext $\mathsf{Enc}_{pk_b}(m)$ replaced by an element distributed according to a distribution D_κ, where $\mathcal{D}_{\mathrm{ind}} = \{D_\kappa\}$, and correspondingly for $\mathsf{Exp}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$. We use $T_\mathcal{D}$ to denote the Turing machine that on input 1^κ returns a sample distributed according to an efficiently sampleable distribution D_κ. In other words we consider the following somewhat artificial experiments.

Experiment 13.7 ($\mathcal{D}_{\mathrm{ind}}$-Indistinguishability, $\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$).

$$\begin{aligned} (pk, sk) &\leftarrow \mathsf{CSKg}(1^\kappa) \\ (m_0, m_1, \mathrm{state}) &\leftarrow A(pk) \\ d &\leftarrow A(T_\mathcal{D}(1^\kappa), \mathrm{state}) \end{aligned}$$

The experiment outputs d.

Experiment 13.8 ($\mathcal{D}_{\mathrm{ind}}$-Anonymity, $\mathsf{Exp}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$).

$$\begin{aligned}(pk_0, sk_0) &\leftarrow \mathsf{CSKg}(1^\kappa)\\ (pk_1, sk_1) &\leftarrow \mathsf{CSKg}(1^\kappa)\\ (m, \mathrm{state}) &\leftarrow A(pk_0, pk_1)\\ d &\leftarrow A(T_{\mathcal{D}}(1^\kappa), \mathrm{state})\end{aligned}$$

The experiment outputs d.

Lemma 13.9. *Let $\mathcal{CS}$ be a cryptosystem which is both polynomially indistinguishable and anonymous. Then there exists an efficiently sampleable distribution $\mathcal{D}_{\mathrm{ind}}$ such that for all $A \in \mathrm{PPT}^*$ the absolute value*

$$|\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}(\kappa)}_{\mathcal{CS},A}(\kappa) = 1]|$$

is negligible for $b \in \{0,1\}$. The reverse implication holds as well.

The intuition behind this lemma is that if a cryptosystem that is both polynomially indistinguishable and anonymous, then the cryptotexts are polynomially indistinguishable from a random distribution which is independent of both the key and the plaintext.

Proof. Suppose that a distribution $\mathcal{D}_{\mathrm{ind}}$ as in the lemma exists. The indistinguishability of $\mathcal{CS}$ then follows by the triangle inequality. Suppose that $\mathcal{CS}$ is not anonymous. Then there exists an adversary $A \in \mathrm{PPT}^*$ such that

$$|\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A}(\kappa) = 1]|$$

is non-negligible which by the triangle inequality implies that

$$|\Pr[\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa) = 1]|$$

is non-negligible for a fixed $b \in \{0,1\}$, which we without loss assume to be 0. Let A' be the adversary in Experiment 2.11 defined as follows. On input pk it sets $pk_0 = pk$ generates $(pk_1, sk_1) = \mathsf{CSKg}(1^\kappa)$ and hands (pk_0, pk_1) to A, which returns (m, state). Then A' returns (m, m, state). When handed (c, state) from the experiment it returns the output of $A(c, \mathrm{state})$. By construction $\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A'}(\kappa)$ is identically distributed to $\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa)$, and $\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A'}(\kappa)$ is identically distributed to $\mathsf{Exp}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$. This is a contradiction, since it implies that

$$|\Pr[\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A'}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A'}(\kappa) = 1]|$$

is non-negligible.

Suppose next that $\mathcal{CS}$ is polynomially indistinguishable and anonymous. We define our candidate distribution $\mathcal{D}_{\text{ind}}$ as follows. To generate a sample from $\mathcal{D}_{\text{ind}}$, generate a key pair $(pk', sk') = \mathsf{CSKg}(1^\kappa)$ and output an encryption $\mathsf{Enc}_{pk'}(m')$, where m' is any fixed message. This implies that $\mathcal{D}_{\text{ind}}$ is efficiently sampleable. Assume that

$$|\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\text{ind}}}_{\mathcal{CS},A}(\kappa) = 1]|$$

is non-negligible for $b = 0$. Then it is also non-negligible for $b = 1$, since $\mathcal{CS}$ is polynomially indistinguishable. Let A'_0 be the adversary in Experiment 2.14 that does the following. On input (pk_0, pk_1) it hands pk_0 to A which returns (m_0, m_1). Then A'_0 returns m_0, and is given $\mathsf{Enc}_{pk_b}(m_0)$ for a randomly chosen $b \in \{0, 1\}$ by the experiment. It hands $\mathsf{Enc}_{pk_b}(m_0)$ to A and returns the output of A. A'_1 is identical to A'_0 except that it hands m' to the experiment instead of m_0. From the construction follows that $\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa)$ and $\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\text{ind}}}_{\mathcal{CS},A}(\kappa)$ are identically distributed to $\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa)$ and $\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_1}(\kappa)$ respectively. Thus

$$|\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_1}(\kappa) = 1]|$$

is non-negligible. From the anonymity of $\mathcal{CS}$ we have that

$$|\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_b}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_b}(\kappa) = 1]|$$

is negligible for $b \in \{0, 1\}$. A hybrid argument then implies that

$$|\Pr[\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A'_0}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A'_1}(\kappa) = 1]|$$

is non-negligible for some $b \in \{0, 1\}$. Without loss we assume $b = 0$. Denote by A'' the adversary in Experiment 2.11 defined as follows. Given input pk it hands pk to A. When A returns (m_0, m_1), it outputs (m_0, m'), and receives either $\mathsf{Enc}_{pk}(m_0)$ or $\mathsf{Enc}_{pk}(m')$, which it forwards to A. Finally, it returns the output of A. Since, $\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A''}(\kappa)$ is identically distributed to $\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa)$ and $\mathsf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS},A''}(\kappa)$ is identically distributed to $\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_1}(\kappa)$, this contradicts the indistinguishability of $\mathcal{CS}$. □

Note that $\mathcal{D}_{\text{ind}}$ depends on $\mathcal{CS}$ but is independent of all stochastic variables in the experiment. In the next chapter we prove that the probabilistic cryptosystem of Goldwasser and Micali [76] is anonymous.

Remark 13.10. Several standard probabilistic cryptosystems can be made anonymous by minor modifications, e.g., it is not hard to see that the El Gamal [71] cryptosystem is anonymous under the DDH-assumption if all parties employ the same group.

Chapter 14

A Construction Under General Assumptions

In this chapter we show how hierarchical group signatures can be constructed under general assumptions. Our focus is on feasibility and conceptual simplicity. More precisely, we prove the following theorem.

Theorem 14.1. *If there exists a family of trapdoor permutations, then there exists a secure hierarchical group signature scheme.*

To prove the theorem we construct a hierarchical group signature scheme and prove its security. This chapter is based on the paper by Trolin and Wikström [141].

14.1 Preliminaries

Our construction is based on three primitives: the group signature scheme of Bellare et al. [17], the public key cryptosystem of Goldwasser and Micali [76], and a non-interactive zero-knowledge proof as defined in Section 2.9. Of these we use the first and last in a blackbox way. Bellare et al. [17] prove the following theorem.

Theorem 14.2. *If there exists a family of trapdoor permutations, then there exists a secure group signature scheme* $\mathcal{GS} = (\mathsf{GKg}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open})$.

As explained in Section 13.3 every group signature scheme is also a hierarchical group signature scheme, and our definition of security reduces to the definition of security given in [17] for group signatures. The definition of a trapdoor permutation family is given in Section 2.3.

Recall from Section 2.9 that a non-interactive zero-knowledge proof (NIZK) allows a prover to send a single message to a verifier that convinces the verifier of some statement in NP. Bellare et al. [17] use a NIZK in their proof of the theorem above, but the NIZK we use must be adaptive zero-knowledge for polynomially

many statements, and not only for a single statement. The requirement on simulation soundness is in fact unchanged compared with [17], i.e., single statement simulation soundness suffices. A precise definition of the type of NIZK we use is given in Section 2.9.

De Santis et al. [135] extend the results in [62] and [133] and prove the following theorem.

Theorem 14.3. *If there exists a family of trapdoor permutations, then there exists a NIZK for any language in* NP*.*

The probabilistic cryptosystem of Goldwasser and Micali [76] is defined in Section 3.1. Goldwasser and Micali prove that their cryptosystem is polynomially indistinguishable, i.e., it satisfies Definition 2.12, but as explained in the previous chapter we need an anonymous cryptosystem. We prove the following lemma using Lemma 13.9 from the previous chapter.

Lemma 14.4. *If* $\mathcal{TPF} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ *is a trapdoor permutation family with hard-core bit* $\mathcal{B}$*, then* $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ *is anonymous.*

Proof. Suppose that $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ is not anonymous. Let $U_{\kappa+1}$ be the uniform and independent distribution over $\{0,1\}^{\kappa+1}$. Then for some adversary $A \in \mathrm{PPT}^*$,

$$|\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-U_{\kappa+1}}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A}(\kappa) = 1]|$$

is non-negligible for a fixed $b \in \{0,1\}$. Without loss we assume $b = 0$. Since $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ is a bitwise cryptosystem, we may without loss assume that $m_0 = 0$ and $m_1 = 1$. Let $m \in \{0,1\}$ be randomly chosen. Then a cryptotext $E_{pk}(m) = (\mathsf{Eval}(pk, r), \mathcal{B}(r) \oplus m)$ is distributed according to $U_{\kappa+1}$, since the function f_{pk} evaluated by Eval is a permutation and $\mathcal{B}(r) \oplus m$ is uniformly and independently distributed. A trivial averaging argument then implies that $|\Pr[\mathsf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A}(\kappa) = 1]|$ is non-negligible which is a contradiction. □

We remind the reader at this point that we define a trapdoor permutation family to have domain $\{0,1\}^{\kappa}$.

14.2 Our Construction

In this section we construct a hierarchical group signature scheme which we denote by $\mathcal{HGS} = (\mathsf{HKg}, \mathsf{HSig}, \mathsf{HVf}, \mathsf{HOpen})$. We let $\mathcal{TPF} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ denote a family of trapdoor permutations with a hard-core bit $\mathcal{B}$, and assume that a Goldwasser-Micali cryptosystem $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ has been constructed from this. We denote by $\mathcal{GS} = (\mathsf{GKg}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open})$ the group signature scheme of Bellare et al. also constructed from $\mathcal{TPF}$. We view this as a hierarchical group signature scheme of depth 1, but we denote its public key map and private key map by *gpk* and *gsk*

respectively instead of hpk and hsk to distinguish them from the public key map and private key maps of the hierarchical group signature scheme we are constructing. We also use $\mathcal{TPF}$ to construct a NIZK for a language L_{HGS} defined below.

The key generator is constructed as follows. First keys for the group signature scheme $\mathcal{GS}$ are generated, where the signers correspond to the signers in the hierarchical group signature scheme we are constructing, but the root group manager is not given its usual private opening key $gsk(\rho)$. Instead, each group manager is given a key pair (pk_β, sk_β) of the $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ cryptosystem. When a signer S_α signs a message m it first forms a group signature σ of the message m. Suppose that the signer corresponds to the path $\alpha_0, \ldots, \alpha_\delta$ in the tree, i.e., $\alpha_0 = \rho$ and $\alpha_\delta = \alpha$. Then the signer forms a chain of cryptotexts $C = (E_{pk_{\alpha_0}}(pk_{\alpha_1}), \ldots, E_{pk_{\alpha_{\delta-1}}}(pk_{\alpha_\delta}))$. Finally, it forms a NIZK π that the chain of cryptotexts C is formed in this way, and that the encrypted path corresponds to the identity of the signer hidden in the group signature σ. The hierarchical group signature consists of the tuple (σ, C, C', π). Verification of a signature corresponds to verifying the NIZK. Opening a signature using the private opening key of a group manager at depth l corresponds to decrypting the lth cryptotext. In the above description we have not mentioned how it is ensured that only one group manager on each level opens a signature to something other than $\perp$. This is done using an additional chain $C' = (E_{pk}(pk_{\alpha_0}), \ldots, E_{pk}(pk_{\alpha_{\delta-1}}))$.

Algorithm 14.5 (Key Generation, $\mathsf{HKg}(1^\kappa, T)$). The key generation algorithm is defined as follows.

1. Generate a random string $\xi \in \{0,1\}^*$ sufficiently long for a NIZK based on $\mathcal{TPF}$ for the language L_{HGS} defined below. Generate $(pk, sk) = \mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$.

2. For each node α in $\mathcal{V}(T)$, compute $(pk_\alpha, sk_\alpha) = \mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$.

3. Let I be the bijection mapping each list $(pk_{\alpha_0}, \ldots, pk_{\alpha_\delta})$ such that $\alpha_0, \ldots, \alpha_\delta$ is a path in T to α_δ, where $\alpha_0 = \rho$ and $\alpha_\delta \in \mathcal{L}(T)$. Define I to map anything else to $\perp$. Denote by $T_{\mathcal{GS}}$ the tree with root ρ and leaves $\mathcal{L}(T)$.

4. Run $(gpk, gsk) = \mathsf{GKg}(1^\kappa, T_{\mathcal{GS}})$ to generate keys for a group signature scheme, and set $(hpk(\alpha), hsk(\alpha)) = ((pk_\alpha, gpk(\alpha)), gsk(\alpha))$ for $\alpha \in \mathcal{L}(T)$.

5. Define the keys of the root ρ by $(hpk(\rho), hsk(\rho)) = ((\xi, pk, pk_\rho, gpk(\rho)), sk_\rho)$ and set $(hpk(\beta), hsk(\beta)) = (pk_\beta, sk_\beta)$ for $\beta \in \mathcal{V}(T) \backslash \mathcal{L}(T)$, $\beta \neq \rho$. Note that $hsk(\rho)$ does not contain $gsk(\rho)$.

6. Output (hpk, hsk).

The result of running the above algorithm is illustrated in Figure 14.1. We are now ready to define the NIZK we need in our construction.

We denote by $\pi_{\mathrm{hgs}} = (P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$ a NIZK of the language L_{HGS} consisting of tuples $(T, hpk, m, \sigma, C, C')$ such that there exists public keys $pk_0, \ldots, pk_\delta, gsk(\alpha)$

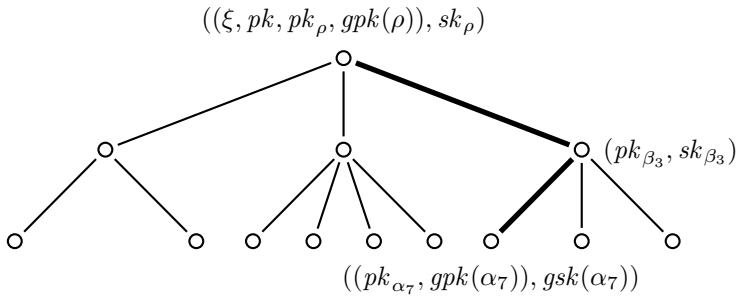

Figure 14.1: The figure illustrates the public and private keys along a path in the tree of keys corresponding to Figure 1.3. The edges along the path have thick edges. Each node contains a pair of public and private keys.

and random strings $r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta$ such that

$$\begin{aligned} \alpha_0 &= \rho \ , \\ (C_l, C'_l) &= (E_{pk_{\alpha_l}}((pk_{\alpha_{l+1}}, r'_l), r_l), E_{pk}(pk_{\alpha_l}, r'_l)) \ \text{ for } l = 0, \ldots, \delta - 1 \ , \\ \alpha &= I(pk_{\alpha_0}, \ldots, pk_{\alpha_{\delta-1}}) \ , \quad \text{and } \ \sigma = \mathsf{GSig}_{r_\delta}(m, T_{\mathcal{GS}}, gpk, gsk(\alpha)) \ . \end{aligned}$$

The NIZK now enables us to give a succinct description of the signature algorithm.

Algorithm 14.6 (Signing, $\mathsf{HSig}(m, T, hpk, hsk(\alpha))$). Let $\alpha_0, \ldots, \alpha_\delta$ be the path to the signer S_α, i.e., $\rho = \alpha_0$ and $\alpha_\delta = \alpha$. Choose r_i and r'_i randomly and compute

$$\begin{aligned} \sigma &= \mathsf{GSig}_{r_\delta}(m, T_{\mathcal{GS}}, gpk, gsk(\alpha)) \\ (C_l, C'_l) &= (E_{pk_{\alpha_l}}((pk_{\alpha_{l+1}}, r'_l), r_l), E_{pk}(pk_{\alpha_l}, r'_l)) \ \text{ for } l = 0, \ldots, \delta - 1 \ , \\ \pi &= P_{\mathrm{hgs}}((T, hpk, m, \sigma, C, C'), \\ &\qquad (pk_{\alpha_0}, \ldots, pk_{\alpha_\delta}, gsk(\alpha), r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta), \xi) \ . \end{aligned}$$

Then output (σ, C, C', π).

Algorithm 14.7 (Verification, $\mathsf{HVf}(T, hpk, m, (\sigma, C, C', \pi))$). On input a candidate signature (σ, C, C', π) output $V_{\mathrm{hgs}}((T, hpk, m, \sigma, C, C'), \pi, \xi)$.

Algorithm 14.8 (Opening, $\mathsf{HOpen}(T, gpk, gsk(\beta), m, (\sigma, C, C', \pi))$). Suppose the index β is on level l in T. If $\mathsf{HVf}(T, hpk, m, (\sigma, C, C', \pi)) = 0$, then return $\perp$. Otherwise, compute $(pk_\alpha, r') = D_{sk_\beta}(C_l)$ and verify that $C'_l = E_{pk}(pk_\beta, r')$ and $\alpha \in \beta$. Return α if this is the case and return $\perp$ otherwise.

Remark 14.9. The scheme described above differs from the scheme in [140]. As explained in Remark 13.4 the original definition in [140] did not capture all the

properties we expect from a hierarchical group signature scheme. We use a stronger definition in this thesis. The role of the new cryptotexts C_l' is to allow a group manager to ensure that no other group manager can open a signature to something other than $\perp$.

Remark 14.10. In Section 14.4 we describe an alternative construction which seems better suited if we try to eliminate the trusted key generator, but which is harder to analyze.

Remark 14.11. Suppose we want to instantiate the scheme using the RSA-function. Then an alternative to using the restrictive definition of a trapdoor permutation family and applying Yao's construction [16] to turn the RSA-function into a trapdoor permutation family with domain $\{0,1\}^\kappa$, is to modify the Goldwasser-Micali encryption algorithm as follows. It repeatedly chooses r until $\mathsf{Eval}(i,r) < 2^\kappa$. This implies that $\mathsf{Eval}(i,r) < N$ for all κ-bit moduli N and that the first part of a Goldwasser-Micali cryptotext is a uniformly distributed element in $\{0,1\}^\kappa$. The probability that r has this property is at least $1/4$. Given that we put a polynomial bound on the number of tried r, the encryption process fails with negligible probability. It is easy to see that the proof of anonymity goes through, and the polynomial indistinguishability of the scheme follows from the polynomial indistinguishability of the original, since the original scheme uses an r with $\mathsf{Eval}(i,r) < 2^\kappa$ with probability at least $1/4$. To change the construction in this way we do need to modify the definition of a public key cryptosystem such that it allows the encryption algorithm to fail with negligible probability.

14.3 Security Analysis

We prove the following lemma on the security of our construction, from which Theorem 14.1 follows immediately.

Lemma 14.12. *If $\mathcal{TPF}$ is a family of trapdoor permutations, then $\mathcal{HGS}$ is secure.*

Proof. We prove the hierarchical anonymity and the hierarchical traceability of $\mathcal{HGS}$ separately.

Proof of Hierarchical Anonymity. Suppose to the contrary that an adversary A breaks hierarchical anonymity. Then we have $\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa,T) \geq 1/\kappa^c$ for some polynomial size tree T, constant $c > 0$ and κ in an infinite index set $\mathcal{N}$. We construct a machine A' that runs A as a blackbox and breaks the hierarchical anonymity of $\mathcal{GS}$ (recall that hierarchical anonymity is a strict generalization of full anonymity).

Definition of A'. The adversary A' simulates the hierarchical anonymity experiment, Experiment 13.2, with $\mathcal{HGS}$ to A. It also plays the role of adversary in Experiment 13.2 with $\mathcal{GS}$.

The key generation is simulated as follows. First the NIZK simulator S_{hgs} is invoked to compute the reference string with trapdoor $(\xi, \text{simstate})$. The string ξ

is used instead of a random string. Recall that $T_{\mathcal{GS}}$ denotes the very simple tree having ρ, the root of T, as root, and children $\mathcal{L}(T)$. The adversary A' waits until it receives gpk and $(gsk(\alpha))_{\alpha\in\mathcal{L}(T)}$. Then it simulates the remaining part of the key generation honestly except that it uses the received values, and it does not define $gsk(\rho)$ at all. Thus, the keys of all intermediate group managers are generated by A'. In each iteration in the simulated experiment A may request $gsk(\alpha)$ for some group manager M_α and the simulator can answer this request honestly.

Queries to the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle are simulated in the following way. Given a query on the form $(\beta, m, (\sigma, C, C', \pi))$, A' first checks that $\beta \in \mathcal{V}(T)$ and

$$\mathsf{HVf}(T, hpk, m, (\sigma, C, C', \pi)) = 1 \ .$$

If not it returns $\perp$. If so it asks its $\mathsf{Open}(T_{\mathcal{GS}}, gpk, gsk(\cdot), \cdot, \cdot)$-oracle the question (ρ, m, σ), to which it replies by α. If $\alpha \notin \mathcal{L}(T)$ it returns $\perp$. Otherwise, let $\alpha_0, \ldots, \alpha_\delta$ be its corresponding path, i.e., $\alpha_0 = \rho$ and $\alpha_\delta = \alpha$. Let β be on depth l. Then A' instructs the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle to return α_{l+1} if $\beta = \alpha_l$ and $\perp$ otherwise. We expect that the answers computed in this way are correct, but this remains to be proved.

When A outputs $(\alpha^{(0)}, \alpha^{(1)}, m)$, A' outputs this as well. Let $\alpha_t^{(0)} = \alpha_t^{(1)}$ be the least common ancestor of $\alpha^{(0)}$ and $\alpha^{(1)}$, and let $\alpha^{(0)}, \alpha_{\delta-1}^{(0)}, \ldots, \alpha_t^{(0)}$ and $\alpha^{(1)}, \alpha_{\delta-1}^{(1)}, \ldots, \alpha_t^{(1)}$ be the paths to this index.

When A' is given a signature σ from its experiment it does the following. It computes the cryptotexts $C_0, C_0', \ldots, C_{t-1}, C_{t-1}'$ honestly. It chooses random samples $C_t, C_t', \ldots, C_{\delta-1}, C_{\delta-1}'$ according to the distribution $\mathcal{D}_{\text{ind}}$ guaranteed to exist by Lemma 13.9. Here we in fact need to apply Lemma 2.13 and Lemma 2.16 to increase the length of messages that can be encrypted, and then apply Lemma 13.9, but we abuse notation below. Then it invokes S_{hgs} of the NIZK on $((T, hpk, m, \sigma, C, C'), \xi, \text{simstate})$ to form a proof π, and hands (σ, C, C', π) to A.

Eventually A outputs a bit d, which A' then returns as output.

Analysis of A'. We divide our analysis into three claims. Denote by $H^b_{\text{c,o,p}}$ the machine that on input κ simply simulates Experiment 13.2 with $\mathcal{HGS}$ to A and outputs the result. Denote by $H^b_{\text{c,o}}$ the machine which is identical to $H^b_{\text{c,o,p}}$ except that it generates ξ as A' and also simulates the NIZK π exactly as A' does. Thus, except from the fact that the proof π in the challenge signature is simulated, $H^b_{\text{c,o}}$ simulates Experiment 13.2 with $\mathcal{HGS}$ to A. We also define H^b_{c} to be identical to $H^b_{\text{c,o}}$ except that it simulates the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle to A precisely as A' does. Finally, we define H^b to be identical to H^b_{c} except that the $C_t, C_t', \ldots, C_{\delta-1}, C_{\delta-1}'$ in the challenge signature are generated precisely as A' does.

Thus, by construction H^b is identically distributed to $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{GS},A'}(\kappa)$. This gives us a chain of distributions $H^b_{\text{c,o,p}}, H^b_{\text{c,o}}, H^b_{\text{c}}, H^b$ starting with $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa)$ and ending with $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{GS},A'}(\kappa)$. In the following claims we show that the distance between each pair of distributions is negligible.

Claim 1. There exists a negligible function $\epsilon_1(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c,o,p}}(\kappa)=1]-\Pr[H^b_{\mathrm{c,o}}(\kappa)=1]|<\epsilon_1(\kappa)\ .$$

Proof. The proof is based on the adaptive zero-knowledge of the NIZK $\pi_{\mathrm{hgs}}=(P_{\mathrm{hgs}},V_{\mathrm{hgs}},S_{\mathrm{hgs}})$.

Consider the adversary A_{adzk} defined as follows. It waits for a string ξ from Experiment 2.42. Then it starts the simulation of $H_{\mathrm{c,o}}$ except that it uses ξ instead of choosing it randomly. Then it continues the simulation of $H_{\mathrm{c,o}}$ until it is about to compute the NIZK π. Instead of computing the NIZK, it requests a NIZK π from Experiment 2.42. More precisely, it hands (T,hpk,m,σ,C,C') and $(pk_0,\ldots,pk_\delta,gsk(\alpha),r_0,r'_0,\ldots,r_{\delta-1},r'_{\delta-1},r_\delta)$ to the experiment. Finally, it continues the simulation of $H_{\mathrm{c,o}}$ until it halts.

It follows that the random variables $H^b_{\mathrm{c,o,p}}(\kappa)$ and $H^b_{\mathrm{c,o}}(\kappa)$ are identically distributed to $\mathsf{Exp}^{\mathsf{adind}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)$ and $\mathsf{Exp}^{\mathsf{adind}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)$ respectively. The adaptive zero-knowledge property of the NIZK implies that there exists a negligible function $\epsilon_1(\kappa)$ such that

$$|\mathsf{Exp}^{\mathsf{adind}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)-\mathsf{Exp}^{\mathsf{adind}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)|<\epsilon_1(\kappa)\ ,$$

and the claim follows. □

Claim 2. There exists a negligible function $\epsilon_2(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c,o}}(\kappa)=1]-\Pr[H^b_{\mathrm{c}}(\kappa)=1]|<\epsilon_2(\kappa)\ .$$

Proof. The proof of this claim follows from the simulation soundness and the soundness of the NIZK, and we give the details below. First we note that any query $(\beta,m,(\sigma,C,C',\pi))$ to the $\mathsf{Open}(T_{\mathcal{GS}},gpk,gsk(\cdot),\cdot,\cdot)$-oracle such that $V_{\mathrm{hgs}}((T,hpk,m,\sigma,C,C',\xi'),\pi,\xi)=0$ is answered correctly.

Consider now a query $(\beta,m,(\sigma,C,C',\pi))$, where π is a valid proof, that is, $V_{\mathrm{hgs}}((T,hpk,m,\sigma,C,C'),\pi,\xi)=1$, and still $(T,hpk,m,\sigma,C,C')\notin L_{\mathrm{HGS}}$. We argue that such queries are asked with negligible probability.

We construct an adversary A_{sims} (or A_{s}) against simulation soundness (or soundness), i.e., Experiment 2.44 (or the soundness property of Definition 2.41), as follows. It accepts the random string ξ as input and simulates H^b_{c} (or $H^b_{\mathrm{c,o}}$). Whenever A asks a query $(\beta,m,(\sigma,C,C',\pi))$, A_{sims} (or A_{s}) interrupts the simulation of H^b_{c} (or $H^b_{\mathrm{c,o}}$) and checks whether the query is such that $(T,hpk,m,\sigma,C,C')\in L_{\mathrm{HGS}}$. This is easily done using the keys to the cryptosystems and the group signature scheme. If $(T,hpk,m,\sigma,C,C')\notin L_{\mathrm{HGS}}$, then A_{sims} (or A_{s}) outputs $((T,hpk,m,\sigma,C,C'),\pi)$. From the simulation soundness (or soundness) we conclude that such queries are asked with negligible probability.

Consider a query $(\beta,m,(\sigma,C,C',\pi))$ to the $\mathsf{HOpen}(T,hpk,hsk(\cdot),\cdot,\cdot)$-oracle. Define $\alpha_0=\rho$ and define α_l for $l=1,\ldots,\delta$ by

$$\alpha_l=\mathsf{HOpen}(T,hpk,hsk(\alpha_{l-1}),m,(\sigma,C,C',\pi))\ .$$

We may assume without loss $(T, hpk, m, \sigma, C, C') \in L_{\mathrm{HGS}}$, since this happens with overwhelming probability for queries that verifies correctly.

This means that a query is answered incorrectly only if β is on level l, $\beta \neq \alpha_l$ and still $\beta' = \mathsf{HOpen}(T, hpk, hsk(\beta), m, (\sigma, C, C', \pi))$ with $\beta' \in \beta$. This is one of the two places in the proof where we use the cryptotexts in the list C' in an essential way. Without them, it is not only possible in theory to compute a signature that opens "correctly" using two distinct secret keys. It is an easy exercise to see that using the Goldwasser-Micali cryptosystem it would be easy to compute such a signature. Thus, we must argue that the cryptotexts in the list C' prohibits the construction of such signatures.

By the definition of the open algorithm the above anomaly can only happen if $(pk_{\beta'}, r') = D_{sk_\beta}(C_l)$ and $E_{pk}(pk_\beta, r') = C'_l$. This is a contradiction and we conclude that the claim is true, since we know that $E_{sk}(C'_l) = pk_{\alpha_l}$. □

Claim 3. There exists a negligible function $\epsilon_3(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c}}(\kappa) = 1] - \Pr[H^b(\kappa) = 1]| < \epsilon_3(\kappa) \ .$$

Proof. This follows from the polynomial indistinguishability and the anonymity of the Goldwasser-Micali cryptosystem $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$ using Theorem 3.1, Lemma 14.4, and Lemma 13.9 by use of a standard hybrid argument. We give details below.

We define a sequence of hybrid machines $A_{\mathrm{ind},l}$ for $l = t, \ldots, \delta - 1$ as follows. $A_{\mathrm{ind},l}$ simulates H^b_{c} until it has computed $(C_t, C'_t, \ldots, C_{\delta-1}, C'_{\delta-1})$. Then it computes samples $(\bar{C}_t, \bar{C}'_t, \ldots, \bar{C}_l, \bar{C}'_l)$ distributed according to the $\mathcal{D}_{\mathrm{ind}}$ distribution guaranteed by Lemma 13.9. Finally, it replaces

$$(C_t, C'_t, \ldots, C_{\delta-1}, C'_{\delta-1})$$

in its simulation by

$$(\bar{C}_t, \bar{C}'_t, \ldots, \bar{C}_l, \bar{C}'_l, C_{l+1}, C'_{l+1}, \ldots, C_{\delta-1}, C'_{\delta-1})$$

and continues the simulation of H^b_{c}. By construction, $A_{\mathrm{ind},t-1}(\kappa)$ and $A_{\mathrm{ind},\delta-1}(\kappa)$ are identically distributed to $H^b_{\mathrm{c}}(\kappa)$ and $H^b(\kappa)$ respectively.

Suppose that the claim is false, i.e., there exists a constant c and an infinite index set $\mathcal{N}'$ such that

$$|\Pr[A_{\mathrm{ind},t-1} = 1] - \Pr[A_{\mathrm{ind},\delta-1} = 1]| \geq \kappa^{-c}$$

for $\kappa \in \mathcal{N}'$. A hybrid argument implies that there exists a fixed $t \leq l < \delta - 1$ such that

$$|\Pr[A_{\mathrm{ind},l-1} = 1] - \Pr[A_{\mathrm{ind},l} = 1]| \geq \kappa^{-c}/\delta \ .$$

Denote by $A^1_{\mathrm{ind},l-1}$ the machine that simulates $A_{\mathrm{ind},l-1}$ except that it also replaces C_l by a sample $\bar{C}_l$ distributed according to $\mathcal{D}_{\mathrm{ind}}$. If we write $A^0_{\mathrm{ind},l-1}$ instead of $A_{\mathrm{ind},l-1}$ and $A^2_{\mathrm{ind},l-1}$ instead of $A_{\mathrm{ind},l}$ the triangle inequality implies that

$$|\Pr[A^{j-1}_{\mathrm{ind},l-1} = 1] - \Pr[A^j_{\mathrm{ind},l-1} = 1]| \geq \frac{1}{2\delta\kappa^c} \ .$$

for a fixed $j = \{1, 2\}$.

We consider the case $j = 1$. The other case follows by similar arguments. Consider the adversary A_{ind} for the indistinguishability experiment of the previous chapter, Experiment 13.7, run with $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$. It chooses $\beta_\delta^{(b)}$ randomly from $\mathcal{L}(T)$. Let $\beta_0, \ldots, \beta_\delta$ be the corresponding path, i.e., $\rho = \beta_0$ and $\beta_\delta = \beta_\delta^{(b)}$. Then it simulates $A^{j-1}_{\text{ind},l-1}$ except that the keys $(pk_{\beta_l}, sk_{\beta_l})$ are not generated. Instead it uses the key it receives from the experiment. It continues the simulation and hands $(pk_{\beta_{l+1}}, pk_{\beta_{l+1}})$ to its experiment. The experiment returns an element $\bar{C}_l$ that is used instead of C_l.

If A requests the private key sk_{β_l}, the simulation can not be continued and A_{ind} outputs 0. Similarly, if A outputs $(\alpha^{(0)}, \alpha^{(1)})$, where $\alpha^{(b)} \neq \beta^{(b)}$, then A_{ind} outputs 0. Since $\beta^{(b)}$ is randomly chosen, we have $\Pr[\alpha^{(b)} = \beta^{(b)}] = 1/|\mathcal{L}(T)|$.

If neither of the two events above occur, A_{ind} continues the simulation. We have

$$\begin{aligned}
&|\Pr[\mathsf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A_{\text{ind}}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind}-\mathcal{D}_{\text{ind}}}_{\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}},A_{\text{ind}}}(\kappa) = 1]| \\
&= |\Pr[A^0_{\text{ind},l-1} = 1 \wedge \alpha^{(b)} = \beta^{(b)}] - \Pr[A_{\text{ind}^1,l-1} = 1 \wedge \alpha^{(b)} = \beta^{(b)}]| \\
&= (1/|\mathcal{L}(T)|)|\Pr[A^0_{\text{ind},l-1} = 1] - \Pr[A^1_{\text{ind},l-1} = 1]| \geq 1/(|\mathcal{L}(T)|\delta\kappa^c) \ .
\end{aligned}$$

The first equality follows by construction. The second equality follows by independence. In view of Theorem 3.1, Lemma 14.4, and Lemma 13.9 this contradicts either the indistinguishability or the anonymity of $\mathcal{CS}^{\mathsf{gm}}_{\mathcal{TPF},\mathcal{B}}$. Thus, the claim is true. □

Claim 4. The hierarchical anonymity of $\mathcal{GS}$ is broken.

Proof. From Claim 1, Claim 2, and Claim 3 follows that

$$|\Pr[H^b_{\text{c,o,p}}(\kappa) = 1] - \Pr[H^b(\kappa) = 1]| < \epsilon_1(\kappa) + \epsilon_2(\kappa) + \epsilon_3(\kappa) \ ,$$

which gives

$$\begin{aligned}
&|\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{GS},A'}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{GS},A'}(\kappa) = 1]| \\
&\geq |\Pr[\mathsf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},A}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},A}(\kappa) = 1]| \\
&\quad -2(\epsilon_1(\kappa) + \epsilon_2(\kappa) + \epsilon_3(\kappa)) \ .
\end{aligned}$$

The assumption about A implies that the hierarchical anonymity is broken. □

Proof of Hierarchical Traceability. Suppose to the contrary that A breaks the hierarchical traceability of $\mathcal{HGS}$. Then $\mathsf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) \geq 1/\kappa^c$ for some polynomial size tree T, constant $c > 0$ and κ in an infinite index set $\mathcal{N}$. We construct a machine A' that runs A as a blackbox and breaks the hierarchical traceability of $\mathcal{GS}$ and thus its full traceability.

Definition of A'. The adversary A' simulates the hierarchical traceability experiment, Experiment 13.3, with $\mathcal{HGS}$ to A. It also plays the role of the adversary in Experiment 13.3 with $\mathcal{GS}$.

The key generation is simulated as follows. First the NIZK simulator is invoked to compute a reference string with a trapdoor $(\xi, \text{simstate})$. The string ξ is used instead of a random string. Recall that $T_{\mathcal{GS}}$ denotes the tree having ρ, the root of T, as root, and children $\mathcal{L}(T)$. The adversary A' waits until it receives the keys $(gpk(\rho), gsk(\rho))$ from its experiment. Then it simulates the key generation honestly except that it uses the received values, and it does not define $gsk(\alpha)$ for any $\alpha \in \mathcal{L}(T)$ at all. Thus, the keys of all intermediate group managers are generated by A'.

In each iteration in the experiment simulated to A, it may request $hsk(\alpha)$ for some signer S_α. When this happens A' requests $gsk(\alpha)$ from its experiment, and hands $gsk(\alpha)$ to A.

When A queries its $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$-oracle on (m, α) the reply is computed as follows. First A' queries its $\mathsf{GSig}(\cdot, T_{\mathcal{GS}}, gpk, gsk(\cdot))$-oracle on (m, α). The answer is a $\mathcal{GS}$ signature σ. Then A' computes $C_0, C'_0, \ldots, C_{\delta-1}, C'_{\delta-1}$ as defined by HSig. Finally, it uses the NIZK simulator S_{hgs} on input $((T, hpk, m, \sigma, C, C'), \xi, \text{simstate})$ to get a simulated proof π, and hands (σ, C, C', π) to A.

At some point A outputs a message-signature pair $(m, (\sigma, C, C', \pi))$. Then A' outputs (m, σ). This concludes the definition of A'.

Analysis of A'. We divide our analysis into several claims. Denote by $H_{\pi,\text{p}}$ the machine that simulates Experiment 13.3 with $\mathcal{HGS}$ to A, and outputs 1 if the experiment outputs 1 and the output $(m, (\sigma, C, C', \pi))$ of A satisfies $(T, hpk, m, \sigma, C, C') \in L_{\text{HGS}}$. Consider such an execution and define $\alpha_0 = \rho$ and α_l for $l = 1, \ldots, \delta$ by

$$\alpha_l = \mathsf{HOpen}(T, hpk, hsk(\alpha_{l-1}), m, (\sigma, C, C', \pi)) \ .$$

Consider how the experiment can output 1. We argue that there does not exist a β on level l in T such that $\beta \neq \alpha_l$ and $\beta' = \mathsf{HOpen}(T, hpk, hsk(\beta), m, (\sigma, C, C', \pi))$ with $\beta' \in \beta$. If it did, then we would have $(pk_{\beta'}, r') = D_{sk_\beta}(C_l)$ and $E_{pk}(pk_\beta, r') = C'_l$, but this contradicts the fact that $E_{sk}(C'_l) = pk_{\alpha_l}$. Thus, the only explanation for the output 1 is that $\alpha_\delta \notin \mathcal{C}$.

Denote by H_π the machine that is identical to $H_{\pi,\text{p}}$ except that it simulates the answers from the $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$-oracle to A precisely as A' does.

Claim 5. There exists a negligible function $\epsilon_1(\kappa)$ such that

$$\Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1] \leq \Pr[H_{\pi,\text{p}}(\kappa) = 1] + \epsilon_1(\kappa) \ .$$

Proof. The claim follows from the soundness of the NIZK. Denote by $E_{\pi,p}$ the event that the output $(m, (\sigma, C, C', \pi))$ of A satisfies $(T, hpk, m, \sigma, C, C') \in L_{\text{HGS}}$. From the soundness of the NIZK follows that $\Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1 \wedge \overline{E_{\pi,\text{p}}}]$ is negligible. By definition we have that $\Pr[H_{\pi,\text{p}}(\kappa) = 1] = \Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1 \wedge E_{\pi,\text{p}}]$. The claim follows. □

Claim 6. There exists a negligible function $\epsilon_2(\kappa)$ such that

$$|\Pr[H_\pi(\kappa)=1]-\Pr[H_{\pi,\mathrm{p}}(\kappa)=1]| \quad < \quad \epsilon_2(\kappa) \ .$$

Proof. The claim follows from the adaptive zero-knowledge of the NIZK. We construct an adversary A_{adzk} against the adaptive zero-knowledge, Experiment 2.42, as follows.

It simulates H_π except for the following two modifications. Firstly, it uses the random string ξ from the experiment instead of generating its own. Secondly, instead of invoking the simulator S_{hgs} on input $((T, hpk, m, \sigma, C, C'), \xi, \mathrm{simstate})$ to produce a NIZK π it requests a NIZK of $(T, hpk, m, \sigma, C, C')$ from its experiment. To do this it must also hand a witness to the experiment, but this is not a problem since the remainder of the signature is generated honestly. It follows that

$$\begin{aligned}&|\Pr[H_{\pi,\mathrm{p}}(\kappa)=1]-\Pr[H_\pi(\kappa)=1]|\\ &=|\Pr[\mathsf{Exp}^{\mathsf{adzk}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)=1]-\Pr[\mathsf{Exp}^{\mathsf{adzk}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)=1]|<\epsilon_2(\kappa)\ ,\end{aligned}$$

for some negligible function $\epsilon_2(\kappa)$. □

Claim 7. $\Pr[H_\pi(\kappa)=1] \leq \Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{GS},A'}(\kappa)=1]$.

Proof. All inputs to A in the simulation of H_π are identically distributed to the corresponding inputs in Experiment 13.3. The only difference in how the output of H_π and the output in the traceability experiment are defined is that H_π outputs 1 if the output $(m,(\sigma,C,C',\pi))$ of A satisfies $(T, hpk, m, \sigma, C, C') \in L_{\mathrm{HGS}}$ and $\alpha \notin \mathcal{C}$, whereas the experiment outputs 1 if $\mathsf{GVf}(T_{\mathcal{GS}}, gpk, m, \sigma) = 1$ and $\alpha_\delta \notin \mathcal{C}$. The former implies the latter and the claim follows. □

Claim 8. The hierarchical traceability of $\mathcal{GS}$ is broken.

Proof. From Claim 5, Claim 6, and Claim 7 follows that

$$\begin{aligned}&\Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa)=1]\leq\Pr[H_{\pi,\mathrm{p}}(\kappa)=1]+\epsilon_1(\kappa)\\ &\leq Pr[H_\pi(\kappa)=1]+\epsilon_1(\kappa)+\epsilon_2(\kappa)\leq\Pr[\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{GS},A'}(\kappa)=1]+\epsilon_1(\kappa)+\epsilon_2(\kappa)\ .\end{aligned}$$

The claim now follows from the assumption that $\mathcal{HGS}$ is broken by A. □

Conclusion of Proof. If $\mathcal{HGS}$ is not hierarchically anonymous, then by Claim 4 neither is $\mathcal{GS}$. If $\mathcal{HGS}$ is not hierarchically traceable, then by Claim 8 neither is $\mathcal{GS}$. This concludes the proof. □

14.4 An Alternative Construction

The construction we describe above is not a good starting point if we want to find a hierarchical group signature scheme for the adaptive setting. In this section we sketch an alternative construction. Let $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme. For each group manager M_α and signer S_α, $(spk_\alpha, ssk_\alpha) = \mathsf{Kg}(1^\kappa)$, and keys $(pk_\alpha, sk_\alpha) = \mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$ to a Goldwasser-Micali cryptosystem are generated. Then for each child α of $\beta \in \mathcal{V}(T)$, $\sigma_\beta(\alpha) = \mathsf{Sig}_{ssk_\beta}(spk_\alpha, pk_\alpha)$ is computed. For each $\alpha \in \mathcal{V}(T) \backslash \mathcal{L}(T) \backslash \{\rho\}$ we set $hpk(\alpha) = (spk_\alpha, pk_\alpha, \sigma_\beta(\alpha))$, where $\alpha \in \beta$, and $hsk(\alpha) = sk_\alpha$. For each $\alpha \in \mathcal{L}(T)$ set $hpk(\alpha) = (spk_\alpha, pk_\alpha, \sigma_\beta(\alpha))$, where $\alpha \in \beta$, and $hsk(\alpha) = (ssk_\alpha, sk_\alpha)$. For the root ρ we set $hpk(\rho) = (spk_\rho, pk_\rho)$ and $hsk(\rho) = sk_\rho$.

Consider a signer S_α corresponding to a path $\alpha_0, \ldots, \alpha_\delta$, where $\alpha_0 = \rho$ and $\alpha_\delta = \alpha$. To sign a message m the signer computes

$$\begin{aligned} C_l &= E_{pk_{\alpha_l}}((\sigma_{\alpha_l}(\alpha_{l+1}), r'_l), r_l) \ \text{, for } l = 0, \ldots, \delta - 1 \ , \\ C'_l &= E_{pk}(pk_{\alpha_l}, r'_l) \ \text{, for } l = 0, \ldots, \delta - 1 \ , \textit{and} \\ C_\delta &= E_{pk_{\alpha_\delta}}(\mathsf{Sig}_{ssk_\alpha}(m))) \end{aligned}$$

and provides a NIZK π that (C, C') is formed as above with $pk_{\alpha_0} = pk_\rho$. The signature consists of the tuple (C, C', π).

To verify a signature (C, C', π) the verifier simply checks the NIZK π. To open a signature, a group manager M_β on depth l first verifies the signature. If it is not valid, it returns $\perp$. Otherwise it computes $(\sigma, r') = D_{sk_\beta}(C_l)$. If σ is equal to $\sigma_\beta(\alpha)$ for some $\alpha \in \beta$ and $E_{pk}(pk_\beta, 1) = C'_l$, then it returns α and otherwise $\perp$.

This construction is a strict generalization of the construction in [17] except that we require that the cryptosystem used is anonymous. We believe that the construction is provably secure under the existence of a family of trapdoor permutations. However, as part of the proof we must essentially redo the analysis of the CCA2-secure cryptosystem of Sahai [133], and the group signature scheme of Bellare et al. [17], which makes the proof more complex than the proof for the construction we detail in this thesis.

A potential advantage of this scheme is that key generation need not be performed centrally. Each group manager M_β could also be given the private signature key ssk_β which allows it to generate (spk_α, pk_α) and (ssk_α, sk_α) for a child M_α or S_α by itself. Thus, a group manager could issue keys without interacting with any other group manager. As we will see in the next section, it is far from obvious how to define the security of such a scheme.

14.5 On Eliminating the Trusted Key Generator

We have defined hierarchical group signatures using a trusted key generator. This is a natural first step when trying to understand a new notion, but there are situations where one would like a hierarchical group signature scheme without a trusted party.

If there exists a set of parties of which the majority is trusted, general multiparty techniques can be used to replace the trusted key generator by the secure evaluation of a function. Although this solves the problem in some sense it introduces a trust hierarchy that is inconsistent with the hierarchy of the group managers and signers.

Consider now the security of a construction without a trusted key generator. In this case hierarchical anonymity and hierarchical traceability do not suffice to ensure security. The problem is that the experiments only consider the advantage of an adversary when all keys are generated honestly. Thus, all bets are off if this is not the case. It is, however, not clear what a definition of security for hierarchical group signatures without a trusted key generator should look like.

The adversary should probably have the power to choose its keys adaptively, based on the keys and signatures of honest parties. There are many subtle issues. For example, without a trusted key generator the default for hierarchical group signatures is that not only trees but general acyclic graphs of group managers are allowed. Is this what we want? If only trees are supposed to be allowed, certificates must embed additional information that restricts how a certificate chain may look and the NIZK must consider this as well. Other interesting questions are: Is there a well defined tree? Do all parties know what the tree looks like? Who generates the common random string used by the NIZKs?

We believe that the alternative construction described above is well suited to a setting without a trusted key generator, but without a rigorous definition of security we cannot claim anything.

Chapter 15

A Construction Under Standard Assumptions

In this section we construct an almost practical hierarchical group signature scheme. We give an explicit construction where the details of all subprotocols are completely specified. Then we prove the security of our construction in the random oracle model under the discrete logarithm assumption and the strong RSA-assumption.

The primitives we use to achieve this result are the El Gamal cryptosystem, the Cramer-Shoup cryptosystem, the Cramer-Shoup signature scheme, the Chaum-van Heijst-Pfitzmann hash function, and the Shamir hash function.

This chapter is based on the paper by Trolin and Wikström [141].

15.1 An Informal Description of Our Construction

Our construction is quite complex, so before presenting any details we give an informal description of the key ideas. Recall from Section 3.6 the definition and basic properties of the El Gamal cryptosystem [71]. It is well known that the El Gamal cryptosystem is semantically secure under the DDH-assumption, but it is easy to see that it is also anonymous, as long as a fixed group is used for all parties. We exploit both properties in our construction. Each group manager M_β holds a private key x_β and a public key $y_\beta = g^{x_\beta}$ of an El Gamal cryptosystem.

Recall from Section 3.10 the definition of the Cramer-Shoup signature scheme [54]. It is provably secure under the strong RSA-assumption. We exploit this to form the private keys of the signers. The private key of a signer S_α is a Cramer-Shoup signature $\sigma_\alpha = \mathsf{Sig}^{\mathsf{cs}}(y_{\alpha_1}, \ldots, y_{\alpha_{\delta-1}})$ of the public keys corresponding to the path $\alpha_0, \alpha_1, \ldots, \alpha_\delta$ from the root $\rho = \alpha_0$ to the leaf $\alpha = \alpha_\delta$.

To form a signature of a message m the signer first computes a chain of crypto-

texts on the form

$$((u_0, v_0, u'_0, v'_0), \ldots, (u_{\delta-1}, v_{\delta-1}, u'_{\delta-1}, v'_{\delta-1})) \\ = (E_{y_{\alpha_0}}(y_{\alpha_1}), E_{y_{\alpha_0}}(1), \ldots, E_{y_{\alpha_{\delta-1}}}(y_{\alpha_\delta}), E_{y_{\alpha_{\delta-1}}}(1)) \ .$$

Then the signer computes a commitment $C(\sigma_\alpha)$ of the signature σ_α. Finally, it computes an honest verifier zero-knowledge proof of knowledge $\pi(m)$ that the cryptotexts above form a chain and that $C(\sigma_\alpha)$ hides a signature of the list $(y_{\alpha_1}, \ldots, y_{\alpha_{\delta-1}})$ of the public keys used to form the chain of cryptotexts. The proof is given in the random oracle model and the message m to be signed is given as a prefix to the query to the random oracle. Thus, the complete signature is given by

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C(\sigma_\alpha), \pi(m)) \ .$$

Recall from Section 2.10 that in a proof in the random oracle model one or several cryptographic hash functions are modeled by randomly chosen functions.

Intuitively, this means that if a signer S_α can produce a valid signature, we can by rewinding extract a signature of the list of public keys corresponding to the path from the root to the signer. Thus, a signature can only be formed if the signer is legitimate and if it has formed the chain correctly.

The pair (u'_l, v'_l) may seem useless, but it allows a group manager M_β on level l to determine if the cryptotext (u_l, v_l) in a signature is computed using its public key y_β or not, and thus avoid that the opening of a signature gives the wrong result.

15.1.1 An Informal Description of the Proof of Knowledge

The main obstacle to find an efficient hierarchical group signature scheme following our approach is how to prove efficiently that $C(\sigma_\alpha)$ is a commitment of a signature σ_α of the list of public keys $(y_{\alpha_1}, \ldots, y_{\alpha_{\delta-1}})$ used to form the chain $((u_0, v_0, u'_0, v'_0), \ldots, (u_{\delta-1}, v_{\delta-1}, u'_{\delta-1}, v'_{\delta-1}))$. We construct a reasonably practical honest verifier zero-knowledge public coin proof for this relation by carefully selecting and combining a variety of cryptographic primitives and techniques.

Let $q_0, \ldots, q_3$ be primes such that $q_i = 2q_{i+1} + 1$ for $i = 0, 1, 2$. Recall from Section 3.2 that such a list of primes is called a Cunningham chain and that it exists under mild assumptions on the distribution of primes. There is a subgroup $G_{q_{i+1}} \subset \mathbb{Z}_{q_i}^*$ of order q_{i+1} for $i = 0, 1, 2$. Denote by g_i and y_i fixed and independently chosen generators of G_{q_i} for $i = 1, 2, 3$, i.e., $\log_{g_i} y_i$ is not known to any party in the protocol. Thus, we can form a commitment of a value $y_\alpha \in G_{q_3}$ in three ways, as

$$(y_3^{t'''} g_3^{s'''}, y_3^{s'''} y_\alpha) \ , \quad (y_2^{t''} g_2^{s''}, y_2^{s''} g_2^{y_\alpha}) \ , \text{ and } \quad (y_1^{t'} g_1^{s'}, y_1^{s'} g_1^{g_2^{y_\alpha}}) \ ,$$

where $t''', s''' \in \mathbb{Z}_{q_3}$, $t'', s'' \in \mathbb{Z}_{q_2}$, and $t', s' \in \mathbb{Z}_{q_1}$ are randomly chosen. By extending the ideas of Stadler [138] we can give a reasonably practical cut-and-choose proof that the elements hidden in two such commitments are identical.

Recall from Section 3.4 that the collision-free Chaum-Heijst-Pfitzmann hash function [47] is defined by $H^{\mathsf{CHP}} : \mathbb{Z}_{q_2}^{\delta} \to G_{q_2}$, $H^{\mathsf{CHP}} : (z_1, \ldots, z_\delta) \mapsto \prod_{l=1}^{\delta} h_l^{z_l}$, where $h_1, \ldots, h_\delta \in G_{q_2}$ are randomly chosen, i.e., no party knows a non-trivial representation of $1 \in G_{q_2}$ in these elements.

We employ El Gamal over G_{q_3}. This means that the public keys $y_{\alpha_1}, \ldots, y_{\alpha_\delta}$ belong to G_{q_3}. Although it is not trivial, the reader should not find it too hard to imagine that Stadler-techniques can be used to prove that the public keys used for encryption are identical to values hidden in a list of commitments formed as

$$((\mu_0, \nu_0), \ldots, (\mu_{\delta-1}, \nu_{\delta-1})) = ((y_2^{t_0''} g_2^{s_0''}, y_2^{s_0''} h_1^{y_{\alpha_1}}), \ldots, (y_2^{t_{\delta-1}''} g_2^{s_{\delta-1}''}, y_2^{s_{\delta-1}''} h_\delta^{y_{\alpha_\delta}})) \ .$$

The importance of this is that if we take the product of the commitments we get a commitment of $H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$, i.e.,

$$\left(\prod_{i=0}^{\delta-1} \mu_i, \prod_{i=0}^{\delta-1} \nu_i \right) = \left(y_2^{t''} g_2^{s''}, y_2^{s''} \prod_{i=1}^{\delta} h_i^{y_{\alpha_i}} \right) \ , \tag{15.1}$$

for some $t'', s'' \in \mathbb{Z}_{q_2}$. Thus, at this point we have devised a way for the signer to verifiably commit to the hash value of the keys it used to form the chain of cryptotexts. This is a key step in the construction.

Recall that the signer commits to a Cramer-Shoup signature σ_α of the list of public keys it uses to form the chain of cryptotexts. The Cramer-Shoup signature scheme uses an RSA-modulus $\mathbf{N}$ and elements from the subgroup $\mathrm{QR}_{\mathbf{N}}$ of squares in $\mathbb{Z}_{\mathbf{N}}^*$, and it is parameterized by two collision-free hash functions. We refer the reader to Section 3.10 for details on this. The first hash function is used to compute a message digest of the message to be signed, i.e., the list $(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$ of public keys. Above we have sketched how the signer can verifiably form a commitment of the H^{CHP} hash value of this message, so it is only natural that we let this be the first of the two hash functions in the signature scheme. In the signature scheme the message digest lives in the exponent of an element in $\mathrm{QR}_{\mathbf{N}}$. To move the hash value up in the exponent and to change group from G_{q_1} to $\mathrm{QR}_{\mathbf{N}}$, the signer forms two commitments

$$\left(y_1^{t'} g_1^{s'}, y_1^{s'} g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})}\right) \text{ and } \quad \mathbf{y}^t \mathbf{g}^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})} \ .$$

Then it gives a cut-and-choose proof that the exponent in the left commitment equals the value committed to in the product (15.1). It also proves that the exponents in the two commitments are equal. Thus, at this point the signer has proved that it holds a commitment over $\mathrm{QR}_{\mathbf{N}}$ of the hash value of the public keys it used to form the chain of cryptotexts.

The second hash function used in the Cramer-Shoup signature scheme is applied to a single element in $\mathrm{QR}_{\mathbf{N}}$. Since H^{CHP} is not collision-free on such inputs, we use the Shamir hash function defined by $H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})} : \mathbb{Z} \to \mathrm{QR}_{\mathbf{N}}$, $x \mapsto \mathbf{g}^x \bmod \mathbf{N}$ instead. A more detailed account of this function is given in Section 3.9. Using similar

techniques as explained above the signer evaluates the hash function and moves the result into the exponent, by two Stadler-like cut-and-choose proofs.

Given the two hash values in the exponents of two commitments, standard techniques can be used to prove that the commitment $C(\sigma_\alpha)$ is a commitment of the Cramer-Shoup signature σ_α of the list of public keys used to form the chain of cryptotexts.

15.2 Our Construction

We are now ready to describe the details of our construction following the informal description above. We denote our scheme by $\mathcal{HGS} = (\mathsf{HKg}, \mathsf{HSig}, \mathsf{HVf}, \mathsf{HOpen})$, and define algorithms HKg, HSig, HVf, and HOpen below.

Denote by κ_c and κ_r two additional security parameters that are defined as functions of κ such that $2^{-\kappa_c}$ and $2^{-\kappa_r}$ are negligible.

15.2.1 Key Generation

The key generation phase proceeds as follows. Each group manager is given an El Gamal key pair, and each signer is given a Cramer-Shoup signature of the public keys of the group managers on the path from the root to the signer.

Algorithm 15.1 (Key Generation, $\mathsf{HKg}(1^\kappa, T)$).

1. Run $(q_0, \ldots, q_3) = \mathsf{CunnGen}_4(1^\kappa)$ to generate a Cunningham chain, and let $g_i, y_i \in G_{q_i}$ be random elements for $i = 1, 2, 3$.
2. Let δ be the depth of the tree T, and run
$$H^{\mathsf{CHP}} = (h_1, \ldots, h_\delta) = \mathsf{CHPg}(q_2, \delta)$$
to generate a collision-free Chaum-van Heijst-Pfitzmann hash function.
3. Run $(X, Y) = \mathsf{Kg}^{\mathsf{cs}}(q_3)$ to generate keys for a Cramer-Shoup cryptosystem over G_{q_3}.
4. Run $((H^{\mathsf{CHP}}, \mathbf{g}, \mathbf{N}, \mathbf{h}, \mathbf{z}, e'), (H^{\mathsf{CHP}}, \mathbf{g}, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}')) = \mathsf{SSKg}^{\mathsf{cs}}_{\mathsf{CHPg},\mathsf{Shg}}(1^\kappa)$ and choose $\mathbf{y} \in \mathrm{QR}_{\mathbf{N}}$ randomly to generate keys for a Cramer-Shoup signature scheme employed with the collision-free hash functions H^{CHP} and $H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}$ and for a commitment scheme. We sometimes write (spk, ssk) for the above keys to simplify notation.
5. Compute the integer $a < \kappa$ such that $P = a\mathbf{pq} + 1$ is prime. Recall from Section 3.10 that there exists such a prime. Choose $g_{\mathbf{N}}, y_{\mathbf{N}} \in G_{\mathbf{N}}$ randomly.
6. For each node $\beta \in \mathcal{V}(T)$, generate keys
$$(hpk(\beta), hsk(\beta)) = (y_\beta, x_\beta) = \mathsf{CSKg}^{\mathsf{elg}}(q_3, g_3)$$
for an El Gamal cryptosystem over G_{q_3}.

7. For each leaf $\alpha \in \mathcal{L}(T)$ let $\alpha_0, \ldots, \alpha_\delta$ be the path from the root to α, where $\alpha_0 = \rho$ and $\alpha_\delta = \alpha$, and compute

$$(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha) = \mathsf{Sig}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, ssk}(y_{\alpha_1}, \ldots, y_{\alpha_\delta}) \ .$$

   Then redefine $hsk(\alpha) = (e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$.

8. Let ρ be the root of T. Redefine the public key $hpk(\rho)$ of the root ρ to be

$$(hpk(\rho), \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{g}, \mathbf{y}, q_0, g_1, y_1, g_2, y_2, g_3, y_3, H^{\mathsf{CHP}}, Y, g_{\mathbf{N}}, y_{\mathbf{N}}, \kappa_c, \kappa_r)$$

   and output (hpk, hsk).

Remark 15.2. The security parameters κ_c and κ_r are used in the proof of knowledge and decide its completeness, soundness, and the statistical distance between a real and simulated view of the protocol.

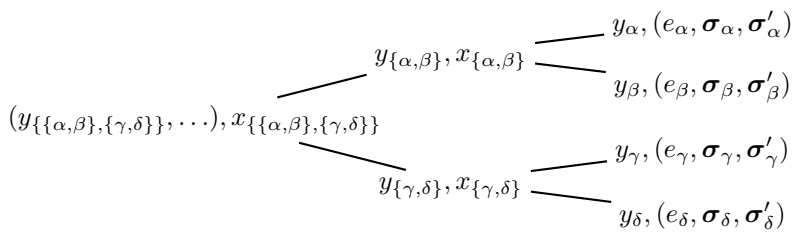

Figure 15.1: An illustration of the output of HKg for a three-level tree. The common group parameters, i.e., key size, generators etc., are not explicit.

15.2.2 Computing, Verifying, and Opening a Signature

In this section we give a detailed description of the signing algorithm, the verification algorithm, and the opening algorithm of the scheme. Denote by $L_{R_{\mathrm{HGS}}}$ the language consisting of tuples

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C})$$

in $G_{q_3}^{4\delta} \times G_{q_3}^4 \times \mathrm{QR}_{\mathbf{N}}^5$ such that there exists

$$((\tau_0, \tau'_0, \ldots, \tau_{\delta-1}, \tau'_{\delta-1}, \tau_\delta), (\boldsymbol{\tau}, \boldsymbol{\zeta}, \boldsymbol{\tau}', \boldsymbol{\zeta}', \boldsymbol{\psi}, \varepsilon))$$

in $\mathbb{Z}_{q_3}^{2\delta+1} \times [0, 2^{\kappa_r}\mathbf{N} - 1]^5 \times [2^{\kappa}, 2^{\kappa+1} - 1]$ such that

$$\begin{aligned}
&\gamma_0 = y_{\alpha_0} \ , \\
&(u_l, v_l, u'_l, v'_l) = (E_{(\gamma_l, g)}(\gamma_{l+1}, \tau_l), E_{(\gamma_l, g)}(1, \tau'_l)) \ \text{ for } l = 0, \ldots, \delta - 1 \ , \\
&C_\delta = E_Y^{\mathsf{cs}}(\gamma_\delta, \tau_\delta) \ , \\
&\mathbf{u} = \mathbf{y}^{\boldsymbol{\zeta}}\mathbf{g}^{\boldsymbol{\tau}} \ , \quad \mathbf{u}' = \mathbf{y}^{\boldsymbol{\zeta}'}\mathbf{g}^{\boldsymbol{\tau}'} \ , \quad \mathbf{C} = \mathbf{y}^{\boldsymbol{\psi}}\mathbf{g}^{\varepsilon} \ , \quad \text{and} \\
&\mathsf{Vf}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}, spk}((\gamma_1, \ldots, \gamma_\delta), (\varepsilon, \mathbf{v}/\mathbf{y}^{\boldsymbol{\tau}}, \mathbf{v}'/\mathbf{y}^{\boldsymbol{\tau}'})) = 1 \ .
\end{aligned}$$

In Chapter 16 we construct a zero-knowledge proof of knowledge denoted by $\pi_{\mathrm{hgs}} = (P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$ for this relation.

Algorithm 15.3 (Signing, $\mathsf{HSig}(m, T, hpk, hsk(\alpha))$). Let $\alpha_0, \ldots, \alpha_\delta$ with $\rho = \alpha_0$ and $\alpha_\delta = \alpha$ be the path to the signer S_α, and write $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha) = hsk(\alpha)$

1. Choose $r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta \in \mathbb{Z}_{q_3}$ randomly and compute $(u_l, v_l, u'_l, v'_l) = (E_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l), E_{(y_{\alpha_l}, g_3)}(1, r'_l))$, for $l = 0, \ldots, \delta-1$, and $C_\delta = E_Y^{\mathsf{cs}}(y_{\alpha_\delta}, r_\delta)$. This is the list of cryptotexts.

2. Choose $r, s, r', s', t \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ randomly and set $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^r, \mathbf{y}^r\boldsymbol{\sigma}_\alpha)$, $(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{s'}\mathbf{g}^{r'}, \mathbf{y}^{r'}\boldsymbol{\sigma}'_\alpha)$, and $\mathbf{C} = \mathbf{y}^t\mathbf{g}^{e_\alpha}$. This is a commitment of the signature $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$.

3. Compute a non-interactive proof

$$\begin{aligned}
\pi = P_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}\big(&((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}), \\
&((\tau_0, \ldots, \tau_\delta), (\boldsymbol{\tau}, \boldsymbol{\zeta}, \boldsymbol{\tau}', \boldsymbol{\zeta}', \boldsymbol{\psi}, \varepsilon))\big)
\end{aligned}$$

   in the random oracle model using the message m as a prefix.

4. Output the signature $\big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \pi\big)$.

Remark 15.4. Note that we switch the order of the components in the El Gamal cryptosystem in order to simplify the construction of the proof of knowledge. For example, $D_{1/x_\rho}(u_0, v_0) = y_{\alpha_1}$.

The construction of the proof of knowledge π_{hgs} is involved and postponed until Chapter 16. The verification algorithm consists simply of verifying the proof of knowledge contained in a signature.

Algorithm 15.5 (Verification, $\mathsf{HVf}(T, hpk, m, \sigma)$). On input a candidate signature $\sigma = (c, \pi) = \big(\big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}\big), \pi\big)$ return $V_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}(c, \pi)$.

To open a signature a group manager on depth l first verifies that the signature is valid and that its public key was used to form (u_l, v_l). Only then does it decrypt (u_l, v_l).

Algorithm 15.6 (Open, $\mathsf{HOpen}(T, hpk, hsk(\beta), m, \sigma)$). On input a candidate signature $\sigma = \big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \pi\big)$, if $\mathsf{HVf}(T, hpk, m, \sigma) = \perp$ or if $u'_l \neq (v'_l)^{x_\beta}$ and $\beta \neq \rho$, then return $\perp$. Otherwise compute $y_\alpha = D_{1/x_\beta}(u_l, v_l)$ and return α.

Remark 15.7. To ensure that the protocol satisfies the new and stronger definition instead of the one presented in [141], the elements (u'_l, v'_l) are added. The role played by these elements is to let a group manager verify, given a signature, that no other group manager can open the signature.

15.2.3 The Complexity of the Scheme

The main computational cost of the protocol lies in computing the zero-knowledge proof of knowledge π_{hgs}. Thus, we postpone the complexity analysis to Section 16.6.

15.3 Security Analysis

We analyze the security of the scheme and prove the following theorem.

Theorem 15.8. *The hierarchical signature scheme $\mathcal{HGS}$ is secure in the random oracle model under the DL-assumption, the DDH-assumption, and the strong RSA-assumption.*

Proof. The proof proceeds by contradiction. We show that an adversary that breaks the hierarchical group signature scheme breaks the DL-assumption, the DDH-assumption, or the strong RSA-assumption.

We can not use the Cramer-Shoup signature scheme as a blackbox and reach a contradiction to its security. The problem is that we use the RSA-modulus of the signature scheme also for commitments. Fortunately, Cramer and Shoup [54] describe a simulator running an adversary A as a blackbox. The simulator simulates the CMA-experiment to the adversary in a way that is statistically indistinguishable from the real experiment. Furthermore, if in the simulation the adversary with non-negligible probability can output a signature of a message on which it never queried the simulated signature oracle, then the strong RSA-assumption is broken.

When invoking the zero-knowledge simulator we must program the random oracle O at some points. In principle it could be the case that the adversary has already asked for the value at the point we need to program, and this would prohibit programming. A standard observation is that an adversary can only query the random oracle at a polynomial number of points, and the point on which the random oracle is programmed is always chosen randomly from an exponentially large space. Thus, it is easy to see that programming the oracle fails with negligible probability. Similarly, in principle it could be the case that the adversary guesses the value of the random oracle at some point, on which the random oracle is never queried. It is easy to see that also this happens with negligible probability. Thus,

in the remainder of the proof we assume without loss that the adversary has never queried the random oracle O at any point x on which we must give $\mathsf{O}(x)$ a specific value, and that the adversary never outputs a point x and a corresponding value $\mathsf{O}(x)$ without querying the oracle on x. This convention simplifies our exposition.

We consider hierarchical anonymity and hierarchical traceability separately.

HIERARCHICAL ANONYMITY. Let A be any adversary. Denote by H^b the machine that simulates the hierarchical anonymity experiment $\mathsf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$ using A as a blackbox.

Denote by H^b_{o} the machine that is identical to H^b except that the open oracle $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ is simulated as follows. Consider a query on the form (α, m, σ), where α is on depth l. If $\mathsf{HVf}(T, hpk, m, \sigma) = 0$, return $\perp$. Otherwise assume that the signature is on the form

$$\sigma = \left((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \pi\right) \ .$$

The machine H^b_{o} computes $D^{\mathsf{cs}}_X(C_\delta)$ and if the result does not equal y_{α_δ} for some $\alpha_\delta \in \mathcal{L}(T)$, the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle is instructed to return $\perp$. Suppose now that y_{α_δ} is on the expected form. Then there is a path $\alpha_0, \ldots, \alpha_\delta$ in the tree T corresponding to α_δ and the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle is instructed to return α_{l+1} if $\beta = \alpha_l$ and $\perp$ otherwise. In principle this answer could be incorrect, but we prove that it is not.

Claim 1. The absolute value $|\Pr[H^b = 1] - \Pr[H^b_{\mathrm{o}} = 1]|$ is negligible under the DL-assumption and the strong RSA-assumption.

Proof. Assume that the claim is false. Then with non-negligible probability some query to the open oracle $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ is answered incorrectly.

Let $p(\kappa)$ denote the running time of A. Then it follows that A asks the simulated $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle at most $p(\kappa)$ queries. Denote by T_l the machine that simulates H^b_{o} until $l-1$ queries have been answered by the simulated $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle, and then halts outputting the lth query. We say that a query is *difficult* if it is answered incorrectly. We show that T_l outputs a difficult query with negligible probability for $l = 0, \ldots, p(\kappa)$. The union bound then implies that all queries are answered correctly with overwhelming probability and the claim follows.

The statement is clearly true for T_0, since its output is empty. Suppose now that the statement is true for T_l for $l < s$, but false for T_s. Thus, T_s outputs a difficult query with non-negligible probability.

Consider a query (α, m, σ) such that

$$\sigma = ((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, (\gamma, c, e)) \ .$$

There are two sorts of difficult queries. Either

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C})$$

does not belong to $L_{R_{\mathrm{HGS}}}$, or it does, but $(u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta$ is not a chain on the form $(E_{y_{\alpha_0}}(y_{\alpha_1}), E_{y_{\alpha_0}}(1), \ldots, E_{y_{\alpha_{\delta-1}}}(y_{\alpha_\delta}), E_{y_{\alpha_{\delta-1}}}(1)), E_Y^{\mathsf{cs}}(y_{\alpha_\delta})$ for any path $\alpha_0, \ldots, \alpha_\delta$ from the root $\rho = \alpha_0$ in T to a leaf $\alpha = \alpha_\delta \in \mathcal{L}(T)$.

Note that if the tuple above does belong to $L_{R_{\mathrm{HGS}}}$, then $(u'_l, v'_l) = E_{(y_{\alpha_l}, g)}(1, r'_l)$ for some r'_l and there exists no $\beta \in \mathcal{V}(T)$ with $\beta \neq \alpha_l$ such that $(v'_l)^{x_\beta} = u'_l$.

Suppose first that T_s outputs a query of the first type with non-negligible probability. Then the soundness of the computationally convincing proof of knowledge π_{hgs} is broken by the interactive prover P^*_{hgs} defined as follows. It accepts special parameters

$$\Lambda = ((\mathbf{N}, \mathbf{g}, \mathbf{y}), (q_0, g_1, y_1, g_2, y_2, g_3, y_3, g_{\mathbf{N}}, y_{\mathbf{N}}))$$

as input. Then it chooses $\mathbf{k} \in \mathrm{QR}_{\mathbf{N}}$ randomly, and invokes the simulator from the proof of the Cramer-Shoup signature scheme on $(\mathbf{N}, \mathbf{k})$ to generate *spk*. The remaining parameters of the experiment simulated to A are generated as in the real experiment, except the signatures $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$. They are computed by invoking the simulated signature oracle of the Cramer-Shoup signature simulator.

The prover P^*_{hgs} chooses a random index $1 \leq i \leq p(\kappa)$ and simulates T_s until A makes its ith query to the random oracle O. Denote the ith new query by $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma)$. The prover P^*_{hgs} then outputs the ith query as its choice of common input and the first message γ in the proof and waits for a challenge c from the honest verifier V_{hgs} of protocol π_{hgs}. It instructs O to output c and continues the simulation of T_s until it gives output (α, m, σ). Note that here we only program the oracle on new queries. If σ is on the form

$$\sigma = ((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, (\gamma, c, e))$$

it outputs e, and otherwise 0. Note that the index i is chosen independently at random. Thus, the probability that the challenge value c in the final output of T_s corresponds to the ith query to O conditioned on the event that the final output of T_s is a difficult query is at least $1/p(\kappa)$.

It is proved in [54] that the distributions of the key *spk* and the signatures $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$ are statistically close to the distributions of a real public key and real signatures. Thus, we conclude that

$$\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V_{\mathrm{hgs}}}_{P^*_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}) \notin L_{R_{\mathrm{HGS}}}]$$

is non-negligible. This contradicts the soundness of the protocol π_{hgs}. Formally, the contradiction follows from combining Proposition 16.39 with Proposition 2.32.

Assume now that T_s outputs a query of the second type with non-negligible probability.

The idea is to execute the extractor of the proof of knowledge of the π_{hgs} protocol to find a Cramer-Shoup signature of a list of public keys that does not correspond to a signer for which the adversary has requested the secret key. This would contradict the CMA-security of the Cramer-Shoup signature scheme.

The problem is that, although the extractor will output a witness in expected polynomial time with non-negligible probability and part of the witness is indeed a Cramer-Shoup signature of a list of public keys, the definition of a computationally convincing proof of knowledge gives no guarantee that the extracted signature is not a signature of list of public keys already given to the adversary.

We resolve this technicality by describing a special hypothetical protocol π'_{hgs} in Section 16.5.1 of the next chapter, and show that it is a computationally convincing proof of knowledge. The protocol is identical to π_{hgs} except that the verifier only accepts a proof corresponding to no signer. We show that a prover in the π'_{hgs} protocol defined in Section 16.5.1 can be constructed from T_s. Then we invoke the extractor and conclude that the extracted Cramer-Shoup signature implies that the CMA-security of the Cramer-Shoup signature scheme is broken.

Denote by $P^{\emptyset}_{\text{hgs}}$ the prover in the protocol π'_{hgs} identical to P^{*}_{hgs} except for the following changes. It accepts as input

$$\Lambda' = ((\mathbf{N}, \mathbf{g}, \mathbf{y}), (q_0, g_1, y_1, g_2, y_2, g_3, y_3, g_{\mathbf{N}}, y_{\mathbf{N}}), (x_\alpha, y_\alpha)_{\alpha \in \mathcal{V}(T)}) \ .$$

It chooses $\mathbf{k} \in \text{QR}_{\mathbf{N}}$ randomly and invokes the simulator from the proof of the Cramer-Shoup signature scheme on $(\mathbf{N}, \mathbf{k})$ to generate *spk*. Finally, it interacts with the honest verifier V'_{hgs} of the π'_{hgs} protocol instead of the honest verifier V_{hgs} of the π_{hgs} protocol.

It follows that there exists a constant c_1 and an infinite index set $\mathcal{N}$ such that for $\kappa \in \mathcal{N}$

$$\begin{aligned}
\kappa^{-c_1} &\leq \Pr[\text{Acc}_{V_{\text{hgs}}}(\text{view}^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}, r_{\text{v}})) = 1] \\
&\leq \Pr[\text{Acc}_{V_{\text{hgs}}}(\text{view}^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}, r_{\text{v}})) = 1 \mid \delta^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\qquad \cdot \Pr[\delta^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\qquad + \frac{1}{2}\kappa^{-c_1} \ .
\end{aligned}$$

Thus, $\Pr[\delta^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda, r_{\text{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq \frac{1}{2}\kappa^{-c_1}$ and from Proposition 16.40 we conclude that there exists an extractor $\mathcal{X}^{P^{\emptyset}_{\text{hgs}}}$ and a polynomial $p(\kappa)$ such that

$$\Pr[(I_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}), \mathcal{X}^{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}})) \in R'_{\text{HGS}} \mid \delta^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq 1 - \epsilon(\kappa)$$

for some negligible function $\epsilon(\kappa)$, and such that the expected running time of $\mathcal{X}^{P^{\emptyset}_{\text{hgs}}}$ on inputs (Λ', r_{p}) such that $\delta^{V'_{\text{hgs}}}_{P^{\emptyset}_{\text{hgs}}}(\Lambda', r_{\text{p}}) \geq \frac{1}{2}\kappa^{-c_1}$ is bounded by some polynomial $t(\kappa)$.

Denote by A_{sig} the algorithm that on input $(\mathbf{N}, \mathbf{k})$ generates the remainder of the parameters in Λ', chooses $r_{\text{p}} \in \{0,1\}^*$ randomly and simulates $\mathcal{X}^{P^{\emptyset}_{\text{hgs}}}$ on these

inputs except that $P^{\emptyset}_{\text{hgs}}$ uses the value of $\mathbf{k}$ instead of generating it. Furthermore, $\mathcal{X}^{P^{\emptyset}_{\text{hgs}}}$ is simulated for at most $4\kappa^{c_1}t(\kappa)$ steps. Note that on inputs such that the expected running time is $t(\kappa)$, Markov's inequality implies that the the probability that the simulation is not completed is bounded by $\frac{1}{4}\kappa^{-c_1}$.

If the simulation is completed it interprets the common input $I_{P^{\emptyset}_{\text{hgs}}}(\Lambda, r_{\text{p}})$ as

$$\big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}\big) \in G_{q_3}^{4\delta} \times G_{q_3}^{4} \times \text{QR}_{\mathbf{N}}^{5}$$

and it interprets the output of $\mathcal{X}^{P^{\emptyset}_{\text{hgs}}}$ as a tuple

$$((\tau_0, \tau'_0, \ldots, \tau_{\delta-1}, \tau'_{\delta-1}, \tau_\delta), (\tau, \zeta, \tau', \zeta', \psi, \varepsilon))$$

in $\mathbb{Z}_{q_3}^{2\delta+1} \times [0, 2^{\kappa_r}\mathbf{N} - 1]^5 \times [2^\kappa, 2^{\kappa+1} - 1]$. Finally, it outputs $(\varepsilon, \mathbf{v}/\mathbf{y}^{\boldsymbol{\tau}}, \mathbf{v}'/\mathbf{y}^{\boldsymbol{\tau}'})$. If on the other hand the simulation is not completed it outputs $\perp$.

We conclude that A_{sig} outputs a Cramer-Shoup signature of $(\gamma_1, \ldots, \gamma_\delta)$ that does not equal $(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$ for any path $\alpha_0, \ldots, \alpha_\delta$ in T with probability at least

$$\frac{1}{2\kappa^{c_1}}\left(\frac{1}{2\kappa^{c_1}} - \frac{1}{4\kappa^{c_1}} - \epsilon(\kappa)\right)$$

which is non-negligible. Note that by construction, the simulated Cramer-Shoup signature oracle has never been queried on $(\gamma_1, \ldots, \gamma_\delta)$. It follows from [54] that this contradicts the strong RSA-assumption, or the collision-freeness of $\mathcal{SH}$ or $\mathcal{CHP}$. From Proposition 3.13 we know that finding a collision in $\mathcal{SH}$ contradicts the strong RSA-assumption, and from Proposition 3.6 we know that finding a collision in $\mathcal{CHP}$ contradicts the DL-assumption. □

Denote by $H^b_{\text{o,g}}$ the machine that is identical to H^b_{o} except that it chooses two leaves $\beta^{(0)}_\delta$ and $\beta^{(1)}_\delta$ randomly. Let $\beta^{(0)}_\delta, \ldots, \beta^{(0)}_t$ and $\beta^{(1)}_\delta, \ldots, \beta^{(1)}_t$ be the paths to their least common ancestor $\beta^{(0)}_t = \beta^{(1)}_t$. The machine $H^b_{\text{o,g}}$ outputs 0 if A requests $x_{\beta^{(b)}_l}$ for some $b \in \{0, 1\}$ and $t \leq l \leq \delta$. It also outputs 0 if $(\alpha^{(0)}, \alpha^{(1)}) \neq (\beta^{(0)}_\delta, \beta^{(1)}_\delta)$.

Claim 2. $\Pr[H^b_{\text{o}} = 1] = \Pr[H^b_{\text{o,g}} = 1]/|\mathcal{L}(\kappa)|^2$.

Proof. If A does not output indices $\alpha^{(0)}, \alpha^{(1)} \in \mathcal{L}(T)$ the output is 0 in both simulations. Suppose it does and let $\alpha^{(0)}_\delta, \ldots, \alpha^{(0)}_{t'}$ and $\alpha^{(1)}_\delta, \ldots, \alpha^{(1)}_{t'}$ be the paths to their least common ancestor $\alpha^{(0)}_{t'} = \alpha^{(1)}_{t'}$. If A ever asks for the secret key of $x_{\alpha^{(b)}_l}$ for any $l = t, \ldots, \delta$ the output is 0 in both simulations. Suppose it does not ask for such keys. Then we have $(\alpha^{(0)}, \alpha^{(1)}) = (\beta^{(0)}_\delta, \beta^{(1)}_\delta)$ with probability $1/|\mathcal{L}(\kappa)|^2$, since the indices $\beta^{(0)}_\delta$ and $\beta^{(1)}_\delta$ are chosen independently at random. The claim follows. □

Denote by $H^b_{\mathrm{o,g,nddh}}$ the machine that is identical to $H^b_{\mathrm{o,g}}$ except for the following. In Step 6 in the key generation algorithm is simulated honestly except that $y_{\beta_l^{(b)}}$, for $l = t, \ldots, \delta$, are instead defined as follows using a randomly chosen elements $(D_{1,l}, D_{2,l}, D_{3,l}, D'_{2,l}, D'_{3,l}) \in G^5_{q_3}$ for $l = t, \ldots, \delta - 1$. The public keys are defined by

$$y_{\beta_l^{(b)}} = D_{1,l} \ .$$

Note that the simulated hierarchical group signature of m is only computed if $(\alpha^{(0)}, \alpha^{(1)}) = (\beta_\delta^{(0)}, \beta_\delta^{(1)})$. The single query m to the $\mathsf{HSig}(T, hpk, hsk(\alpha^{(b)}), \cdot)$ oracle is simulated as follows. To simplify the exposition we write α_l instead of $\alpha_l^{(b)}$ as in Experiment 13.2. The machine $H^b_{\mathrm{o,g,nddh}}$ chooses $\boldsymbol{\tau}, \boldsymbol{\zeta}, \boldsymbol{\tau}', \boldsymbol{\zeta}', \boldsymbol{\psi} \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly and computes

$$\begin{aligned}
&(u_l, v_l, u'_l, v'_l) = (D_{2,l}, y_{\alpha_{l+1}} D_{3,l}, D'_{2,l}, D'_{3,l}), \quad \text{for } l = 0, \ldots, \delta - 1 \ , \\
&C_\delta = E^{\mathsf{cs}}_Y(y_{\alpha_\delta}, r) \ , \quad \text{and} \\
&(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^{\boldsymbol{\zeta}}, \mathbf{g}^{\boldsymbol{\tau}}), \quad (\mathbf{u}', \mathbf{v}') = (\mathbf{g}^{\boldsymbol{\zeta}'}, \mathbf{g}^{\boldsymbol{\tau}'}), \quad \mathbf{C} = \mathbf{g}^{\boldsymbol{\psi}} \ .
\end{aligned}$$

To construct the proof π, $H^b_{\mathrm{o,g,nddh}}$ simply invokes the simulator for the proof of knowledge. This is guaranteed to exist by Proposition 16.38. To do this the random oracle O is programmed. As explained at the beginning of the proof this is not a problem since the input to the random oracle is chosen randomly from an exponentially large space.

Claim 3. The absolute value $|\Pr[H^b_{\mathrm{o,g}} = 1] - \Pr[H^b_{\mathrm{o,g,nddh}} = 1]|$ is negligible under the DL-assumption.

Proof. Recall the definition of the variant DDH-assumption from Section 3.5. A tuple $(D_1, D_2, D_3, D'_2, D'_3)$ is called a DDH-tuple if $\log_{g_3} D_3 = \log_{g_3} D_1 \log_{g_3} D_2$ and $\log_{g_3} D'_3 = \log_{g_3} D_1 \log_{g_3} D'_2$.

Denote by $H^{b,i}_{\mathrm{o,g,nddh}}$ the machine that simulates $H^b_{\mathrm{o,g,nddh}}$ except that it uses random triples only for $t \geq l > i$. The distributions of the simulated $(\mathbf{u}, \mathbf{v})$, $(\mathbf{u}', \mathbf{v}')$, and $\mathbf{C}$ are statistically close to those in the real experiment. From Proposition 16.38, i.e., the statistical zero-knowledge property of the protocol π_{hgs}, we have that $|\Pr[H^{b,-1}_{\mathrm{o,g,nddh}} = 1] - \Pr[H^b_{\mathrm{o,g}} = 1]|$ is negligible. It follows that the distributions of $H^{b,-1}_{\mathrm{o,g,nddh}}$ and $H^{b,\delta-1}_{\mathrm{o,g,nddh}}$ are statistically close to the distributions of $H^b_{\mathrm{o,g}}$ and $H^b_{\mathrm{o,g,nddh}}$ respectively.

Suppose that $|\Pr[H^b_{\mathrm{o,g}} = 1] - \Pr[H^b_{\mathrm{o,g,nddh}} = 1]|$ is non-negligible. Then it follows from a hybrid argument that there exists a fixed i such that

$$|\Pr[H^{b,i}_{\mathrm{o,g,nddh}} = 1] - \Pr[H^{b,i+1}_{\mathrm{o,g,nddh}} = 1]|$$

is non-negligible.

Denote by A' the adversary in the variant DDH-experiment of Lemma 3.8 that proceeds as follows. On input $(q_3, g_3, D_1, D_2, D_3, D'_2, D'_3)$ it computes q_0, q_1, q_2 from

q_3 and then simulates $H^{b,i}_{\mathrm{o,g,ddh}}$ on these values except that instead of generating $(D_{1,l}, D_{2,l}, D_{3,l}, D'_{2,l}, D'_{3,l})$ it uses $(D_1, D_2, D_3, D'_2, D'_3)$. We conclude that the distribution of the variable $A'(q_3, g_3, D_1, D_2, D_3, D'_2, D'_3)$ is identical to the distribution of $H^{b,i+1}_{\mathrm{o,g,nddh}}$ or $H^{b,i}_{\mathrm{o,g,nddh}}$ depending on if $(D_1, D_2, D_3, D'_2, D'_3)$ is a DDH-tuple or not. Thus, by Lemma 3.8, A' contradicts the DDH-assumption, and the claim holds. □

Claim 4. The absolute value $|\Pr[H^0_{\mathrm{o,g,nddh}} = 1] - \Pr[H^1_{\mathrm{o,g,nddh}} = 1]|$ is negligible under the DDH-assumption.

Proof. Suppose that the claim is false. Then the CCA2-security of the Cramer-Shoup cryptosystem is broken by the adversary A' taking part in Experiment 2.17 and defined as follows. It simulates $H^0_{\mathrm{o,g,nddh}}$ except that it waits for a Cramer-Shoup public key Y over G_{q_3}, computes q_0, q_1, q_2, and uses these values in the simulation. Thus, it does not know the private key X to the Cramer-Shoup cryptosystem. Instead of computing $D^{\mathsf{cs}}_X(C_\delta)$ to simulate the answer to a query to the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle, it queries its decryption oracle to find this value. When computing the hierarchical group signature of m, it hands $y_{\beta^{(0)}_\delta}$ and $y_{\beta^{(1)}_\delta}$ to the encryption oracle and receives a challenge cryptotext C_δ. It then uses this challenge cryptotext to construct the simulated hierarchical group signature.

It follows that $\mathsf{Exp}^{\mathsf{cca2}-b}_{\mathcal{CS}^{\mathsf{cs}},A'}(\kappa)$ is identically distributed to $H^b_{\mathrm{o,g,nddh}}$, and the CCA2-security of the Cramer-Shoup cryptosystem is broken. This contradicts Proposition 3.10 and the claim holds. □

The hierarchical anonymity now follows immediately from Claims 1–4.

Hierarchical Traceability. Let A be any adversary. Denote by H the machine that simulates the experiment $\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$. Denote by H_{p} the machine that is identical to H except that it simulates the $\mathsf{HSig}(\cdot, T, hpk, hsk(\cdot))$-oracle as follows. The first step is simulated honestly. In Step 2 $(\mathbf{u}, \mathbf{v})$, $(\mathbf{u}', \mathbf{v}')$ and $\mathbf{C}$ are replaced by $(\mathbf{g}^{\boldsymbol{\zeta}}, \mathbf{g}^{\boldsymbol{\tau}})$, $(\mathbf{g}^{\boldsymbol{\zeta}'}, \mathbf{g}^{\boldsymbol{\tau}'})$ and $\mathbf{g}^{\boldsymbol{\psi}}$ respectively with randomly chosen $\boldsymbol{\zeta}, \boldsymbol{\tau}, \boldsymbol{\zeta}', \boldsymbol{\tau}', \boldsymbol{\psi} \in [0, 2^{\kappa+\kappa_r} - 1]$. In Step 3, the simulator of the proof of knowledge guaranteed to exist by Proposition 16.38 is invoked to construct π. This requires that the random oracle O is programmed, but this is not a problem, since the query to the random oracle is chosen randomly from an exponentially large space.

Claim 5. The absolute value $|\Pr[H = 1] - \Pr[H_{\mathrm{p}} = 1]|$ is negligible.

Proof. The distributions of $(\mathbf{u}, \mathbf{v})$, $(\mathbf{u}', \mathbf{v}')$, and $\mathbf{C}$ in simulated hierarchical signatures are statistically close to those in the real experiment. The statistical zero-knowledge simulator guaranteed to exist by Proposition 16.38 implies that the distributions of the simulated proofs are statistically close to those in the experiment. The claim follows. □

Claim 6. The probability $\Pr[H_{\mathrm{p}} = 1]$ is negligible.

Proof. The idea of the proof is to execute the extractor of the π_{hgs} protocol to find a Cramer-Shoup signature on a list of public keys that does not correspond to a signer for which the adversary has requested the private key.

The problem is that the extractor does not guarantee that the extracted witness has any particular properties, and we need a witness that contains not just any Cramer-Shoup signature, but a signature on a message on which the simulated Cramer-Shoup oracle has never been queried. Note that this problem is similar to the problem we encountered when proving that the simulated opening oracle behaved correctly in the proof of hierarchical anonymity. We resolve the problem in a similar way and use the extractor of the slightly modified protocol π'_{hgs}.

Denote by T_β the tree T except that a leaf β is removed. We also write $T_\emptyset$ for the tree T. This allows us to consider two different cases at once. Denote by $p(\kappa)$ the running time of A. We construct a prover P^β_{hgs} to the π'_{hgs} protocol. The prover P^β_{hgs} is given the special parameter

$$\Lambda' = ((\mathbf{N}, \mathbf{g}, \mathbf{y}), (q_0, g_1, y_1, g_2, y_2, g_3, y_3, g_{\mathbf{N}}, y_{\mathbf{N}}), (x_\alpha, y_\alpha)_{\alpha \in \mathcal{V}(T_\beta)})$$

as input. If a leaf is removed it extends T_β to T if $\beta \neq \emptyset$ and generates x_β and y_β honestly. It also chooses $\mathbf{k} \in \text{QR}_{\mathbf{N}}$ randomly and invokes the simulator from the proof of the Cramer-Shoup signature scheme on $(\mathbf{N}, \mathbf{k})$ to generate spk. Then it simulates H_{p} on these values except for the following.

Whenever A requests $hsk(\alpha)$ for a leaf $\alpha \in \mathcal{L}(T)$ with $\alpha \neq \beta$, P^β_{hgs} invokes the simulated Cramer-Shoup signature oracle on input $(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$, where $\alpha_0, \ldots, \alpha_\delta$ is the path from the root $\rho = \alpha_0$ to the leaf $\alpha_\delta = \alpha$. The simulated Cramer-Shoup signature oracle then returns a signature $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$, which is handed to A. If A requests $hsk(\beta)$ it is handed $\perp$.

The prover P^β_{hgs} chooses a random index $1 \leq i \leq p(\kappa)$ and simulates H_{p} until A makes the ith new query to the random oracle O. Let the ith query to O be given by $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma)$. The prover P^β_{hgs} outputs

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma)$$

and waits for a challenge c from the honest verifier V'_{hgs} of the protocol π'_{hgs}.

The prover P^β_{hgs} programs O to output c and continues the simulation of H_{p} until A outputs a pair (m, σ). Programming O is not a problem since only new queries are considered. If σ is on the form

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma, c, d)$$

it outputs d, and otherwise 0. Note that the index i is chosen independently at random. Thus, the probability that the challenge value c in the final output of A corresponds to the ith query to O conditioned on the event that $c = \mathsf{O}((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma)$ is at least $1/p(\kappa)$. Recall from the beginning of the proof of the theorem that we do not worry that the adversary guesses the value of the random oracle at any point. This completes the description of P^β_{hgs}.

Suppose first that the probability that A outputs (m, σ) and

$$V_{\mathrm{hgs}}((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma, c, d) = 1 \text{ and still}$$
$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \notin L_{R_{\mathrm{HGS}}}$$

is non-negligible. This implies that

$$\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \notin L_{R_{\mathrm{HGS}}}]$$

is non-negligible. This contradicts the soundness of the protocol π_{hgs}. Formally, the contradiction follows from combining Proposition 16.39 with Proposition 2.32.

Consider a σ such that $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \in L_{R_{\mathrm{HGS}}}$. Then we have $(u_l, v_l) = E_{(\gamma_l, g)}(\gamma_{l+1}, r_l)$ and $(u'_l, v'_l) = E_{(\gamma_l, g)}(1, r'_l)$ for some γ_l, γ_{l+1}, r_l and r'_l. Thus, there does not exist any $\alpha' \in \mathcal{V}(T)$ with $y_{\alpha'} \neq \gamma_l$ such that $(v'_l)^{x_{\alpha'}} = u'_l$.

We conclude that if $\Pr[H_{\mathrm{p}} = 1]$ is non-negligible, then there also exists a β such the probability that $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \in L_{R_{\mathrm{HGS}}}$ and $\mathsf{HVf}(T, hpk, m, \sigma) = 1$ and $\alpha_\delta = \beta$ and A is never given $hsk(\beta) \neq \perp$ is non-negligible. Remember that we allow β to be either the index of a signer or equal to $\emptyset$. This implies that $\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1]$ is non-negligible, since the distributions of the simulated public key *spk* and the simulated signatures $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$ are statistically close to the corresponding distributions in the simulation of H_{p}.

More precisely there exists a constant c_1 and an infinite index set $\mathcal{N}$ such that for $\kappa \in \mathcal{N}$

$$\begin{aligned}
\kappa^{-c_1} &\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1] \\
&\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \mid \delta^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\quad \cdot \Pr[\delta^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\quad + \frac{1}{2}\kappa^{-c_1} \ .
\end{aligned}$$

Thus, $\Pr[\delta^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq \frac{1}{2}\kappa^{-c_1}$ and from Proposition 16.40 we conclude that there exists an extractor $\mathcal{X}^{P^\beta_{\mathrm{hgs}}}$ and a polynomial $t(\kappa)$ such that

$$\Pr[(I_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}), \mathcal{X}^{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}})) \in R'_{\mathrm{HGS}} \mid \delta^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq 1 - \epsilon(\kappa)$$

for some negligible function $\epsilon(\kappa)$, and such that the expected running time of $\mathcal{X}^{P^\beta_{\mathrm{hgs}}}$ on inputs $(\Lambda', r_{\mathrm{p}})$ such that $\delta^{V'_{\mathrm{hgs}}}_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}$ is bounded by some polynomial

$t(\kappa)$. Denote by A' the algorithm that on input $(\mathbf{N}, \mathbf{k})$ generates the remainder of the parameters in Λ', chooses $r_{\mathrm{p}} \in \{0,1\}^*$ randomly and simulates $\mathcal{X}^{P^{\beta}_{\mathrm{hgs}}}$ on these inputs for at most $4\kappa^{c_1} t(\kappa)$ steps. Note that if the expected running time of $\mathcal{X}^{P^{\beta}_{\mathrm{hgs}}}$ is $t(\kappa)$ on a given input $(\Lambda', r_{\mathrm{p}})$, Markov's inequality implies that the the probability that the simulation is not completed is bounded by $\frac{1}{4}\kappa^{-c_1}$.

Using the union bound we conclude that A' outputs a Cramer-Shoup signature of a message $(\gamma_1, \ldots, \gamma_\delta)$ that does not equal $(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$ for any path $\alpha_0, \ldots, \alpha_\delta$ in T_β with probability at least

$$\frac{1}{2\kappa^{c_1}} \left(\frac{1}{2\kappa^{c_1}} - \frac{1}{4\kappa^{c_1}} - \epsilon(\kappa) \right)$$

which is non-negligible. By construction the simulated Cramer-Shoup signature oracle has never been queried on the list of public keys $(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$ corresponding to the path to S_β. Thus, A' breaks the CMA-security of the Cramer-Shoup signature scheme. Proposition 3.10 implies that this contradicts the strong RSA-assumption, or the collision-freeness of the Shamir hash function $\mathcal{SH}$ or the Chaum-van Heijst-Pfitzmann hash function $\mathcal{CHP}$. From Proposition 3.13 we know that the first event contradicts the strong RSA-assumption, and from Proposition 3.6 we know that the second event contradicts the DL-assumption. Thus, the claim holds. □

Conclusion of Proof. To conclude the proof it suffices to note that we have proved that both $\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T)$ and $\mathsf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$ are negligible. □

Remark 15.9. It is shown in [17] that it is necessary to use a CCA2-secure cryptosystem to form a group signature scheme. Still we only use a CCA2-secure cryptosystem for the leaves. This apparent contradiction is resolved by noting that since the public keys y_α are distinct, and we may identify the leaves with their paths in the tree, any query to the $\mathsf{HOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle for intermediate levels of the tree can be answered using a single query to the decryption oracle for the CCA2-secure Cramer-Shoup cryptosystem used to encrypt leaves.

Remark 15.10. The exposition here differs from the exposition in [140]. There it is not taken into account that the protocol π_{hgs} is a computationally convincing proof of knowledge and not a proof of knowledge. Furthermore, it is not clear from the proof in [140] that it is safe to use the RSA-modulus of the Cramer-Shoup signature scheme to form integer commitments and to use it as a one-way hash function in the signature scheme. This inter-dependency could potentially be dangerous. These deficiencies are eliminated here and this will be reflected in the final full version of the paper as well.

Chapter 16

Construction of the Proof of Knowledge

In this chapter we describe the proof of knowledge needed in Chapter 15. We give zero-knowledge proofs of knowledge for a number of subprotocols which combined gives the proof of knowledge we need to apply the Fiat-Shamir heuristic to get a signature scheme in the random oracle model. This chapter is based on the paper by Trolin and Wikström [141].

Our protocols are based on a variety of proof techniques including: proofs of knowledge of exponents, double-decker exponentiation, equality of exponents over distinct groups, interval proofs, and equality of integer exponents over an RSA-modulus.

The exposition is divided into a number of subsections. First we describe the protocols that execute in the groups G_{q_1}, G_{q_2}, and G_{q_3}. Then we describe a protocol that executes in both G_{q_1} (or $G_{\mathbf{N}}$) and in $\mathbb{Z}^*_{\mathbf{N}}$. This is followed descriptions of the protocols that execute in $\mathbb{Z}^*_{\mathbf{N}}$. Finally, the combined protocol is described. For intuition on how the proof is constructed we refer the reader to Section 15.1.

Although we focus on efficiency, in some cases we have chosen to divide the protocol into subprotocols for clarity, thus sacrificing some efficiency. Since the by far most time-consuming part of the protocol are the proofs of exponential relations, where to our knowledge the most efficient known method is based on cut-and-choose techniques, saving a few exponentiations in other parts of the protocol yields little in terms of overall performance.

In some protocols we use the additional security parameters κ_c and κ_r. The first parameter normally decides the number of bits in a challenge, and the second parameter is used to pad exponents with additional random bits to achieve statistical zero-knowledge, when the order of a group is not known. For example, if we wish to compute a commitment $\mathbf{y}^r\mathbf{g}^b$ of a bit b using randomness r, where $\mathbf{N}$ is a κ-bit RSA-modulus and $\mathbf{g}$ and $\mathbf{y}$ are random elements in $\mathrm{QR}_{\mathbf{N}}$, then the random exponent should be chosen in $[0, 2^{\kappa+\kappa_r} - 1]$ to achieve a statistically hiding commit-

ment scheme. The parameter κ_r also decides the completeness of several protocols. It suffices if $2^{-\kappa_c}$ and $2^{-\kappa_r}$ are negligible in the main security parameter κ.

Sometimes it is more convenient to keep the committed number in the base rather than in the exponent. In this case a commitment to an element $\mathbf{z} \in \mathrm{QR}_{\mathbf{N}}$ can be computed as

$$(\mathbf{y}^r \mathbf{g}^s, \mathbf{y}^r \mathbf{z}) \ ,$$

where $r, s \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ are randomly chosen. We use this trick also over the groups G_{q_1}, G_{q_2}, and G_{q_3}.

Remark 16.1. The exposition here differs from the exposition in the preliminary full version [140] of [141] in one important aspect. In [140] the various special cases above are treated rather informally. It is never clearly stated that the protocols are in fact computationally convincing proofs of knowledge, and not proofs of knowledge. This deficiency is eliminated here and this will be reflected in the final full version of the paper as well.

16.1 A Simplifying Convention

Most subprotocols below are strictly speaking not proofs of knowledge of their private input from the prover. It may happen that an extractor finds elements on the form listed below instead of a witness. To simplify the exposition we do not state this explicitly in each lemma. Instead we point to one of the special cases below whenever such a case occurs in the analysis of each protocol. Then when we combine all subprotocols we state explicitly the dependence on the special parameters.

We stress that we do not expect any adversary to find a witness of the type below. In fact if an adversary finds a witness of the type below with non-negligible probability, then the adversary can be used to break either the DL-assumption or the strong RSA-assumption. Thus, each subprotocol is in fact a computationally convincing proof of knowledge of the private input of the prover as stated in the protocol for some special input.

Another simplifying assumption is that we assume that any element $\mathbf{A} \in \mathbb{Z}_{\mathbf{N}}$ can be inverted modulo $\mathbf{N}$. Note that if this is not the case $\mathbf{A}$ is a non-trivial factor of $\mathbf{N}$, i.e., Case 7 is satisfied. We do not mention this case explicitly every time we invert an element.

1. An element $\eta \in \mathbb{Z}_{q_1}$ such that $y_1 = g_1^{\eta}$.
2. An element $\eta \in \mathbb{Z}_{q_2}$ such that $y_2 = g_2^{\eta}$.
3. An element $\eta \in \mathbb{Z}_{q_3}$ such that $y_3 = g_3^{\eta}$.
4. An element $\eta \in \mathbb{Z}_{\mathbf{N}}$ such that $y_{\mathbf{N}} = g_{\mathbf{N}}^{\eta}$.

5. Integers $\eta_0 \neq 0$ and η_1, η_2 not both zero and $\mathbf{b} \in \mathbb{Z}_{\mathbf{N}}^*$ such that η_0 does not divide both η_1 and η_2, and $\mathbf{b}^{\eta_0} = \mathbf{g}^{\eta_1}\mathbf{y}^{\eta_2}$.

6. Integers η_0, η_1 not both zero such that $\mathbf{g}^{\eta_0}\mathbf{y}^{\eta_1} = 1$.

7. An integer η such that $1 < |\eta| < \mathbf{N}$ and $\eta \mid \mathbf{N}$.

For simplicity we also assume that each protocol is given the representation of the appropriate group as common input, i.e., if the protocol executes in G_{q_1}, G_{q_2}, or G_{q_3} it is given q_0 as input, and if it executes in $\mathrm{QR}_{\mathbf{N}}$ or $G_{\mathbf{N}}$ it is given $\mathbf{N}$ as input. We do not state this explicitly to avoid cluttering the exposition.

16.2 Protocols in Groups of Known Prime Order

The goal of this section is to provide subprotocols that can be used to prove knowledge of $\gamma_1, \ldots, \gamma_\delta$ and $\tau_0, \tau_0', \ldots, \tau_{\delta-1}, \tau_{\delta-1}', \tau_\delta$ satisfying the parts of the relation R_{HGS} that are defined exclusively over G_{q_1}, G_{q_2}, and G_{q_3}. Most of the ideas we use in this section have appeared in various forms in the literature.

We begin our program by considering a problem related to that of proving that a list of cryptotexts is chained properly.

Protocol 16.2 (Chained Cryptotexts).
COMMON INPUT: $y_0, g, y \in G_q$ and $\big((u_l, v_l, u_l', v_l'), (\mu_l, \nu_l)\big)_{l=0}^{\delta-1} \in G_q^{6\delta}$
PRIVATE INPUT: $r_l, r_l', s_l, t_l \in \mathbb{Z}_q$ for $l = 0, \ldots, \delta-1$ and $y_l \in G_q$ for $l = 1, \ldots, \delta$ such that $(u_l, v_l, u_l', v_l') = (E_{(y_l,g)}(y_{l+1}, r_l), E_{(y_l,g)}(1, r_l')) = (y_l^{r_l}, g^{r_l}y_{l+1}, y_l^{r_l'}, g^{r_l'})$ and $(\mu_l, \nu_l) = (y^{t_l}g^{s_l}, y^{s_l}y_{l+1})$.

1. The prover chooses $a_l, a_l', a_l'' \in \mathbb{Z}_q$ randomly and computes

$$A_{1,l} = y^{a_l'}g^{a_l}\mu_l^{r_{l+1}} \ , \ A_{2,l} = y^{a_l}\nu_l^{r_{l+1}} \ , \text{ and } \ A_{3,l} = y^{a_l''}g^{r_{l+1}}$$

   for $l = 0, \ldots, \delta-2$.

2. The prover chooses $b_l, b_l', b_l'', e_l, f_l, h_l, i_l, j_l, w_l, w_l', k_l, k_l' \in \mathbb{Z}_q$ randomly and computes $B_0 = y_0^{e_0}$,

$$B_{1,l} = g^{e_l}y^{i_l} \quad \text{and} \quad B_{2,l} = g^{i_l}y^{j_l}$$

   for $l = 0, \ldots, \delta-1$, and

$$\begin{aligned}
B_{3,l} &= y^{b_l'}g^{b_l}\mu_l^{e_{l+1}} \ , & B_{4,l} &= y^{b_l}\nu_l^{e_{l+1}} \ , \\
B_{5,l} &= y^{h_l}g^{f_l} \ , & B_{6,l} &= y^{f_l} \ , \\
B_{7,l} &= y^{b_l''}g^{e_{l+1}} \ , & B_{8,l} &= (u_{l+1}')^{k_l} \ , \\
B_{9,l} &= y^{w_l'}(v_{l+1}')^{k_l} \ , & B_{10,l} &= g^{w_l}
\end{aligned}$$

for $l = 0, \ldots, \delta - 2$. Then it hands

$$\big(B_0, (B_{1,l}, B_{2,l})_{l=0}^{\delta-1}, \\ (A_{1,l}, A_{2,l}, A_{3,l}, B_{3,l}, B_{4,l}, B_{5,l}, B_{6,l}, B_{7,l}, B_{8,l}, B_{9,l}, B_{10,l})_{l=0}^{\delta-2}\big)$$

to the verifier.

3. The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands c to the prover.
4. The prover computes

$$d_{1,l} = cr_l + e_l \ , \quad d_{2,l} = -cs_l + i_l \ , \text{ and } \quad d_{3,l} = -ct_l + j_l \tag{16.1}$$

for $l = 0, \ldots, \delta - 1$ and

$$d_{4,l} = ca_l + b_l \ , \qquad d_{5,l} = ca'_l + b'_l \ , \tag{16.2}$$
$$d_{6,l} = c\,(a_l + s_l r_{l+1}) + f_l \ , \qquad d_{7,l} = ct_l r_{l+1} + h_l \ , \tag{16.3}$$
$$d_{8,l} = ca''_l + b''_l \ , \qquad d_{9,l} = ca''_l + w'_l \ , \tag{16.4}$$
$$d_{10,l} = c(r_{l+1}/r'_l) + k_l \ , \qquad d_{11,l} = cr'_l + w_l \tag{16.5}$$

for $l = 0, \ldots, \delta - 2$. Then it hands

$$((d_{1,l}, d_{2,l}, d_{3,l})_{l=0}^{\delta-1}, (d_{4,l}, d_{5,l}, d_{6,l}, d_{7,l}, d_{8,l}, d_{9,l}, d_{10,l}, d_{11,l})_{l=0}^{\delta-2})$$

to the verifier.

5. The verifier checks that

$$u_0^c B_0 = y_0^{d_{1,0}} \tag{16.6}$$

and

$$(v_l/\nu_l)^c B_{1,l} = g^{d_{1,l}} y^{d_{2,l}} \text{ and } \qquad B_{2,l} = \mu_l^c y^{d_{3,l}} g^{d_{2,l}} \ , \tag{16.7}$$
$$\tag{16.8}$$

for $l = 0, \ldots, \delta - 1$ and

$$A_{1,l}^c B_{3,l} = y^{d_{5,l}} g^{d_{4,l}} \mu_l^{d_{1,l+1}} \ , \qquad A_{2,l}^c B_{4,l} = y^{d_{4,l}} \nu_l^{d_{1,l+1}} \ , \tag{16.9}$$
$$A_{1,l}^c B_{5,l} = g^{d_{6,l}} y^{d_{7,l}} \ , \qquad (A_{2,l}/u_{l+1})^c B_{6,l} = y^{d_{6,l}} \ , \tag{16.10}$$
$$A_{3,l}^c B_{7,l} = y^{d_{8,l}} g^{d_{1,l+1}} \ , \qquad u_l^c B_{8,l} = (u'_{l+1})^{d_{10,l}} \ , \tag{16.11}$$
$$A_{3,l}^c B_{9,l} = y^{d_{9,l}} (v'_{l+1})^{d_{10,l}} \ , \qquad (v'_{l+1})^c B_{10,l} = g^{d_{11,l}} \tag{16.12}$$

for $l = 0, \ldots, \delta - 2$.

Intuitively the proof works by first showing that (u_l, v_l) encrypts the key y_{l+1} that is committed to in (μ_l, ν_l) and then by showing that key y_{l+1} in the commitment (μ_l, ν_l) is the encryption key used to produce (u_{l+1}, v_{l+1}). Based on these relations it is then proved that (u'_l, v'_l) is on the form $(y_l^{r'_l}, g^{r'_l})$.

This is depicted in Figure 16.1.

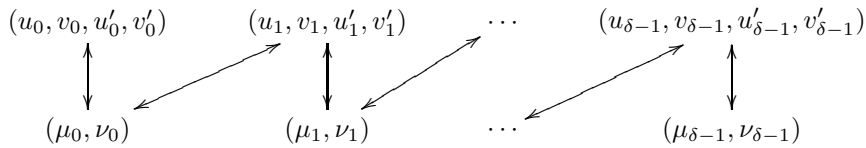

Figure 16.1: The protocol for a chain of cryptotexts. The elements $B_{1,l}$ and $B_{2,l}$ are used to prove the lth vertical relation. The elements $B_{3,l}, B_{4,l}, B_{5,l}, B_{6,l}$ are used to prove the lth diagonal relation. This explains why there are fewer element of the second type than the first. Finally, $B_{7,l}, B_{8,l}, B_{9,l}, B_{10,l}$ is used to prove that (u'_l, v'_l) is on the correct form.

Lemma 16.3. *Protocol 16.2 is a $\mathbb{Z}_{q_3}$-Σ-protocol.*

Proof. It is straightforward to see that the protocol has perfect completeness. We now prove special soundness. Suppose we have a list $(B_0, (B_{1,l}, B_{2,l})_{l=0}^{\delta-1}$, $(A_{1,l}, A_{2,l}, A_{2,l}, B_{3,l}, B_{4,l}, B_{5,l}, B_{6,l}, B_{7,l}, B_{8,l}, B_{9,l})_{l=0}^{\delta-2})$ and $((d_{1,l}, d_{2,l}, d_{3,l})_{l=0}^{\delta-1}$, $(d_{4,l}, d_{5,l}, d_{6,l}, d_{7,l}, d_{8,l}, d_{9,l}, d_{10,l})_{l=0}^{\delta-2})$ that satisfy the Equations (16.6)–(16.12), and $c' \neq c$ and $((d'_{1,l}, d'_{2,l}, d'_{3,l})_{l=0}^{\delta-1}, (d'_{4,l}, d'_{5,l}, d'_{6,l}, d'_{7,l}, d'_{8,l}, d'_{9,l}, d'_{10,l})_{l=0}^{\delta-2})$ that satisfies the same equations.

We solve the equation systems corresponding to Equations (16.1)–(16.5) to extract ρ_l, ζ_l, and τ_l for $l = 0, \ldots, \delta - 1$ such that

$$u_0 = y_0^{\rho_0} \ ,$$
$$v_l/\nu_l = g^{\rho_l} y^{-\zeta_l} \quad \text{and} \quad \mu_l = y^{\tau_l} g^{\zeta_l}$$

and α_l, α'_l, α''_l, λ_l, ω_l, $\omega_l^{\times}$, $\rho_l^{\times}$, and ρ_l^{+} for $l = 0, \ldots, \delta - 2$ such that

$$A_{1,l} = y^{\alpha'_l} g^{\alpha_l} \mu_l^{\rho_{l+1}} \ , \qquad A_{2,l} = y^{\alpha_l} \nu_l^{\rho_{l+1}} \ ,$$
$$A_{1,l} = y^{\lambda_l} g^{\omega_l} \ , \qquad A_{2,l}/u_{l+1} = y^{\omega_l} \ ,$$
$$A_{3,l} = y^{\alpha''_l} g^{\rho_{l+1}} \ , \qquad u_l = (u'_{l+1})^{\rho_l^{\times}} \ ,$$
$$A_{3,l} = y^{\omega_l^{\times}} (v'_{l+1})^{\rho_l^{\times}} \ , \qquad v'_{l+1} = g^{\rho_l^{+}}$$

From this we can compute $\zeta_l^* = (\omega_l - \alpha_l)/\rho_{l+1}$ and $\tau_l^* = (\lambda_l - \alpha'_l)/\rho_{l+1}$ for $l = 0, \ldots, \delta - 2$ such that $\mu_l = y^{\tau_l^*} g^{\zeta_l^*}$ since

$$y^{\tau_l^*} g^{\zeta_l^*} = \left(y^{\lambda_l - \alpha'_l} g^{(w_l - \alpha_l)} \right)^{1/\rho_{l+1}} = \left(\frac{A_{1,l}}{y^{\alpha'} g^{\alpha_l}} \right)^{1/\rho_{l+1}} = \mu_l \ .$$

We have $\nu_l = y^{\zeta_l^*} \gamma_{l+1}$ for some γ_{l+1}, i.e., $(\mu_l, \nu_l) = (y^{\tau_l^*} g^{\zeta_l^*}, y^{\zeta_l^*} \gamma_{l+1})$, for $l =$

$0, \ldots, \delta - 2$. This implies that

$$\begin{aligned} u_{l+1} &= A_{2,l} y^{-\omega_l} = y^{\alpha_l} \nu_l^{\rho_{l+1}} y^{-\omega_l} = y^{\alpha_l} y^{\zeta_l^* \rho_{l+1}} \gamma_{l+1}^{\rho_{l+1}} y^{-\omega_l} \\ &= y^{\alpha_l - \omega_l + \zeta_l^* \rho_{l+1}} \gamma_{l+1}^{\rho_{l+1}} = \gamma_{l+1}^{\rho_{l+1}} \ . \end{aligned}$$

Define γ_{l+1}^* by $v_l = g^{\rho_l} \gamma_{l+1}^*$, i.e., $(u_l, v_l) = E_{(\gamma_l, g)}(\gamma_{l+1}^*, \rho_l)$, for $l = 0, \ldots, \delta - 1$. What remains is to argue that $\zeta_l^* = \zeta_l$, $\tau_l^* = \tau_l$, and $\gamma_{l+1}^* = \gamma_{l+1}$ for $l = 0, \ldots, \delta - 2$ to connect the "links in the chain".

If one of the first two types of equalities does not hold, then we have $g^{\zeta_l} y^{\tau_l} = \mu_l = g^{\zeta_l^*} y^{\tau_l^*}$ and we can define $\eta = (\zeta_l - \zeta_l^*)/(\tau_l^* - \tau_l)$ such that $y = g^\eta$. In the main protocol this protocol is executed in G_{q_3}. Thus, if the equality does not hold Case 3 in Section 16.1 is satisfied. Thus, we assume that the first two types of equalities hold. Next we note that

$$g^{\rho_l} y^{-\zeta_l} = v_l / \nu_l = g^{\rho_l} \gamma_{l+1}^* y^{-\zeta_l^*} \gamma_{l+1}^{-1} = g^{\rho_l} y^{-\zeta_l} (\gamma_{l+1}^* / \gamma_{l+1}) \ ,$$

which implies that $\gamma_{l+1}^* = \gamma_{l+1}$. To summarize, we have found elements $\rho_0, \ldots, \rho_{\delta-1}$, $\tau_0, \ldots, \tau_{\delta-1}$, $\zeta_0, \ldots, \zeta_{\delta-1}$, and $\gamma_1, \ldots, \gamma_\delta$ such that

$$\begin{aligned} (u_0, v_0) &= (y_0^{\rho_0}, g^{\rho_0} \gamma_1) & (\mu_0, \nu_0) &= (y^{\tau_0} g^{\zeta_0}, y^{\zeta_0} \gamma_1) \\ (u_1, v_1) &= (\gamma_1^{\rho_1}, g^{\rho_1} \gamma_2) & (\mu_1, \nu_1) &= (y^{\tau_1} g^{\zeta_1}, y^{\zeta_1} \gamma_2) \\ &\vdots & &\vdots \\ (u_{\delta-1}, v_{\delta-1}) &= (\gamma_{\delta-1}^{\rho_{\delta-1}}, g^{\rho_{\delta-1}} \gamma_\delta) & (\mu_{\delta-1}, \nu_{\delta-1}) &= (y^{\tau_{\delta-1}} g^{\zeta_{\delta-1}}, y^{\zeta_{\delta-1}} \gamma_\delta) \ . \end{aligned}$$

Thus, we have

$$\begin{aligned} u'_{l+1} &= u_{l+1}^{1/\rho_l^\times} = \gamma_{l+1}^{\rho_{l+1}/\rho_l^\times} \\ v'_{l+1} &= (y^{\alpha_l''} g^{\rho_{l+1}} y^{-\omega_l^\times})^{1/\rho_l^\times} = y^{(\alpha_l'' - \omega_l^\times)/\rho_l^\times} g^{\rho_{l+1}/\rho_l^\times} \ . \end{aligned}$$

If $\alpha_l'' \neq \omega_l^\times$, then we define $\eta = (\rho_l + -\rho_{l+1}/\rho_l^\times)/((\alpha_l'' - \omega_l^\times)/\rho_l^+)$ and conclude that $y = g^\eta$ and Case 3 in Section 16.1 is satisfied, since in the main protocol this subprotocol is invoked in the group G_{q_3}.

To summarize we may define $\rho'_{l+1} = \rho_{l+1}/\rho_l^\times$ and have

$$\begin{aligned} (u'_1, v'_1) &= (\gamma_1^{\rho'_1}, g^{\rho'_1}) \\ (u'_2, v'_2) &= (\gamma_2^{\rho'_2}, g^{\rho'_2}) \\ &\vdots \\ (u'_{\delta-1}, v'_{\delta-1}) &= (\gamma_{\delta-1}^{\rho'_{\delta-1}}, g^{\rho'_{\delta-1}}) \ . \end{aligned}$$

We conclude that the protocol is special-sound.

The special zero-knowledge simulator is defined as follows. Given the challenge $c \in \mathbb{Z}_q$ it chooses $A_{1,l}, A_{2,l}, A_{3,l} \in G_q$, and

$$((d_{1,l}, d_{2,l}, d_{3,l})_{l=0}^{\delta-1}, (d_{4,l}, d_{5,l}, d_{6,l}, d_{7,l}, d_{8,l}, d_{9,l}, d_{10,l}, d_{11,l})_{l=0}^{\delta-2})$$

with $d_{i,l} \in \mathbb{Z}_q$ randomly and defines

$$(B_0, (B_{1,l}, B_{2,l})_{l=0}^{\delta-1}, (B_{3,l}, B_{4,l}, B_{5,l}, B_{6,l}, B_{7,l}, B_{8,l}, B_{9,l}, B_{10,l})_{l=0}^{\delta-2})$$

by Equations (16.6)–(16.12). It is easy to see that the resulting simulation is perfectly distributed. Thus, the protocol is special honest verifier perfect zero-knowledge. □

Next we consider the problem of proving that the values $y_\alpha \in G_{q_3}$ and $g_2^{y_\alpha} \in G_{q_2}$ committed to in two commitments $(\mu, \nu) = (y_3^t g_3^s,\ y_3^s y_\alpha)$ and $(\mu', \nu') = (y_2^{t'} g_2^{s'},\ y_2^{s'} h^{y_\alpha})$ respectively satisfy an exponential relation. Stadler [138] studied a simpler problem, namely, given a cryptotext $E_{(g,y)}(m)$ with $g, y \in G_{q_3}$ and g_2^m, prove that an exponential relation holds between the cleartext and the exponent. Although we consider a more complex problem, our protocol is based on his ideas. Note that proving that our relation holds is equivalent to proving knowledge of $s, t \in \mathbb{Z}_{q_2}$ and $s', t' \in \mathbb{Z}_{q_3}$ such that $(\theta, \omega, \phi) = ((\mu')^{\nu^{-1}}, (\nu')^{\nu^{-1}}, \mu^{-1})$ is on the form $(y_2^{t'} g_2^{s'}, y_2^{s'} h^{y_3^s}, y_3^t g_3^s)$. For clarity we state this observation as a protocol below.

As stated the two next protocols execute in the groups G_{q_3} and G_{q_2}, but we invoke the protocol also in the similarly related groups G_{q_2} and G_{q_1}. It is trivial to see that the security properties of the protocols are not changed by this.

Protocol 16.4 (Exponential Relation Between Committed Values).
COMMON INPUT: $g_3, y_3, \mu, \nu \in G_{q_3}$ and $g_2, y_2, h, \mu', \nu' \in G_{q_2}$.
PRIVATE INPUT: $t, s \in \mathbb{Z}_{q_3}$ such that $(\mu, \nu) = (y_3^t g_3^s, y_3^s y_\alpha)$ and $t', s' \in \mathbb{Z}_{q_2}$ such that $(\mu', \nu') = (y_2^{t'} g_2^{s'}, y_2^{s'} h^{y_\alpha})$.

1. Invoke Protocol 16.6 on common input $g_3, y_3, \phi \in G_{q_3}$ and $g_2, y_2, h, \theta, \omega \in G_{q_2}$, where $(\theta, \omega, \phi) = ((\mu')^{\nu^{-1}}, (\nu')^{\nu^{-1}}, \mu^{-1})$, and private input $-t, -s \in \mathbb{Z}_{q_3}$ and $t'\nu^{-1}, s'\nu^{-1} \in \mathbb{Z}_{q_2}$.

Lemma 16.5. *Protocol 16.4 is a* $\{0,1\}^{\kappa_c}$*-*Σ*-protocol.*

Proof. This follows directly from Lemma 16.7 below. □

We now give the double-decker exponentiation protocol called from within the protocol above.

Protocol 16.6 (Double-Decker Exponentiation).
COMMON INPUT: $g_3, y_3, \phi \in G_{q_3}$ and $g_2, y_2, h, \theta, \omega \in G_{q_2}$.
PRIVATE INPUT: $t, s \in \mathbb{Z}_{q_3}$ and $t', s' \in \mathbb{Z}_{q_2}$ with $(\theta, \omega, \phi) = (y_2^{t'} g_2^{s'}, y_2^{s'} h^{y_3^s}, y_3^t g_3^s)$.

1. The prover chooses $e_l, f_l \in \mathbb{Z}_{q_3}$ and $e'_l, f'_l \in \mathbb{Z}_{q_2}$ randomly for $l = 1, \ldots, \kappa_c$, computes $F_{1,l} = y_2^{e'_l} g_2^{f'_l}$, $F_{2,l} = y_2^{f'_l} h^{y_3^{f_l}}$, and $A_l = y_3^{e_l} g_3^{f_l}$. Then it hands $(F_{1,l}, F_{2,l}, A_l)_{l=1}^{\kappa_c}$ to the verifier.

2. The verifier chooses $b = (b_1, \ldots, b_{\kappa_c}) \in \{0,1\}^{\kappa_c}$ randomly and hands b to the prover.

3. The prover computes $d_{1,l} = e_l - b_l t$, $d_{2,l} = f_l - b_l s$, $d_{3,l} = f'_l - b_l y_3^{d_{2,l}} s'$, and $d_{4,l} = e'_l - b_l y_3^{d_{2,l}} t'$, and hands $(d_{1,l}, d_{2,l}, d_{3,l}, d_{4,l})_{l=1}^{\kappa_c}$ to the verifier.

4. The verifier checks for $l = 1, \ldots, \kappa_c$ that

$$\theta^{b_l} y_3^{d_{2,l}} y_2^{d_{4,l}} g_2^{d_{3,l}} = F_{1,l} \ , \qquad y_2^{d_{3,l}} (\omega^{b_l} h^{(1-b_l)})^{y_3^{d_{2,l}}} = F_{2,l} \ , \text{ and} \tag{16.13}$$

$$\phi^{b_l} y_3^{d_{1,l}} g_3^{d_{2,l}} = A_l \ . \tag{16.14}$$

Lemma 16.7. *Protocol 16.6 is a $\{0,1\}^{\kappa_c}$-Σ-protocol.*

Proof. It is easy to see that the protocol has perfect completeness. Consider now special soundness. Suppose that we are given the outputs from two executions $(F_{1,l}, F_{2,l}, A_l)_{l=1}^{\kappa_c}$, b, $(d_{1,l}, d_{2,l})_{l=1}^{\kappa_c}$ and b', $(d'_{1,l}, d'_{2,l})_{l=1}^{\kappa_c}$ with $b \neq b'$ that satisfy Equations (16.13)–(16.14). Thus, for some l we have $b_l \neq b'_l$.

Let (ε, τ) and $(\psi, \zeta) \in \mathbb{Z}_{q_3}$ be solutions to the equation systems

$$\left\{ \begin{array}{l} d_{1,l} = e_l - b_l t \\ d'_{1,l} = e_l - b'_l t \end{array} \right\} \quad \text{and} \quad \left\{ \begin{array}{l} d_{2,l} = f_l - b_l s \\ d'_{2,l} = f_l - b'_l s \end{array} \right\} \ ,$$

This implies that $\phi = y_3^{\tau} g_3^{\zeta}$. Consider next the equation system

$$\left\{ \begin{array}{l} d_{3,l} = f'_l - b_l y_3^{d_{2,l}} s' \\ d'_{3,l} = f'_l - b'_l y_3^{d'_{2,l}} s' \end{array} \right\} \ .$$

Note that $b_l y_3^{d_{2,l}}$ is zero if $b_l = 0$ and non-zero otherwise. Thus, the system is solvable. Let (ψ', ζ') be a solution and assume without loss that $b'_l = 0$. Then we have

$$F_{2,l} = y_2^{d_{3,l}} \omega^{y_3^{d_{2,l}}} = y_2^{\psi' - y_3^{d_{2,l}} \zeta'} \omega^{y_3^{d_{2,l}}} = y_2^{\psi' - y_3^{\psi - \zeta} \zeta'} \omega^{y_3^{\psi - \zeta}} \quad \text{and}$$

$$F_{2,l} = y_2^{d'_{3,l}} h^{y_3^{d'_{2,l}}} = y_2^{\psi'} h^{y_3^{d'_{2,l}}} = y_2^{\psi'} h^{y_3^{\psi}} \ .$$

Solving for ω gives $\omega = y_2^{\zeta'} h^{y_3^{\zeta}}$. Finally, let (ε', τ') be the solution to

$$\left\{ \begin{array}{l} d_{4,l} = e'_l - b_l y_3^{d_{2,l}} t' \\ d'_{4,l} = e'_l - b'_l y_3^{d'_{2,l}} t' \end{array} \right\} \ .$$

Then we have

$$F_{1,l} = \theta^{y_3^{d_{2,l}}} y_2^{d_{4,l}} g_2^{d_{3,l}} = \theta^{y_3^{d_{2,l}}} y_2^{\varepsilon' - y_3^{d_{2,l}}\tau'} g_2^{\psi' - y_3^{d_{2,l}}\zeta'} \quad \text{and}$$
$$F_{1,l} = y_2^{d'_{4,l}} g_2^{d'_{3,l}} = y_2^{\varepsilon'} g_2^{\psi'} \ .$$

Solving for θ gives $\theta = y_2^{\tau'} g_2^{\zeta'}$. We conclude that the protocol is special-sound.

The special zero-knowledge simulator is defined as follows. Given $b \in \{0,1\}^{\kappa_c}$ it chooses $d_{1,l}, d_{2,l} \in \mathbb{Z}_{q_3}$ and $d_{3,l}, d_{4,l} \in \mathbb{Z}_{q_2}$ randomly for $l = 1, \ldots, \kappa_c$ and defines $(F_{1,l}, F_{1,l}, A_l)$ by Equations (16.13)–(16.14). We conclude that the protocol is special honest verifier perfect zero-knowledge. □

Our next protocol shows that the cleartext of an El Gamal encryption is the value hidden in a commitment. Since the protocol is used in conjunction with Cramer-Shoup cryptotexts, we use a notation that is consistent with the notation we use for the Cramer-Shoup cryptosystem in the main protocol.

Protocol 16.8 (Equality of Committed and Encrypted Cleartexts).
COMMON INPUT: $g_3, y_3, \mu, \nu, \bar{g}_1, \bar{h}, \bar{u}, \bar{v} \in G_{q_3}$.
PRIVATE INPUT: t, s, r such that $(\mu, \nu) = (y_3^t g_3^s, y_3^s m)$ and $(\bar{u}, \bar{v}) = (\bar{g}_1^r, \bar{h}^r m)$.

1. The prover chooses $a, e, f \in \mathbb{Z}_{q_3}$ randomly, computes $A_1 = y_3^a g_3^e$, $A_2 = y_3^e \bar{h}^f$, $A_3 = \bar{g}_1^f$, and hands (A_1, A_2, A_3) to the verifier.
2. The verifier chooses $c \in \mathbb{Z}_{q_3}$ randomly and hands it to the verifier.
3. The prover computes $d_1 = ct + a$, $d_2 = cs + e$, $d_3 = -cr + f$ and hands (d_1, d_2, d_3) to the verifier.
4. The verifier checks that

$$\mu^c A_1 = y_3^{d_1} g_3^{d_2}, \quad (\nu/\bar{v})^c A_2 = y_3^{d_2} \bar{h}^{d_3}, \quad A_3/\bar{u}^c = \bar{g}_1^{d_3} \ . \tag{16.15}$$

Lemma 16.9. *Protocol 16.8 is a $\mathbb{Z}_{q_3}$-Σ-protocol.*

Proof. It is straightforward to see that the protocol has perfect completeness. Consider special soundness. Given (A_1, A_2, A_3), (c, d_1, d_2, d_3), and (c', d'_1, d'_2, d'_3), with $c \neq c'$, that satisfy Equation (16.15) above, we can solve the corresponding equation systems to find $\tau, \zeta, \rho \in \mathbb{Z}_{q_3}$ such that

$$(\mu, \nu/\bar{v}, \bar{u}) \quad = \quad (y_3^{\tau} g_3^{\zeta}, y_3^{\zeta} \bar{h}^{\rho}, \bar{g}_1^{-\rho}) \ .$$

This implies that the cryptotext and commitment holds the same value $\bar{v}/\bar{h}^{\tau}$ as prescribed. Thus, the protocol is special-sound.

Given the challenge c the special zero-knowledge simulator chooses $d_1, d_2, d_3 \in \mathbb{Z}_{q_3}$ randomly and defines A_1, A_2, A_3 by Equation (16.15). It is easy to see that the resulting distribution is distributed exactly like that in a real execution. Thus, the protocol is special honest verifier perfect zero-knowledge. □

Our next protocol shows that a Cramer-Shoup cryptotext is valid. Here H denotes the representation of a collision-free hash function.

Protocol 16.10 (Validity of Cramer-Shoup Cryptotext).
COMMON INPUT: $H : G_{q_3}^3 \to \mathbb{Z}_{q_3}$, $\bar{g}_1, \bar{g}_2, \bar{c}, \bar{d} \in G_{q_3}$, and $\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu} \in G_{q_3}$.
PRIVATE INPUT: $r \in \mathbb{Z}_{q_3}$ such that $(\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu}) = (\bar{g}_1^r, \bar{g}_2^r, \bar{v}, \bar{c}^r \bar{d}^{rH(\bar{u},\bar{\mu},\bar{v})})$.

1. The prover chooses $a \in \mathbb{Z}_{q_3}$ randomly and computes $B_1 = \bar{g}_1^a$, $B_2 = \bar{g}_2^a$, $B_3 = (\bar{c}\bar{d}^{H(\bar{u},\bar{\mu},\bar{v})})^a$ and hands (B_1, B_2, B_3) to the verifier.
2. The verifier chooses $c \in \mathbb{Z}_{q_3}$ randomly and hands c to the prover.
3. The prover computes $d = cr + a$ and hands d to the verifier.
4. The verifier checks that $\bar{u}^c B_1 = \bar{g}_1^d$, $\bar{\mu}^c B_2 = \bar{g}_2^d$ and $\bar{\nu}^c B_3 = (\bar{c}\bar{d}^{H(\bar{u},\bar{\mu},\bar{v})})^d$.

Lemma 16.11. *Protocol 16.10 is a $\mathbb{Z}_{q_3}$-Σ-protocol.*

Proof. It is straightforward to see that the protocol has perfect completeness. Assuming the output of two executions B_1, B_2, B_3, c, d and B_1, B_2, B_3, c', d' for $c \neq c'$ both satisfying the verification of Step 4, we can compute $\rho = (d - d')/(c - c')$ such that $(\bar{u}, \bar{\mu}, \bar{\nu}) = (\bar{g}_1^\rho, \bar{g}_2^\rho, \bar{c}^\rho \bar{d}^{\rho H(\bar{u},\bar{\mu},\bar{v})})$. Thus, the protocol is special-sound.

Given the challenge c the special zero-knowledge simulator chooses $d \in \mathbb{Z}_{q_3}$ randomly and defines B_1, B_2, and B_3 by the equations in Step 4. It follows that the protocol is special honest verifier perfect zero-knowledge. □

The next protocol combines the protocols above and provides a solution to the goal of this section, i.e., proving the relations in Step 3 in Algorithm 15.3 involving only elements from G_{q_1}, G_{q_2}, and G_{q_3}.

Protocol 16.12 (Commitment to Hash of Chained Keys).
COMMON INPUT: $g_3, y_3, y_{\alpha_0} \in G_{q_3}$, $g_2, y_2 \in G_{q_2}$, $g_1, y_1 \in G_{q_1}$, $H^{\mathsf{CHP}} = (h_1, \ldots, h_\delta) \in G_{q_2}^\delta$, $(u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1} \in G_{q_3}^{2\delta}$, $(\mu'', \nu'') \in G_{q_1}^2$, $\bar{g}_1, \bar{g}_2, \bar{c}, \bar{d}, \bar{h} \in G_{q_3}$,
$C_\delta = (\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu}) \in G_{q_3}^4$.
PRIVATE INPUT: $r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta \in \mathbb{Z}_{q_3}$, $y_{\alpha_1}, \ldots, y_{\alpha_\delta} \in G_{q_3}$, and $s'', t'' \in \mathbb{Z}_{q_2}$ such that

$$
\begin{aligned}
(u_l, v_l) &= E_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
(u'_l, v'_l) &= E_{(y_{\alpha_l}, g_3)}(1, r'_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
C_\delta &= E_Y^{\mathsf{cs}}(y_{\alpha_\delta}, r_\delta) \ , \quad \text{and} \\
(\mu'', \nu'') &= (y_1^{t''} g_1^{s''}, y_1^{s''} g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})}) \ .
\end{aligned}
$$

1. The prover chooses $s_l, t_l \in \mathbb{Z}_{q_2}$ randomly, computes commitments

$$(\mu_l, \nu_l) = \left(y_3^{t_l} g_3^{s_l}, y_3^{s_l} y_{\alpha_{l+1}}\right)$$

for $l = 0, \ldots, \delta - 1$, and hands $(\mu_l, \nu_l)_{l=0}^{\delta-1}$ to the verifier.

2. The prover chooses $s'_l, t'_l \in \mathbb{Z}_{q_3}$ randomly, computes commitments $(\mu'_l, \nu'_l) = (y_2^{t'_l} g_2^{s'_l}, y_2^{s'_l} h_{l+1}^{y_{\alpha_{l+1}}})$ for $l = 0, \ldots, \delta - 1$, and hands $(\mu'_l, \nu'_l)_{l=1}^{\delta}$ to the verifier.
3. The prover and verifier computes $(\mu', \nu') = \left(\prod_{l=0}^{\delta-1} \mu'_l, \prod_{l=0}^{\delta-1} \nu'_l\right)$. The prover computes $s' = \sum_{l=0}^{\delta-1} s'_l$ and $t' = \sum_{l=0}^{\delta-1} t'_l$.
4. Invoke the following protocols in parallel:
   a) Protocol 16.2 on public input y_{α_0}, g_3, y_3, $\left((u_l, v_l, u'_l, v'_l), (\mu_l, \nu_l)\right)_{l=0}^{\delta-1}$, and private input $(r_l, r'_l, s_l, t_l)_{l=0}^{\delta-1}$ to show that the chain is a valid chain of encrypted keys and commitments.
   b) Protocol 16.4 for $l = 0, \ldots, \delta - 1$ on public input $g_3, y_3, \mu_l, \nu_l \in G_{q_3}$ and $g_2, y_2, h_l, \mu'_l, \nu'_l \in G_{q_2}$, and private input $s_l, t_l \in \mathbb{Z}_{q_3}$ and $s'_l, t'_l \in \mathbb{Z}_{q_2}$. This "lifts" each committed public key up into the exponent.
   c) Protocol 16.4 on public input $g_2, y_2, \mu', \nu' \in G_{q_2}$ and $g_1, y_1, g_1, \mu'', \nu'' \in G_{q_1}$, and private input $s', t' \in \mathbb{Z}_{q_2}$ and $s'', t'' \in \mathbb{Z}_{q_1}$. This "lifts" the Chaum-van Heijst-Pfitzmann hash value of the public keys along the chain up into the exponent.
   d) Protocol 16.8 on common input $g_3, y_3, \mu_{\delta-1}, \nu_{\delta-1} \in G_{q_3}$ and $\bar{g}_1, \bar{h}, \bar{u}, \bar{v} \in G_{q_3}$, and private input $t_{\delta-1}, s_{\delta-1}, r_\delta \in \mathbb{Z}_{q_3}$ to show that C_δ is an encryption of the value y_{α_δ} committed to in $(\mu_{\delta-1}, \nu_{\delta-1})$.
   e) Protocol 16.10 on common input H, and $\bar{g}_1, \bar{g}_2, \bar{c}, \bar{d}, \bar{h} \in G_{q_3}$, $C_\delta = (\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu}) \in G_{q_3}^4$, and private input $r_\delta \in \mathbb{Z}_{q_3}$ to show that C_δ is correctly formed.

Lemma 16.13. *Protocol 16.12 is a* $\{0,1\}^{\kappa_c} \times \mathbb{Z}_{q_3}$*-*$\Sigma$*-protocol.*

Proof. The perfect completeness of the protocol follows from the perfect completeness of the subprotocols.

From the Observations 2.38 and 2.39 it follows that Step 4 may be considered a single combined $\{0,1\}^{\kappa_c} \times \mathbb{Z}_{q_3}$-$\Sigma$-protocol. Given two satisfying transcripts the special soundness of each subprotocol can be used to find suitable values, but we must also show that the values found this way for the different subprotocols are consistent to prove special soundness.

Using Lemma 16.3 we can find $\tau_l, \tau'_l, \zeta_l, \psi_l \in \mathbb{Z}_{q_3}$, $\gamma_l \in G_{q_3}$ such that

$$\begin{aligned}(u_l, v_l) &= E_{(\gamma_l, g_3)}(\gamma_{l+1}, \tau_l) = (\gamma_l^{\tau_l}, g_3^{\tau_l}\gamma_{l+1}) \ , \\ (u'_l, v'_l) &= E_{(\gamma_l, g_3)}(1, \tau'_l) = (\gamma_l^{\tau'_l}, g_3^{\tau'_l}) \ , \text{ and} \\ (\mu_l, \nu_l) &= (y_3^{\psi_l} g_3^{\zeta_l}, y_3^{\zeta_l}\gamma_{l+1})\end{aligned}$$

for $l = 0, \ldots, \delta - 1$. Using Lemma 16.5 we can find $\tau^*_l, \zeta^*_l, \psi^*_l \in \mathbb{Z}_{q_3}$, $\gamma^*_l \in G_{q_3}$, and $\zeta'_l, \psi'_l \in \mathbb{Z}_{q_2}$ such that

$$(\mu_l, \nu_l) = (y_3^{\psi^*_l} g_3^{\zeta^*_l}, y_3^{\zeta^*_l}\gamma^*_{l+1}) \quad \text{and} \quad (\mu'_l, \nu'_l) = (y_2^{\psi'_l} g_2^{\zeta'_l}, y_2^{\zeta'_l} h_l^{\gamma^*_{l+1}})$$

for $l = 0, \ldots, \delta - 1$. If $\gamma^*_{l+1} \neq \gamma_{l+1}$, then either $\psi^*_l \neq \psi_l$ or $\zeta^*_l \neq \zeta_l$. Then we define $\eta = (\zeta_l - \zeta^*_l)/(\psi^*_l - \psi_l)$ and conclude that $y_3 = g_3^\eta$. In other words Case 3 in Section 16.1 is satisfied. Thus, we assume that $\gamma^*_\delta = \gamma_\delta$, $\psi^*_\delta = \psi_\delta$, and $\zeta^*_\delta = \zeta_\delta$.

Using Lemma 16.5 we can find $\zeta', \psi' \in \mathbb{Z}_{q_2}$, $\zeta'', \psi'' \in \mathbb{Z}_{q_1}$, and $\Gamma \in G_{q_2}$ such that

$$(\mu', \nu') = (y_2^{\psi'} g_2^{\zeta'}, y_2^{\zeta'} \Gamma) \quad \text{and} \quad (\mu'', \nu'') = (y_1^{\psi''} g_1^{\zeta''}, y_1^{\zeta''} g_1^{\Gamma}) \ .$$

If $\prod_{l=1}^{\delta} h_l^{\gamma_l} \neq \Gamma$, then either $\psi' \neq \sum_{l=0}^{\delta-1} \psi'_l$ or $\zeta' \neq \sum_{l=0}^{\delta-1} \zeta'_l$. Then we define $\eta = \psi' - \sum_{l=0}^{\delta-1} \psi'_l$ and $\sum_{l=0}^{\delta-1} \zeta'_l - \zeta'$ and conclude that $y_2 = g_2^\eta$. In other words Case 2 in Section 16.1 is satisfied. Thus, we assume that $\prod_{l=1}^{\delta} h_l^{\gamma_l} = \Gamma$, $\psi' = \sum_{l=0}^{\delta-1} \psi'_l$, and $\zeta' = \sum_{l=0}^{\delta-1} \zeta'_l$.

Using Lemma 16.9 we can find $\psi^\times_{\delta-1}, \zeta^\times_{\delta-1}, \tau \in \mathbb{Z}_{q_3}$ and $\gamma^\times_\delta \in G_{q_3}$ such that

$$(\mu_{\delta-1}, \nu_{\delta-1}) = (y_3^{\psi^\times_{\delta-1}} g_3^{\zeta^\times_{\delta-1}}, y_3^{\zeta^\times_{\delta-1}} \gamma^\times_\delta) \quad \text{and} \quad (\bar{u}, \bar{v}) = (\bar{g}_1^\tau, \bar{h}^\tau \gamma^\times_\delta) \ .$$

If $\gamma^\times_\delta \neq \gamma_\delta$, then either $\psi^\times_{\delta-1} \neq \psi_{\delta-1}$ or $\zeta^\times_{\delta-1} \neq \zeta_{\delta-1}$. Then we define $\eta = (\psi^\times_{\delta-1} - \psi_{\delta-1})/(\zeta_{\delta-1} - \zeta^\times_{\delta-1})$ and conclude that $y_3 = g_3^\eta$. In other words Case 3 in Section 16.1 is satisfied. Thus, we assume that $\gamma^\times_\delta = \gamma_\delta$, $\psi^\times_{\delta-1} = \psi_{\delta-1}$ and $\zeta^\times_{\delta-1} = \zeta_{\delta-1}$.

Using Lemma 16.11 we can find $\tau \in \mathbb{Z}_{q_3}$ such that $(\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu}) = E^{\mathsf{cs}}_Y(\gamma_\delta, \tau)$, where Y is the public key $Y = (H, \bar{g}_1, \bar{g}_2, \bar{c}, \bar{d}, \bar{h})$ to the Cramer-Shoup cryptosystem over G_{q_3}. This concludes the proof of special soundness of the protocol.

Given a challenge $(b, c) \in \{0,1\}^{\kappa_c} \times \mathbb{Z}_{q_3}$ the special zero-knowledge simulator chooses $\mu_l, \nu_l \in G_{q_3}$ and $\mu'_l, \nu'_l \in G_{q_2}$ randomly and invokes the special zero-knowledge simulator of each invoked subprotocol. Since the commitments (μ_l, ν_l) and (μ'_l, ν'_l) are perfectly distributed and each subprotocol is special honest verifier perfect zero-knowledge, then so is the combined protocol. □

16.3 Protocols in Two Distinct Groups

In this section we consider the problem of proving equality of exponents over distinct groups. This is used as a bridge between the two parts of the main protocol. Two Pedersen commitments are given: one over G_n denoted $C = y^{s'} g^e$, with $e, s' \in \mathbb{Z}_n$ and one over $\mathrm{QR}_{\mathbf{N}}$ denoted $\mathbf{C} = \mathbf{y}^s \mathbf{g}^e$ with $s \in [0, 2^{\kappa+\kappa_r} - 1]$. In our application G_n is a group G_q of prime order q or a group $G_{\mathbf{N}}$ with order equal to the RSA modulus $\mathbf{N}$.

This problem has been studied by Boudot and Traoré [30] as well as by Camenisch and Michels [36]. We use Boudot's protocol [29] for proving that a committed value is contained in a certain interval. Instead of giving the complete protocol, we only give the interface and refer the reader to [29] for details.

Protocol Head 16.14 (A Committed Number Lies in an Interval).
Common Input: $\mathbf{g}, \mathbf{y}, \mathbf{C} \in \mathrm{QR}_{\mathbf{N}}$ and $a, b \in \mathbb{Z}$.
Private Input: $e \in [a, b]$ and $s \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ such that $\mathbf{C} = \mathbf{y}^s \mathbf{g}^e$.

Lemma 16.15. *Protocol 16.14 is a* $\{0,1\}^{\kappa_c}$*-*Σ*-protocol.*

Proof. Boudot [29] essentially shows that either $e \in [a,b]$ or Case 5 in Section 16.1 is satisfied. □

We now give the proof of equality of exponents over distinct groups using the protocol above.

Protocol 16.16 (Equality of Exponents Over Distinct Groups).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{C} \in \mathbb{Z}_{\mathbf{N}}^*$ and $g, y, C \in G_n$.
PRIVATE INPUT: $e \in [0, n-1]$, $s \in [0, 2^{\kappa_r}\mathbf{N} - 1]$, and $s' \in \mathbb{Z}_n$. such that $\mathbf{C} = \mathbf{y}^s\mathbf{g}^e$ and $C = y^{s'}g^e$.

1. The prover chooses $a \in [0, 2^{\kappa_c+\kappa_r}n - 1]$, $b \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$ and $b' \in \mathbb{Z}_n$ randomly, computes
$$\mathbf{A} = \mathbf{y}^b\mathbf{g}^a \quad \text{and} \quad A = y^{b'}g^a$$
and hands $(\mathbf{A}, A)$ to the verifier.
2. Protocol 16.14 is executed in parallel with the protocol below on common input $\mathbf{g}, \mathbf{y}, \mathbf{C}$ and using the interval $[0, n-1]$ and private input e and s.
3. The verifier chooses $c \in [0, 2^{\kappa_c} - 1]$ and hands it to the prover.
4. The prover computes
$$d_1 = ce + a \bmod 2^{\kappa_c+\kappa_r}n \ , \tag{16.16}$$
$$d_2 = cs + b \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \text{ and} \tag{16.17}$$
$$d_3 = cs' + b' \bmod n \ , \tag{16.18}$$
and hands (d_1, d_2, d_3) to the verifier.
5. The verifier checks that $\mathbf{y}^{d_2}\mathbf{g}^{d_1} = \mathbf{C}^c\mathbf{A}$ and $y^{d_3}g^{d_1} = C^cA$.

Lemma 16.17. *Protocol 16.16 with* $n = q$ *or* $n = \mathbf{N}$ *is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. If the prover is honest the verifier accepts if there is no modular reduction in the computation of d_1, d_2. By the union bound this happens with probability not more than $2 \cdot 2^{-\kappa_r}$, which is negligible. Thus, the protocol has overwhelming completeness.

To prove that the protocol is special-sound, assume we have $\mathbf{A}, A, c, d_1, d_2, d_3$ as well as $c' \neq c$, d_1', d_2', d_3', each list satisfying the equations of Step 5. Then we have
$$\mathbf{y}^{d_2-d_2'}\mathbf{g}^{d_1-d_1'} = \mathbf{C}^{c-c'} \quad \text{and} \quad y^{d_3-d_3'}g^{d_1-d_1'} = C^{c-c'} \ .$$
If $c - c'$ does not divide both $d_1 - d_1'$ and $d_2 - d_2'$ we define $\eta_0 = c - c'$, $\eta_1 = d_1 - d_1'$, $\eta_2 = d_2 - d_2'$, and $\mathbf{b} = \mathbf{C}$ and conclude that Case 5 in Section 16.1 is satisfied.

If $n = q$, then $c - c'$ is obviously invertible in $\mathbb{Z}_n$. If $n = \mathbf{N}$ and $c - c'$ is not invertible, we know that $\gcd(c - c', \mathbf{N})$ is a non-trivial factor of $\mathbf{N}$, and Case 7 in Section 16.1 is satisfied.

Thus, we assume that $c - c'$ divides both $d_1 - d_1'$ and $d_2 - d_2'$ and define $\varepsilon = (d_1 - d_1')/(c - c')$ and $\zeta = (d_2 - d_2')/(c - c')$ over the integers and $\zeta' = (d_3 - d_3')/(c - c')$ over $\mathbb{Z}_n$. This gives

$$\mathbf{C} = \mathbf{y}^{\zeta}\mathbf{g}^{\varepsilon} \quad \text{and} \quad C = y^{\zeta'}g^{\varepsilon} \ .$$

Finally, using Lemma 16.15 we can find $\varepsilon_* \in [0, n-1]$ and ζ_* such that

$$\mathbf{C} = \mathbf{y}^{\zeta_*}\mathbf{g}^{\varepsilon_*} \ .$$

We may assume that $\varepsilon_* = \varepsilon$, since otherwise we can define $\eta_0 = \varepsilon - \varepsilon_*$, $\eta_1 = \zeta - \zeta_*$, and $\mathbf{b} = \mathbf{C}$ and conclude that Case 6 in Section 16.1 is satisfied.

Given the challenge $c \in [0, 2^{\kappa_c} - 1]$ the special zero-knowledge simulator chooses $d_1 \in [0, 2^{\kappa_c + \kappa_r} n - 1]$, $d_2 \in [0, 2^{\kappa_c + 2\kappa_r}\mathbf{N} - 1]$, and $d_3 \in \mathbb{Z}_n$ randomly and defines A and $\mathbf{A}$ by the equations in Step 5. This gives the same distribution as an execution of the protocol. Thus, the protocol is special honest verifier perfect zero-knowledge. □

16.4 Protocols in the Squares Modulo An RSA-modulus

Zero-knowledge proofs of knowledge of logarithms of elements in $\mathrm{QR}_{\mathbf{N}}$ have been studied by Fujisaki and Okamoto [67] and Damgård and Fujisaki [55]. We use similar techniques. More precisely we the consider Pedersen commitments $\mathbf{y}^s\mathbf{g}^e$ over $\mathrm{QR}_{\mathbf{N}}$ and the problem of proving relations between the committed values in such commitments.

Protocol 16.18 (Knowledge of Committed Value).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{QR}_{\mathbf{N}}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_{\mathbf{N}}^*$.
PRIVATE INPUT: $s, t \in [0, 2^{\kappa_r}\mathbf{N} - 1]$, $\mathbf{r} \in \mathrm{QR}_{\mathbf{N}}$ such that $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^t, \mathbf{y}^t\mathbf{r})$.

1. The prover chooses $a, b \in [0, 2^{\kappa_c + 2\kappa_r}\mathbf{N} - 1]$ randomly, computes $\boldsymbol{\mu} = \mathbf{y}^a\mathbf{g}^b$, and hands $\boldsymbol{\mu}$ to the verifier.
2. The verifier chooses $c \in [0, 2^{\kappa_c} - 1]$ randomly and hands it to the prover.
3. The prover computes

   $$d_1 = cs + a \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N} \quad \text{and} \quad d_2 = ct + b \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N}$$

   and hands (d_1, d_2) to the verifier.
4. The verifier checks that $\mathbf{u}^c\boldsymbol{\mu} = \mathbf{y}^{d_1}\mathbf{g}^{d_2}$.

Lemma 16.19. *Protocol 16.18 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. It is easy to check that the verifier accepts when there is no modular reduction in the computation of d_1 or d_2. Such a reduction occurs with probability at most $2 \cdot 2^{\kappa_r}$, which is negligible. Thus, the protocol has overwhelming completeness.

For the extraction of s, t and $\mathbf{r}$ to prove special soundness, assume that we have two lists $(\boldsymbol{\mu}, c, d_1, d_2)$ and $(\boldsymbol{\mu}, c', d_1', d_2')$, where $c \neq c'$, that satisfy the equations in Step 4. Thus, we have

$$\mathbf{u}^{c-c'} = \mathbf{y}^{d_1-d_1'}\mathbf{g}^{d_2-d_2'} \ .$$

If $(c-c')$ does not divide (d_1-d_1') and (d_2-d_2') we define $\eta_0 = c-c'$, $\eta_1 = d_1-d_1'$, $\eta_2 = d_2-d_2'$, and $\mathbf{b} = \mathbf{u}$ and conclude that Case 5 in Section 16.1 is satisfied.

Thus, we assume that $(c-c')$ divides (d_1-d_1') and (d_2-d_2') and define $\zeta = (d_1-d_1')/(c-c')$ and $\tau = (d_2-d_2')/(c-c')$. This gives

$$\mathbf{u} = \mathbf{y}^{\zeta}\mathbf{g}^{\tau} \ .$$

On input a challenge $c \in [0, 2^{\kappa_c}-1]$ the special zero-knowledge simulator chooses $d_1, d_2 \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N}-1]$ randomly and defines $\boldsymbol{\mu}$ by the equation in Step 4. The resulting transcript is identically distributed to that in the real protocol and we conclude that the protocol is special honest verifier perfect zero-knowledge. □

Next we give a protocol that shows that two committed values are equal. Note that we parameterize the protocol on a positive integer z to allow for different sizes of the exponents.

Protocol 16.20 (Equality of Committed Values).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \in \mathrm{QR}_{\mathbf{N}}$ and $\mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}' \in \mathbb{Z}_{\mathbf{N}}^*$.
PRIVATE INPUT: $s, t, s', t' \in [-2^{\kappa_r}z+1, 2^{\kappa_r}z-1]$ and $\mathbf{r} \in \mathrm{QR}_{\mathbf{N}}$ such that $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^t, \mathbf{y}^t\mathbf{r})$ and $(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{s'}\mathbf{g}^{t'}, \mathbf{y}^{t'}\mathbf{r})$.

1. The prover chooses $a, b, a', b' \in [0, 2^{\kappa_c+2\kappa_r}z-1]$ randomly, computes

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = (\mathbf{y}^a\mathbf{g}^b, \mathbf{y}^b\mathbf{y}^{-b'}, \mathbf{y}^{a'}\mathbf{g}^{b'})$$

   and hands $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ to the verifier.

2. The verifier chooses $c \in [0, 2^{\kappa_c}-1]$ randomly and hands it to the prover.

3. The prover computes

$$\begin{aligned} d_1 &= cs + a \bmod 2^{\kappa_c+2\kappa_r}z \ , \\ d_2 &= ct + b \bmod 2^{\kappa_c+2\kappa_r}z \ , \\ d_3 &= cs' + a' \bmod 2^{\kappa_c+2\kappa_r}z \ , \quad \text{and} \\ d_4 &= ct' + b' \bmod 2^{\kappa_c+2\kappa_r}z \ , \end{aligned}$$

   and hands (d_1, d_2, d_3, d_4) to the verifier.

4. The verifier checks that

$$(\mathbf{u}^c\boldsymbol{\alpha}, (\mathbf{v}/\mathbf{v}')^c\boldsymbol{\beta}, (\mathbf{u}')^c\boldsymbol{\gamma}) = (\mathbf{y}^{d_1}\mathbf{g}^{d_2}, \mathbf{y}^{d_2}\mathbf{y}^{-d_4}, \mathbf{y}^{d_3}\mathbf{g}^{d_4}) \ .$$

Lemma 16.21. *Protocol 16.20 is a* $[0, 2^{\kappa_c} - 1]$-Σ-*protocol.*

Proof. An honest prover fails to convince the verifier if there is a modular reduction in the computation of d_1, d_2, d_3, and d_4. It is easy to see that this happens with negligible probability. Thus, the protocol has overwhelming completeness.

To show special soundness assume that we have $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$, c and (d_1, d_2, d_3, d_4) satisfying the equations of Step 4 as well as $c' \neq c$ and (d_1', d_2', d_3', d_4') satisfying the same equations. We have

$$(\mathbf{u}^{c-c'}, (\mathbf{v}/\mathbf{v}')^{c-c'}, (\mathbf{u}')^{c-c'}) = (\mathbf{y}^{d_1-d_1'}\mathbf{g}^{d_2-d_2'}, \mathbf{y}^{d_2-d_2'}\mathbf{y}^{-(d_4-d_4')}, \mathbf{y}^{d_3-d_3'}\mathbf{g}^{d_4-d_4'}) \ .$$

If $(c-c')$ does not divide (d_1-d_1') and (d_2-d_2') we define $\eta_0 = c-c'$, $\eta_1 = d_1-d_1'$, $\eta_2 = d_2-d_2'$, and $\mathbf{b} = \mathbf{u}$ and conclude that Case 5 in Section 16.1 is satisfied. We do correspondingly if $(c-c')$ does not divide (d_3-d_3') and (d_4-d_4').

Thus, we assume that $(c-c')$ divides (d_1-d_1'), (d_2-d_2'), (d_3-d_3'), and (d_4-d_4'), and define $\zeta = (d_1-d_1')/(c-c')$, $\tau = (d_2-d_2')/(c-c')$, $\zeta' = (d_3-d_3')/(c-c')$, and $\tau' = (d_4-d_4')/(c-c')$. This gives

$$(\mathbf{u}, \mathbf{v}/\mathbf{v}', \mathbf{u}') = (\mathbf{y}^{\zeta}\mathbf{g}^{\tau}, \mathbf{y}^{\tau}\mathbf{y}^{-\tau'}, \mathbf{y}^{\zeta'}\mathbf{g}^{\tau'}) \ .$$

On input a challenge $c \in [0, 2^{\kappa_c}-1]$ the special zero-knowledge simulator chooses $d_1, d_2, d_3, d_4 \in [0, 2^{\kappa_c+2\kappa_r}z - 1]$ randomly and defines $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ by the equations in Step 4. The resulting distribution is equal to the distribution of the transcript of an honest execution of the protocol. Thus, the protocol is special honest verifier perfect zero-knowledge. □

The above protocol can also be used to prove that a pair $\mathbf{u}, \mathbf{v}$ is a commitment to a public value $\mathbf{w}$. For clarity we state this as a protocol, also this time parameterized on z:

Protocol 16.22 (Specific Committed Value).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{QR}_{\mathbf{N}}$ and $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{Z}_{\mathbf{N}}^*$.
PRIVATE INPUT: $s, t \in [-2^{\kappa_r}z + 1, 2^{\kappa_r}z - 1]$ such that $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^t, \mathbf{y}^t\mathbf{w})$.

1. Invoke protocol 16.20 on common input $\mathbf{g}, \mathbf{y}, (\mathbf{u}, \mathbf{v}), (\mathbf{1}, \mathbf{w})$ and private exponents $s, t, 0, 0$.

Lemma 16.23. *Protocol 16.22 is a* $[0, 2^{\kappa_c} - 1]$-Σ-*protocol.*

Proof. This follows directly from Lemma 16.21. □

In Protocol 16.4 we showed how to prove that two committed values have an exponential relation. We need to be able to do this also over $\mathbb{Z}_{\mathbf{N}}$. We use a protocol for double-decker exponential relations similar to Protocol 16.6. Once again we use the fact that proving that (u, v) and $(\mathbf{u}, \mathbf{v})$ are on the forms $(u, v) = (y_{\mathbf{N}}^{t'} g_{\mathbf{N}}^{s'}, y_{\mathbf{N}}^{s'} g_{\mathbf{N}}^{\mathbf{r}})$ and $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^t \mathbf{g}^s, \mathbf{y}^s \mathbf{r})$ is equivalent to proving that $(\theta, \omega, \boldsymbol{\phi}) = (u^{\mathbf{v}^{-1}}, v^{\mathbf{v}^{-1}}, \mathbf{u}^{-1})$ is on the form $(y_{\mathbf{N}}^{t'} g_{\mathbf{N}}^{s'}, y_{\mathbf{N}}^{s'} g_{\mathbf{N}}^{\mathbf{y}^t}, \mathbf{y}^t \mathbf{g}^s)$.

Protocol 16.24 (Basic Double-Decker Exponentiation).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \boldsymbol{\phi} \in \mathrm{QR}_{\mathbf{N}}$ and $g_{\mathbf{N}}, y_{\mathbf{N}}, \theta, \omega \in G_{\mathbf{N}}$.
PRIVATE INPUT: $t, s \in [-2^{\kappa_r}\mathbf{N} + 1, 2^{\kappa_r}\mathbf{N} - 1]$ and $t', s' \in \mathbb{Z}_{\mathbf{N}}$ such that $(\theta, \omega, \boldsymbol{\phi}) = (y_{\mathbf{N}}^{t'} g_{\mathbf{N}}^{s'}, y_{\mathbf{N}}^{s'} g_{\mathbf{N}}^{\mathbf{y}^s}, \mathbf{y}^t \mathbf{g}^s)$.

1. The prover chooses $e_l, f_l \in [0, 2^{2\kappa_r}\mathbf{N} - 1]$ and $e'_l, f'_l \in \mathbb{Z}_{\mathbf{N}}$ randomly for $l = 1, \ldots, \kappa_c$. Then it computes

$$(F_{1,l}, F_{2,l}, \mathbf{A}_l) = (y_{\mathbf{N}}^{e'_l} g_{\mathbf{N}}^{f'_l}, y_{\mathbf{N}}^{f'_l} g_{\mathbf{N}}^{\mathbf{y}^{f_l}}, \mathbf{y}^{e_l} \mathbf{g}^{f_l})$$

   and hands $(F_{1,l}, F_{2,l}, \mathbf{A}_l)_{l=1}^{\kappa_c}$ to the verifier.

2. The verifier randomly chooses $b = (b_1, \ldots, b_{\kappa_c}) \in \{0,1\}^{\kappa_c}$ and hands b to the prover.

3. The prover computes

$$\begin{aligned} d_{1,l} &= e_l - b_l t \bmod 2^{2\kappa_r}\mathbf{N} \ , \\ d_{2,l} &= f_l - b_l s \bmod 2^{2\kappa_r}\mathbf{N} \ , \\ d_{3,l} &= f'_l - b_l \mathbf{y}^{d_{2,l}} s' \bmod \mathbf{N} \ , \quad \text{and} \\ d_{4,l} &= e'_l - b_l \mathbf{y}^{d_{2,l}} t' \bmod \mathbf{N} \ , \end{aligned}$$

   and hands $(d_{1,l}, d_{2,l}, d_{3,l}, d_{4,l})_{l=1}^{\kappa_c}$ to the verifier.

4. The verifier checks for $l = 1, \ldots, \kappa_c$ that

$$\begin{aligned} \theta^{b_l \mathbf{y}^{d_{2,l}}} y_{\mathbf{N}}^{d_{4,l}} g_{\mathbf{N}}^{d_{3,l}} &= F_{1,l} \ , \\ y_{\mathbf{N}}^{d_{3,l}} (\omega^{b_l} g_{\mathbf{N}}^{1-b_l})^{\mathbf{y}^{d_{2,l}}} &= F_{2,l} \ , \quad \text{and} \\ \boldsymbol{\phi}^{b_l} \mathbf{y}^{d_{1,l}} \mathbf{g}^{d_{2,l}} &= \mathbf{A}_l \ . \end{aligned}$$

Lemma 16.25. *Protocol 16.24 is a* $\{0,1\}^{\kappa_c}$*-*Σ*-protocol.*

Proof. If there is no reduction in the computations of $d_{1,l}$ and $d_{2,l}$ the verifier will accept if the prover is honest. It is easy to see that a reduction occurs with negligible probability. Thus, the protocol has overwhelming completeness.

Now we prove special soundness. For this we follow the proof of Lemma 16.5, taking into account that the order of $\mathbb{Z}_{\mathbf{N}}^*$ is unknown.

Suppose that we are given two outputs $(F_{1,l},\ F_{2,l},\ \mathbf{A}_l)_{l=1}^{\kappa_c}$, b, $(d_{1,l}, d_{2,l})_{l=1}^{\kappa_c}$ and b', $(d'_{1,l}, d'_{2,l})_{l=1}^{\kappa_c}$ with $b \neq b'$ that satisfy the equations of Step 4. Thus, for some l, $b_l \neq b'_l$.

Let (ε, τ) and (ψ, ζ) be solutions to the equation systems

$$\left\{ \begin{array}{l} d_{1,l} = e_l - b_l t \\ d'_{1,l} = e_l - b'_l t \end{array} \right\} \quad \text{and} \quad \left\{ \begin{array}{l} d_{2,l} = f_l - b_l s \\ d'_{2,l} = f_l - b'_l s \end{array} \right\} ,$$

i.e., $\tau = \frac{d_{1,l} - d'_{1,l}}{b_l - b'_l}$ and $\zeta = \frac{d_{2,l} - d'_{2,l}}{b_l - b'_l}$. Since $|b_l - b'_l| = 1$ this gives integral values of τ, ζ when the system is solved over $\mathbb{Z}$. We now have that $\boldsymbol{\phi} = \mathbf{y}^{\tau}\mathbf{g}^{\zeta}$.

Consider next the equation system

$$\left\{ \begin{array}{l} d_{3,l} = f'_l - b_l \mathbf{y}^{d_{2,l}} s' \\ d'_{3,l} = f'_l - b'_l \mathbf{y}^{d'_{2,l}} s' \end{array} \right\} .$$

Note that $b_l \mathbf{y}^{d_{2,l}}$ is zero if $b_l = 0$ and non-zero otherwise. Thus, the system is solvable. Let (ψ', ζ') be a solution and assume without loss that $b'_l = 0$. Then we have

$$\begin{aligned} F_{2,l} &= y_{\mathbf{N}}^{d_{3,l}} \omega^{\mathbf{y}^{d_{2,l}}} = y_{\mathbf{N}}^{\psi' - \mathbf{y}^{d_{2,l}}\zeta'} \omega^{\mathbf{y}^{d_{2,l}}} = y_{\mathbf{N}}^{\psi' - \mathbf{y}^{\psi - \zeta}\zeta'} \omega^{\mathbf{y}^{\psi - \zeta}} \quad \text{and} \\ F_{2,l} &= y_{\mathbf{N}}^{d'_{3,l}} g_{\mathbf{N}}^{\mathbf{y}^{d'_{2,l}}} = y_{\mathbf{N}}^{\psi'} g_{\mathbf{N}}^{\mathbf{y}^{d'_{2,l}}} = y_{\mathbf{N}}^{\psi'} g_{\mathbf{N}}^{\mathbf{y}^{\psi}} . \end{aligned}$$

Solving for ω gives $\omega = y_{\mathbf{N}}^{\zeta'} g_{\mathbf{N}}^{\mathbf{y}^{\zeta}}$. Finally, let (ε', τ') be the solution to

$$\left\{ \begin{array}{l} d_{4,l} = e'_l - b_l \mathbf{y}^{d_{2,l}} t' \\ d'_{4,l} = e'_l - b'_l \mathbf{y}^{d'_{2,l}} t' \end{array} \right\} .$$

Then we have

$$\begin{aligned} F_{1,l} &= \theta^{\mathbf{y}^{d_{2,l}}} y_{\mathbf{N}}^{d_{4,l}} g_{\mathbf{N}}^{d_{3,l}} = \theta^{\mathbf{y}^{d_{2,l}}} y_{\mathbf{N}}^{\varepsilon' - \mathbf{y}^{d_{2,l}}\tau'} g_{\mathbf{N}}^{\psi' - \mathbf{y}^{d_{2,l}}\zeta'} \quad \text{and} \\ F_{1,l} &= y_{\mathbf{N}}^{d'_{4,l}} g_{\mathbf{N}}^{d'_{3,l}} = y_{\mathbf{N}}^{\varepsilon'} g_{\mathbf{N}}^{\psi'} . \end{aligned}$$

Solving for θ gives $\theta = y_{\mathbf{N}}^{\tau'} g_{\mathbf{N}}^{\zeta'}$. We conclude that the protocol is special-sound.

On input $b \in \{0,1\}^{\kappa_c}$ the special zero-knowledge simulator chooses random elements $d_{1,l}, d_{2,l}, d_{3,l}, d_{4,l} \in [0, 2^{2\kappa_r}\mathbf{N} - 1]$ and defines $(F_{1,l}, F_{2,l}, \mathbf{A}_l)$ by the equation in Step 4. The resulting distribution is identical to that in a real execution protocol. Thus, the protocol is special honest verifier perfect zero-knowledge. □

Unfortunately, the protocol does not give us exactly what we need. Although we have moved the committed value into the exponent the commitment (u, v) is defined over $G_{\mathbf{N}}$. We need a corresponding commitment over $\mathrm{QR}_{\mathbf{N}}$. To achieve this we combine the above protocol with a protocol for proving equivalence of exponents over distinct groups. This is illustrated in Figure 16.2.

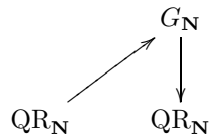

Figure 16.2: Double-decker exponentiation proof over an RSA-modulus.

Protocol 16.26 (Double-Decker Exponentiation).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{h} \in \mathrm{QR}_{\mathbf{N}}$, $(\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}') \in (\mathbb{Z}_{\mathbf{N}}^*)^2$, and $g_{\mathbf{N}}, y_{\mathbf{N}} \in G_{\mathbf{N}}$.
PRIVATE INPUT: $\mathbf{r} \in \mathrm{QR}_{\mathbf{N}}$, $s, t, s', t' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ such that $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^t, \mathbf{y}^t\mathbf{r})$ and $(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{s'}\mathbf{g}^{t'}, \mathbf{y}^{t'}\mathbf{h}^{\mathbf{r}})$.

1. The prover chooses $s'', t'' \in \mathbb{Z}_{\mathbf{N}}$ randomly, computes $(u, v) = (y_{\mathbf{N}}^{s''} g_{\mathbf{N}}^{t''}, y_{\mathbf{N}}^{t''} g_{\mathbf{N}}^{\mathbf{r}})$ and hands (u, v) to the verifier.
2. The following two protocols are executed in parallel:
   a) Protocol 16.24 on common input $\mathbf{g}, \mathbf{y}, \boldsymbol{\phi} \in \mathrm{QR}_{\mathbf{N}}$ and $g_{\mathbf{N}}, y_{\mathbf{N}}, \theta, \omega \in G_{\mathbf{N}}$ where $(\theta, \omega, \boldsymbol{\phi}) = (u^{\mathbf{v}^{-1}}, v^{\mathbf{v}^{-1}}, \mathbf{u}^{-1})$ and private input $t''\mathbf{v}^{-1}, s''\mathbf{v}^{-1} \in \mathbb{Z}_{\mathbf{N}}$ and $-t, -s \in [-2^{\kappa_r}\mathbf{N} + 1, 2^{\kappa_r}\mathbf{N} - 1]$.
   b) Protocol 16.16 on common input $\mathbf{y}, \mathbf{h}, \mathbf{v}' \in \mathrm{QR}_{\mathbf{N}}$, $g_{\mathbf{N}}, y_{\mathbf{N}}, v \in G_{\mathbf{N}}$ and private input $\mathbf{r}, t', t''$.

Lemma 16.27. *Protocol 16.26 is a $\{0,1\}^{\kappa_c}$-Σ-protocol.*

Proof. The completeness follows from the completeness of the subprotocols.

We now prove special soundness. Using Lemma 16.25 we can find $\zeta'', \tau'', \zeta, \tau$ such that

$$(\theta, \omega, \boldsymbol{\phi}) = (y_{\mathbf{N}}^{\tau''} g_{\mathbf{N}}^{\zeta''}, y_{\mathbf{N}}^{\zeta''} g_{\mathbf{N}}^{\mathbf{y}^{\zeta}}, \mathbf{y}^{\tau}\mathbf{g}^{\zeta}) \ .$$

Thus, we can compute $\boldsymbol{\rho}$ such that

$$(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^{\zeta}\mathbf{y}^{\tau}, \mathbf{g}^{\tau}\boldsymbol{\rho}) \quad \text{and} \quad (u, v) = (g_{\mathbf{N}}^{\zeta''} y_{\mathbf{N}}^{\tau''}, g_{\mathbf{N}}^{\tau''} y_{\mathbf{N}}^{\boldsymbol{\rho}}) \ .$$

Using Lemma 16.17 we can find ζ', τ_*'' and $\rho \in [0, \mathbf{N} - 1]$ such that

$$\mathbf{v}' = \mathbf{g}^{\zeta'}\mathbf{h}^{\rho} \quad \text{and} \quad v = y_{\mathbf{N}}^{\tau_*''} g_{\mathbf{N}}^{\rho} \ .$$

We may assume that $(\tau_*'', \rho) = (\tau'', \boldsymbol{\rho})$, since otherwise we can define $\eta_0 = \tau'' - \tau_*''$ and $\eta_1 = \boldsymbol{\rho} - \rho$ and Case 4 in Section 16.1 is satisfied.

On input $c \in \{0,1\}^{\kappa_c}$ the special zero-knowledge simulator chooses $u, v \in G_{\mathbf{N}}$ randomly and invokes the special zero-knowledge simulators of the subprotocols on input c. The generated pair (u, v) is identically distributed as in a real execution. Thus, it follows from Lemma 16.25 and Lemma 16.17 that the protocol is special honest verifier perfect zero-knowledge. □

Protocol 16.28 (Knowledge of a Root of a Committed Value).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{QR}_{\mathbf{N}}$ and $\mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}', \mathbf{C} \in \mathbb{Z}_{\mathbf{N}}^*$.
PRIVATE INPUT: $s, t, s', t', s'', e \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ and $\mathbf{r} \in \mathrm{QR}_{\mathbf{N}}$ such that $(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^s\mathbf{g}^t, \mathbf{y}^t\mathbf{r})$, $(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{s'}\mathbf{g}^{t'}, \mathbf{y}^{t'}\mathbf{r}^e)$ and $\mathbf{C} = \mathbf{y}^{s''}\mathbf{g}^e$.

1. The prover chooses $a, b \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ and $f, h, i, j \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$ randomly and computes
$$(\mathbf{A}_1, \mathbf{A}_2) = (\mathbf{y}^a\mathbf{g}^b\mathbf{u}^e, \mathbf{y}^b\mathbf{v}^e) \ , \tag{16.19}$$
$$(\mathbf{B}_1, \mathbf{B}_2) = (\mathbf{y}^f\mathbf{g}^h\mathbf{u}^i, \mathbf{y}^h\mathbf{v}^i) \ , \quad \text{and} \tag{16.20}$$
$$\mathbf{B}_3 = \mathbf{y}^j\mathbf{g}^i \ . \tag{16.21}$$
Then it hands $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$ to the verifier. The following protocols are executed in parallel with the protocol below:
   a) Protocol 16.20 parameterized with $z = (2^{\kappa_c}\mathbf{N})^2 + 2^{\kappa_c+2\kappa_r}\mathbf{N}$ on public input $\mathbf{g}$, $\mathbf{y}$, $(\mathbf{A}_1, \mathbf{A}_2)$, $(\mathbf{u}', \mathbf{v}')$ and private input $se + a$, $te + b$, s', t', and $\mathbf{r}^e$.
   b) Protocol 16.18 on public input $\mathbf{g}, \mathbf{y}, (\mathbf{u}, \mathbf{v})$ and private input $s, t, \mathbf{r}$.
2. The verifier chooses $c \in [0, 2^{\kappa_c} - 1]$ randomly and hands it to the prover.
3. The prover computes
$$d_1 = ca + f \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \tag{16.22}$$
$$d_2 = cb + h \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \tag{16.23}$$
$$d_3 = ce + i \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \quad \text{and} \tag{16.24}$$
$$d_4 = cs'' + j \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ . \tag{16.25}$$
4. The verifier checks that
$$\mathbf{A}_1^c\mathbf{B}_1, \mathbf{A}_2^c\mathbf{B}_2 = (\mathbf{y}^{d_1}\mathbf{g}^{d_2}\mathbf{u}^{d_3}, \mathbf{y}^{d_2}\mathbf{v}^{d_3}) \ , \quad \text{and} \tag{16.26}$$
$$\mathbf{C}^c\mathbf{B}_3 = \mathbf{y}^{d_4}\mathbf{g}^{d_3} \ . \tag{16.27}$$

Lemma 16.29. *Protocol 16.28 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. The verifier rejects if one of the three subprotocols fails or if there is a modular reduction in the computation of d_1, d_2, d_3 or d_4. It is easy to see that this happens with negligible probability. Thus, the protocol has overwhelming completeness.

We prove that the protocol is special sound. Suppose we have two transcripts $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, c, d_1, d_2, d_3, d_4)$ and $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, c', d_1', d_2', d_3', d_4')$ with $c \neq c'$ satisfying the equations in Step 4. Then we have
$$\mathbf{A}_1^{c-c'} = \mathbf{y}^{d_1-d_1'}\mathbf{g}^{d_2-d_2'}\mathbf{u}^{d_3-d_3'} \ ,$$
$$\mathbf{A}_2^{c-c'} = \mathbf{y}^{d_2-d_2'}\mathbf{v}^{d_3-d_3'} \ , \quad \text{and}$$
$$\mathbf{C}^{c-c'} = \mathbf{y}^{d_4-d_4'}\mathbf{g}^{d_3-d_3'} \ .$$

If $c-c'$ does not divide d_1-d_1', d_2-d_2', d_3-d_3', and d_4-d_4' we conclude similarly to previous proofs that Case 5 in Section 16.1 is satisfied.

Thus, we assume that $c-c'$ divides d_1-d_1', d_2-d_2', d_3-d_3', and define $\alpha=(d_1-d_1')/(c-c')$, $\beta=(d_2-d_2')/(c-c')$, $\varepsilon=(d_3-d_3')/(c-c')$, and $\zeta''=(d_4-d_4')/(c-c')$. This gives

$$\begin{aligned}\mathbf{A}_1 &= \mathbf{y}^{\alpha}\mathbf{g}^{\beta}\mathbf{u}^{\varepsilon}\ ,\\ \mathbf{A}_2 &= \mathbf{y}^{\beta}\mathbf{v}^{\varepsilon}\ ,\quad \text{and}\\ \mathbf{C} &= \mathbf{y}^{\zeta''}\mathbf{g}^{\varepsilon}\ .\end{aligned}$$

Using Lemma 16.19 we can find $\zeta,\tau,\mathbf{r}$ such that

$$(\mathbf{u},\mathbf{v}) = (\mathbf{y}^{\zeta}\mathbf{g}^{\tau},\mathbf{y}^{\tau}\mathbf{r})\ .$$

If we combine the equations we have

$$(\mathbf{A}_1,\mathbf{A}_1) = (\mathbf{y}^{\zeta\varepsilon+\alpha}\mathbf{g}^{\tau\varepsilon+\beta},\mathbf{y}^{\zeta\tau+\beta}\mathbf{r}^{\varepsilon})\ .$$

Using Lemma 16.21 we can find $\alpha_*,\beta_*,\zeta',\tau'$ such that

$$(\mathbf{A}_1,\mathbf{A}_2/\mathbf{u}',\mathbf{v}') = (\mathbf{y}^{\alpha_*}\mathbf{g}^{\beta_*},\mathbf{y}^{\beta_*}\mathbf{y}^{-\tau'},\mathbf{y}^{\zeta'}\mathbf{g}^{\tau'})\ .$$

If $(\zeta\varepsilon+\alpha,\tau\varepsilon+\beta)\neq(\alpha_*,\beta_*)$ then we set $\eta_0=\zeta\varepsilon+\alpha-\alpha_*$ and $\eta_1=\tau\varepsilon+\beta-\beta_*$ and conclude that Case 6 in Section 16.1 is satisfied. Thus, we assume that equality holds and have

$$(\mathbf{u}',\mathbf{v}') = (\mathbf{y}^{\zeta'}\mathbf{g}^{\tau'},\mathbf{y}^{\tau'}\mathbf{r}^{\varepsilon})\ .$$

This concludes the proof of special-soundness.

On input a challenge $c\in[0,2^{\kappa_c}-1]$ the special zero-knowledge simulator chooses $\mathbf{A}_1,\mathbf{A}_2\in\mathrm{QR}_{\mathbf{N}}$ and $d_1,d_2,d_3,d_4\in[0,2^{\kappa_c+2\kappa_r}\mathbf{N}-1]$ randomly and defines $\mathbf{B}_1,\mathbf{B}_2,\mathbf{B}_3$ by the equations in Step 4. Finally, the simulator invokes the special zero-knowledge simulators of the subprotocols on input c. The distribution of $(\mathbf{A}_1,\mathbf{A}_2)$ is statistically close to the distribution of this pair in a real execution, and both subprotocols are special honest verifier perfect zero-knowledge. Thus, the protocol is special honest verifier statistical zero-knowledge. □

Protocol 16.30 (Equality of Exponents of Committed Values).
COMMON INPUT: $\mathbf{g},\mathbf{y},\mathbf{h},\mathbf{u},\mathbf{v},\mathbf{C}\in\mathrm{QR}_{\mathbf{N}}$
PRIVATE INPUT: $r,s,t,w\in[0,2^{\kappa_r}\mathbf{N}-1]$ such that $(\mathbf{u},\mathbf{v})=(\mathbf{y}^r\mathbf{g}^s,\mathbf{y}^s\mathbf{h}^w)$ and $\mathbf{C}=\mathbf{y}^t\mathbf{g}^w$.

1. The prover chooses $a,b,e,f\in[0,2^{\kappa_c+2\kappa_r}\mathbf{N}-1]$, sets $(\boldsymbol{\mu},\boldsymbol{\nu})=(\mathbf{y}^a\mathbf{g}^b,\mathbf{y}^b\mathbf{h}^e)$ and $\mathbf{B}=\mathbf{g}^e\mathbf{y}^f$ and hands $(\boldsymbol{\mu},\boldsymbol{\nu},\mathbf{B})$ to the verifier.

2. The verifier randomly chooses $c\in[0,2^{\kappa_c}-1]$ and hands it to the prover.

3. The prover computes

$$\begin{aligned} d_1 &= cr + a \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \\ d_2 &= cs + b \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \\ d_3 &= ct + e \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \quad \text{and} \\ d_4 &= cw + f \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \end{aligned}$$

   and hands (d_1, d_2, d_3, d_4) to the verifier.

4. The verifier checks that $\mathbf{u}^c\boldsymbol{\mu} = \mathbf{y}^{d_1}\mathbf{g}^{d_2}$, $\mathbf{v}^c\boldsymbol{\nu} = \mathbf{y}^{d_2}\mathbf{h}^{d_4}$ and $\mathbf{C}^c\mathbf{B} = \mathbf{y}^{d_3}\mathbf{g}^{d_4}$.

Lemma 16.31. *Protocol 16.30 is a* $[0, 2^{\kappa_c} - 1]$-Σ-*protocol.*

Proof. An honest verifier will convince the verifier except possibly when there is a modular reduction in the computation of d_1, d_2, d_3, or d_4. It is easy to see that this happens with negligible probability. Thus, the protocol has overwhelming completeness.

Now we show that the protocol is special-sound. Assume that we have two lists $(\boldsymbol{\mu}, \boldsymbol{\nu}, \mathbf{B}, c, d_1, d_2, d_3, d_4)$ and $(\boldsymbol{\mu}, \boldsymbol{\nu}, \mathbf{B}, c', d'_1, d'_2, d'_3, d'_4)$ with $c \neq c'$ both satisfying the equations of Step 4. Then we have

$$\begin{aligned} (\mathbf{u}^{c-c'}, \mathbf{v}^{c-c'}) &= (\mathbf{y}^{d_1-d'_1}\mathbf{g}^{d_2-d'_2}, \mathbf{y}^{d_2-d'_2}\mathbf{h}^{d_4-d'_4}) \ , \quad \text{and} \\ \mathbf{C}^{c-c'} &= \mathbf{y}^{d_3-d'_3}\mathbf{g}^{d_4-d'_4} \ . \end{aligned}$$

If $c - c'$ does not divide $d_1 - d'_1$, $d_2 - d'_2$, $d_3 - d'_3$, and $d_4 - d'_4$ we conclude similarly to previous proofs that Case 5 in Section 16.1 is satisfied.

Thus, we assume that $c - c'$ divides $d_1 - d'_1$, $d_2 - d'_2$, $d_3 - d'_3$, and $d_4 - d'_4$ and define $\rho = (d_1 - d'_1)/(c - c')$, $\zeta = (d_2 - d'_2)/(c - c')$, $\tau = (d_3 - d'_3)/(c - c')$, $\omega = (d_4 - d'_4)/(c - c')$. This gives

$$\begin{aligned} (\mathbf{u}, \mathbf{v}) &= (\mathbf{y}^{\rho}\mathbf{g}^{\zeta}, \mathbf{y}^{\zeta}, \mathbf{h}^{\omega}) \ , \quad \text{and} \\ \mathbf{C} &= \mathbf{g}^{\omega}\mathbf{y}^{\tau} \ . \end{aligned}$$

This concludes the proof of special-soundness.

On input a challenge $c \in [0, 2^{\kappa_c} - 1]$ the special zero-knowledge simulator chooses $d_1, d_2, d_3, d_4 \in [0, 2^{\kappa_c+2\kappa_r} - 1]$ randomly and defines $\boldsymbol{\mu}$, $\boldsymbol{\nu}$ and $\mathbf{C}$ by the equations of Step 4. This gives a distribution equal to that of an honest execution. Thus, the protocol is special honest verifier perfect zero-knowledge. □

The following is a protocol, parameterized on k and l, is used to show that a committed value can be written as $ka + l$ for some a.

Protocol 16.32 (A Committed Value Can Be Written as $ka + l$).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{QR}_{\mathbf{N}}$ and $\mathbf{C} \in \mathbb{Z}^*_{\mathbf{N}}$.
PRIVATE INPUT: $a, t \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ such that $\mathbf{C} = \mathbf{y}^t\mathbf{g}^{ka+l}$.

1. The prover selects $e, f, h \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N}-1]$, $i \in [0, 2^{\kappa_c+2\kappa_r}k\mathbf{N}-1]$ at random, computes

$$\mathbf{A} = \mathbf{y}^e\mathbf{g}^a \ , \tag{16.28}$$
$$\mathbf{B}_1 = \mathbf{y}^h\mathbf{g}^f \ , \quad \text{and} \tag{16.29}$$
$$\mathbf{B}_2 = \mathbf{y}^i \ , \tag{16.30}$$

   and hands $(\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2)$ to the verifier.

2. The verifier randomly chooses $c \in [0, 2^{\kappa_c} - 1]$ and hands it to the prover.

3. The prover computes

$$d_1 = ca + f \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \tag{16.31}$$
$$d_2 = ce + h \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \quad \text{and} \tag{16.32}$$
$$d_3 = c(ek - t) + i \bmod 2^{\kappa_c+2\kappa_r}k\mathbf{N} \ , \tag{16.33}$$

   and hands (d_1, d_2, d_3) to the verifier.

4. The verifier checks that $\mathbf{A}^c\mathbf{B}_1 = \mathbf{y}^{d_2}\mathbf{g}^{d_1}$ and $(\mathbf{g}^l\mathbf{A}^k/\mathbf{C})^c\mathbf{B}_2 = \mathbf{y}^{d_3}$.

Lemma 16.33. *Protocol 16.32 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. The prover succeeds to convince the verifier unless there is a modular reduction in the computation of d_1, d_2, or d_3. It is easy to see that this happens with negligible probability. Thus, the protocol has overwhelming completeness.

Consider now special soundness. Assume we have lists $(\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, c, d_1, d_2, d_3)$ and $(\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, c', d_1', d_2', d_3')$, with $c \neq c'$, satisfying the equations of Step 4. We have

$$\mathbf{A}^{c-c'} = \mathbf{y}^{d_2-d_2'}\mathbf{g}^{d_1-d_1'} \text{ and}$$
$$(\mathbf{g}^l\mathbf{A}^k/\mathbf{C})^{c-c'} = \mathbf{y}^{d_3-d_3'} \ .$$

If $c - c'$ does not divide $d_1 - d_1'$, $d_2 - d_2'$, and $d_3 - d_3'$ we conclude similarly to previous proofs that Case 5 in Section 16.1 is satisfied.

Thus, we assume that $c - c'$ divides $d_1 - d_1'$, $d_2 - d_2'$, and $d_3 - d_3'$ and define $\alpha = (d_1 - d_1')/(c - c')$, $\varepsilon = (d_2 - d_2')/(c - c')$, and $\zeta = (d_3 - d_3')/(c - c')$. This gives

$$\mathbf{A} = \mathbf{y}^{\varepsilon}\mathbf{g}^{\alpha} \quad \text{and} \quad \mathbf{g}^l\mathbf{A}^k/\mathbf{C} = \mathbf{y}^{\zeta}$$

and we conclude that

$$\mathbf{C} = \mathbf{y}^{k\varepsilon-\zeta}\mathbf{g}^{k\alpha+l} \ .$$

On input $c \in [0, 2^{\kappa_c} - 1]$ the special zero-knowledge simulator chooses $\mathbf{A} \in \mathrm{QR}_{\mathbf{N}}$, $d_1, d_2 \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$, $d_3 \in [0, 2^{\kappa_c+2\kappa_r}k\mathbf{N} - 1]$ randomly and defines $\mathbf{B}_1, \mathbf{B}_2$ by the equations in Step 4. This gives a distribution that is statistically close to that in the real protocol. Thus, the protocol is special honest verifier statistical zero-knowledge. □

From these building blocks we can now present the proof that a committed signature is valid.

Protocol 16.34 (Validity of Committed Signature from Hash).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{z} \in \mathrm{QR}_{\mathbf{N}}$, $\mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}', \mathbf{C}, \mathbf{C}' \in \mathbb{Z}^*_{\mathbf{N}}$, $e' \in [2^{\kappa}, 2^{\kappa+1} - 1]$.
PRIVATE INPUT: $r, s, r', s', t, t' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$, $e \in [2^{\kappa}, 2^{\kappa+1} - 1]$, and $w_\alpha \in \mathbb{Z}_{q_2}$ such that

$$\begin{aligned}
(\mathbf{u}, \mathbf{v}) &= (\mathbf{y}^s \mathbf{g}^r, \mathbf{y}^r \boldsymbol{\sigma}) \ , \\
(\mathbf{u}', \mathbf{v}') &= (\mathbf{y}^{s'} \mathbf{g}^{r'}, \mathbf{y}^{r'} \boldsymbol{\sigma}') \ , \\
\mathbf{C} &= \mathbf{y}^t \mathbf{g}^e \ , \\
\mathbf{C}' &= \mathbf{y}^{t'} \mathbf{g}^{w_\alpha} \quad , \quad \text{and} \\
\mathsf{Vf}^{\mathsf{cs}}_{\mathrm{id}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, (\mathbf{N},\mathbf{h},\mathbf{z},e')}(w_\alpha, (e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')) &= 1 \ .
\end{aligned}$$

In other words $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$ is a valid Cramer-Shoup signature of w_α if the first hash function is the identity map.

1. Let $\mathbf{z}'$ denote $(\boldsymbol{\sigma}')^{e'} \mathbf{h}^{-w_\alpha}$. The prover chooses ζ, τ, ζ', τ', ζ'', τ'', ζ''', τ''', ζ'''', $\tau'''' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ and sets

$$\begin{aligned}
(\boldsymbol{\mu}, \boldsymbol{\nu}) &= (\mathbf{y}^{\zeta} \mathbf{g}^{\tau}, \mathbf{y}^{\tau} \boldsymbol{h}^{-w_\alpha}) \ , \\
(\boldsymbol{\mu}', \boldsymbol{\nu}') &= (\mathbf{y}^{\zeta'} \mathbf{g}^{\tau'}, \mathbf{y}^{\tau'} \mathbf{z}') \ , \\
(\boldsymbol{\mu}'', \boldsymbol{\nu}'') &= (\mathbf{y}^{\zeta''} \mathbf{g}^{\tau''}, \mathbf{y}^{\tau''} \boldsymbol{\sigma}^e) \ , \\
(\boldsymbol{\mu}''', \boldsymbol{\nu}''') &= (\mathbf{y}^{\zeta'''} \mathbf{g}^{\tau'''}, \mathbf{y}^{\zeta'''} H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{z}')) \ , \quad \text{and} \\
(\boldsymbol{\mu}'''', \boldsymbol{\nu}'''') &= (\mathbf{y}^{\zeta''''} \mathbf{g}^{\tau''''}, \mathbf{y}^{\zeta''''} \mathbf{h}^{-H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{z}')}) \ .
\end{aligned}$$

   Then it hands $(\boldsymbol{\mu}, \boldsymbol{\nu})$, $(\boldsymbol{\mu}', \boldsymbol{\nu}')$, $(\boldsymbol{\mu}'', \boldsymbol{\nu}'')$, $(\boldsymbol{\mu}''', \boldsymbol{\nu}''')$, and $(\boldsymbol{\mu}'''', \boldsymbol{\nu}'''')$ to the verifier.

2. The following protocols are run in parallel

   a) Protocol 16.18 on the public input $\mathbf{g}, \mathbf{y}, (\mathbf{u}, \mathbf{v})$ and private input $s, r, \boldsymbol{\sigma}$ to show that the prover knows how to open the commitment $(\mathbf{u}, \mathbf{v})$.

   b) Protocol 16.18 on the public input $\mathbf{g}, \mathbf{y}, (\mathbf{u}', \mathbf{v}')$ and private input $s', r', \boldsymbol{\sigma}'$ to show that the prover knows how to open the commitment $(\mathbf{u}', \mathbf{v}')$.

   c) Protocol 16.30 on public input $\mathbf{g}, \mathbf{y}, \mathbf{h}, (\boldsymbol{\mu}, \boldsymbol{\nu}), (\mathbf{C}')^{-1}$ and private input ζ, τ, $-t'$, $-w_\alpha$ to show that $(\boldsymbol{\mu}, \boldsymbol{\nu})$ is a commitment of $\mathbf{h}^{-w_\alpha}$.

   d) Protocol 16.20 with $z = \mathbf{N} + \mathbf{N}2^{\kappa+1}$ on public input $\mathbf{g}$, $\mathbf{y}$, $(\boldsymbol{\mu}', \boldsymbol{\nu}')$, $(\boldsymbol{\mu}(\mathbf{u}')^{e'}, \boldsymbol{\nu}(\mathbf{v}')^{e'})$ and private input ζ', τ', $\zeta + s'e'$, $\tau + r'e'$ to show that $(\boldsymbol{\mu}', \boldsymbol{\nu}')$ is a commitment of $\mathbf{z}'$.

e) Protocol 16.28 on public input $\mathbf{g}, \mathbf{y}, (\mathbf{u}, \mathbf{v}), (\boldsymbol{\mu}'', \boldsymbol{\nu}''), \mathbf{C}$ and private exponents $s, r, \zeta'', \tau'', t, e$. This shows that $(\boldsymbol{\mu}'', \boldsymbol{\nu}'')$ hides the value hidden in $(\mathbf{u}, \mathbf{v})$ to the power of the value hidden in $\mathbf{C}$.

f) Protocol 16.26 on public input $\mathbf{g}, \mathbf{y}, \mathbf{g}, (\boldsymbol{\mu}', \boldsymbol{\nu}'), (\boldsymbol{\mu}''', \boldsymbol{\nu}'''), g_{\mathbf{N}}, y_{\mathbf{N}}$ and ζ', τ', ζ''', τ''' as private input to show that $(\boldsymbol{\mu}''', \boldsymbol{\nu}''')$ is a commitment of a Shamir hash of $\mathbf{z}'$.

g) Protocol 16.26 on public input $\mathbf{g}, \mathbf{y}, \mathbf{h}^{-1}, (\boldsymbol{\mu}''', \boldsymbol{\nu}'''), (\boldsymbol{\mu}'''', \boldsymbol{\nu}''''), g_{\mathbf{N}}, y_{\mathbf{N}}$ and private input ζ''', τ''', ζ'''', τ'''' to show that $(\boldsymbol{\mu}'''', \boldsymbol{\nu}'''')$ commits to $\mathbf{h}$ to the power of $-H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{z}')$.

h) Protocol 16.22 with $z = 2\mathbf{N}$ on public input $\mathbf{g}, \mathbf{y}, (\boldsymbol{\mu}''\boldsymbol{\mu}'''', \boldsymbol{\nu}''\boldsymbol{\nu}''''), \mathbf{z}$ with private input $\zeta'' + \zeta''''$, $\tau'' + \tau''''$ to finally show that the signature is valid.

i) Protocol 16.32 with $k = 4$ and $l = 3$ on public input $\mathbf{g}, \mathbf{y}, \mathbf{C}$ and private input e, t to prove that e is odd and different from e'.

j) Protocol 16.14 on public input $\mathbf{g}, \mathbf{y}, \mathbf{C}, 2^{\kappa}, 2^{\kappa+1} - 1$ and private input e, t to prove that e belongs to the correct interval.

Lemma 16.35. *Protocol 16.34 is a* $[0, 2^{\kappa_c} - 1]$-Σ-*protocol.*

Proof. Since there is a natural bijection between $[0, 2^{\kappa_c} - 1]$ and $\{0,1\}^{\kappa_c}$, the resulting protocol is a $[0, 2^{\kappa_c} - 1]$-Σ-protocol. The completeness follows from the completeness of the subprotocols.

Consider now special soundness. Using Lemma 16.19 we can find $\zeta, \rho, \mathbf{r}$ and $\zeta', \rho', \mathbf{r}'$ such that

$$(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^{\zeta}\mathbf{g}^{\rho}, \mathbf{y}^{\rho}\mathbf{r}) \text{ and} \tag{16.34}$$

$$(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{\zeta'}\mathbf{g}^{\rho'}, \mathbf{y}^{\rho'}\mathbf{r}') \ . \tag{16.35}$$

Using Lemma 16.31 we can find $\zeta_1, \rho_1, \omega, \tau$ such that

$$(\boldsymbol{\mu}, \boldsymbol{\nu}) = (\mathbf{y}^{\zeta_1}\mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1}\mathbf{h}^{-\omega}) \text{ and} \tag{16.36}$$

$$\mathbf{C}' = \mathbf{y}^{-\tau}\mathbf{g}^{\omega} \ . \tag{16.37}$$

Using Lemma 16.21 we can find $\zeta_1, \tau_1, \mathbf{r}_1, \zeta_e, \tau_e, \mathbf{r}_e$ such that

$$(\boldsymbol{\mu}', \boldsymbol{\nu}') = (\mathbf{y}^{\zeta_1}\mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1}\mathbf{r}_1) \text{ and} \tag{16.38}$$

$$(\boldsymbol{\mu}(\mathbf{u}')^{e'}, \boldsymbol{\nu}(\mathbf{v}')^{e'}) = (\mathbf{y}^{\zeta_e}\mathbf{g}^{\rho_e}, \mathbf{y}^{\rho_e}\mathbf{r}_1) \ . \tag{16.39}$$

If we combine Equations (16.34), (16.36), and (16.39) we get

$$(\mathbf{y}^{\zeta_1 + \zeta' e'}\mathbf{g}^{\rho_1 + \rho' e'}, \mathbf{y}^{\rho_1 + \rho' e'}\mathbf{h}^{-\omega}\mathbf{r}'^{e'}) = (\mathbf{y}^{\zeta_e}\mathbf{g}^{\rho_e}, \mathbf{y}^{\rho_e}\mathbf{r}_1) \ .$$

We may assume that $(\zeta_1 + \zeta' e', \rho_1 + \rho' e') = (\zeta_e, \rho_e)$ and thus $\mathbf{r}_e = \mathbf{h}^{-\omega}\mathbf{r}'^{e'}$, since otherwise Case 6 in Section 16.1 is satisfied. Thus, we have

$$(\boldsymbol{\mu}', \boldsymbol{\nu}') = (\mathbf{y}^{\zeta_1}\mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1}\mathbf{h}^{-\omega}\mathbf{r}'^{e'}) \ .$$

Using Lemma 16.29 we can find $\zeta_2, \tau_2, \mathbf{r}_2, \zeta_2', \tau_2', \varepsilon, \zeta_2''$ such that

$$\begin{aligned}(\mathbf{u}, \mathbf{v}) &= (\mathbf{y}^{\zeta_2}\mathbf{g}^{\rho_2}, \mathbf{y}^{\rho_2}\mathbf{r}_2) \ , \\ (\boldsymbol{\mu}'', \boldsymbol{\nu}'') &= (\mathbf{y}^{\zeta_2'}\mathbf{g}^{\rho_2'}, \mathbf{y}^{\rho_2'}\mathbf{r}_2^{\varepsilon}) \ , \quad \text{and} \\ \mathbf{C} &= \mathbf{y}^{\zeta_2''}\mathbf{g}^{\varepsilon} \ .\end{aligned}$$

We may assume that $(\zeta_2, \rho_2) = (\zeta, \rho)$ and thus $\mathbf{r}_2 = \mathbf{r}$, since otherwise Case 6 in Section 16.1 is satisfied.

Using Lemma 16.27 we can find $\zeta_3, \tau_3, \mathbf{r}_3, \zeta_3', \tau_3'$ such that

$$\begin{aligned}(\boldsymbol{\mu}', \boldsymbol{\nu}') &= (\mathbf{y}^{\zeta_3}\mathbf{g}^{\rho_3}, \mathbf{y}^{\rho_3}\mathbf{r}_3) \text{ and} \\ (\boldsymbol{\mu}''', \boldsymbol{\nu}''') &= (\mathbf{y}^{\zeta_3'}\mathbf{g}^{\rho_3'}, \mathbf{y}^{\rho_3'}\mathbf{h}^{\mathbf{r}_3}) \ .\end{aligned}$$

We may assume that $(\zeta_3, \rho_3) = (\zeta_1', \rho_1')$ and thus $\mathbf{r}_3 = \mathbf{h}^{-\omega}\mathbf{r}'^{e'}$, since otherwise Case 6 in Section 16.1 is satisfied.

Using Lemma 16.27 we can find $\zeta_4, \tau_4, \mathbf{r}_4, \zeta_4', \tau_4'$ such that

$$\begin{aligned}(\boldsymbol{\mu}''', \boldsymbol{\nu}''') &= (\mathbf{y}^{\zeta_4}\mathbf{g}^{\rho_4}, \mathbf{y}^{\rho_4}\mathbf{r}_4) \text{ and} \\ (\boldsymbol{\mu}'''', \boldsymbol{\nu}'''') &= (\mathbf{y}^{\zeta_4'}\mathbf{g}^{\rho_4'}, \mathbf{y}^{\rho_4'}\mathbf{h}^{\mathbf{r}_4}) \ .\end{aligned}$$

We may assume that $(\zeta_4, \rho_4) = (\zeta_3', \rho_3')$ and thus $\mathbf{r}_4 = \mathbf{h}^{\mathbf{r}_3}$, since otherwise Case 6 in Section 16.1 is satisfied.

Using Lemma 16.23 we can find ζ_5, τ_5 and $\mathbf{z}$ such that

$$(\boldsymbol{\mu}''\boldsymbol{\mu}'''', \boldsymbol{\nu}''\boldsymbol{\nu}'''') = (\mathbf{y}^{\zeta_5}\mathbf{g}^{\rho_5}, \mathbf{y}^{\rho_5}\mathbf{z}) \ .$$

We may assume that $(\zeta_5, \rho_5) = (\zeta_2' + \zeta_4', \rho_2' + \rho_3')$ and thus we have $\mathbf{z} = \mathbf{r}^{\varepsilon}\mathbf{h}^{\mathbf{r}_3} = \mathbf{r}^{\varepsilon}\mathbf{h}^{H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{h}^{-\omega}\mathbf{r}'^{e'})}$, since otherwise Case 6 in Section 16.1 is satisfied.

Using Lemma 16.33 we can find α, τ' such that

$$\mathbf{C} = \mathbf{y}^{\tau'}\mathbf{g}^{4\alpha+3} \ .$$

We may assume that $\varepsilon = 4\alpha + 3$, since otherwise Case 6 in Section 16.1 is satisfied.

Using Lemma 16.15 we can find $\zeta_6, \varepsilon_6 \in [2^{\kappa}, 2^{\kappa+1} - 1]$ such that

$$\mathbf{C} = \mathbf{y}^{\zeta_6}\mathbf{g}^{\varepsilon_6} \ .$$

We may assume that $\varepsilon_6 = \varepsilon$, since otherwise Case 6 in Section 16.1 is satisfied.

To summarize we have found $\zeta, \rho, \zeta', \rho', \tau, \zeta_2''$ and $\omega, (\varepsilon, \mathbf{r}, \mathbf{r}')$, with $\varepsilon \in [2^{\kappa}, 2^{\kappa+1} - 1]$ and $\varepsilon = 3 \bmod 4$, and ζ, ρ, such that

$$\begin{aligned}(\mathbf{u}, \mathbf{v}) &= (\mathbf{y}^{\zeta}\mathbf{g}^{\rho}, \mathbf{y}^{\rho}\mathbf{r}) \ , \\ (\mathbf{u}', \mathbf{v}') &= (\mathbf{y}^{\zeta'}\mathbf{g}^{\rho'}, \mathbf{y}^{\rho'}\mathbf{r}') \ , \\ \mathbf{C}' &= \mathbf{y}^{-\tau}\mathbf{g}^{\omega} \ , \\ \mathbf{C} &= \mathbf{y}^{\zeta_2''}\mathbf{g}^{\varepsilon} \ , \quad \text{and} \\ \mathsf{Vf}^{\mathsf{cs}}_{\mathrm{id}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, (\mathbf{N},\mathbf{h},\mathbf{z},e')}(\omega, (\varepsilon, \mathbf{r}, \mathbf{r}')) &= 1 \ .\end{aligned}$$

This concludes the proof of special-soundness.

On input a challenge $c \in \{0,1\}^{\kappa_c}$ the special zero-knowledge simulator chooses random elements $\boldsymbol{\mu}', \boldsymbol{\mu}'', \boldsymbol{\mu}''', \boldsymbol{\mu}'''', \boldsymbol{\nu}', \boldsymbol{\nu}'', \boldsymbol{\nu}''', \boldsymbol{\nu}'''' \in \mathrm{QR}_{\mathbf{N}}$ and invokes the special zero-knowledge simulator of each subprotocol on input c. The distribution of the above elements is statistically close to their distribution in the real protocol. Since all subprotocols are special honest verifier statistical zero-knowledge, so is the combined protocol. □

16.5 The Complete Protocol

We are finally ready to give the complete proof of a correct signature corresponding to the proof in Step 3 of Algorithm 15.3. The common input consists of a chain of cryptotexts and commitments of a $\mathcal{SS}^{\mathsf{cs}}$ signature of the public keys corresponding to the path of the signer in the tree.

Protocol 16.36 (Valid $\mathcal{HGS}$ Signature).
Common Input:

$$H^{\mathsf{CHP}} = (h_1, \ldots, h_\delta) \in G_{q_2}^\delta \qquad (u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1} \in G_{q_3}^{4\delta}$$
$$g_1, y_1 \in G_{q_1}, g_2, y_2 \in G_{q_2}, g_3, y_3 \in G_{q_3} \qquad C_\delta \in G_{q_3}^4$$
$$y_{\alpha_0} \in G_{q_3}, Y \in G_{q_3}^5 \qquad \mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}', \mathbf{C} \in \mathrm{QR}_{\mathbf{N}}$$
$$e' \in [2^\kappa, 2^{\kappa+1} - 1]$$
$$\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{z} \in \mathrm{QR}_{\mathbf{N}}$$

Private Input: $(r_0, \ldots, r_\delta) \in \mathbb{Z}_{q_3}^{\delta+1}$, $(y_{\alpha_1}, \ldots, y_{\alpha_\delta}) \in G_{q_3}^\delta$, $e \in [2^\kappa, 2^{\kappa+1} - 1]$, and $(r, s, r', s', t) \in [0, 2^{\kappa_r}\mathbf{N} - 1]^5$ such that

$$\begin{aligned}
(u_l, v_l) &= E_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
(u'_l, v'_l) &= E_{(y_{\alpha_l}, g_3)}(1, r'_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
C_\delta &= E_Y^{\mathsf{cs}}(y_{\alpha_\delta}, r_\delta) \ , \\
\mathbf{u} &= \mathbf{y}^s \mathbf{g}^r \ , \\
\mathbf{u}' &= \mathbf{y}^{s'} \mathbf{g}^{r'} \ , \\
\mathbf{C} &= \mathbf{y}^t \mathbf{g}^e \ , \quad \text{and} \\
\mathsf{Vf}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, (\mathbf{N},\mathbf{h},\mathbf{z},e')}&((y_{\alpha_1}, \ldots, y_{\alpha_\delta}), (e, \mathbf{v}/\mathbf{y}^r, \mathbf{v}'/\mathbf{y}^{r'})) = 1 \ .
\end{aligned}$$

1. The prover chooses $s, t \in \mathbb{Z}_{q_2}$ and $t' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ randomly and computes $(\mu, \nu) = (y_1^s g_1^t, y_1^t g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})})$, $\mathbf{C}' = \mathbf{y}^{t'} \mathbf{g}^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})}$. The prover hands (μ, ν) and $\mathbf{C}'$ to the verifier.

2. The following protocols are executed in parallel

a) Protocol 16.12 on public input $g_3, y_3, y_{\alpha_0}, g_2, y_2, g_1, y_1, H^{\mathsf{CHP}}$, $(u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, (\mu, \nu), Y, C_\delta$ and private input $r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta$, $y_{\alpha_1}, \ldots, y_{\alpha_\delta}$, s, t.

b) Protocol 16.16 on public input $\mathbf{g}, \mathbf{y}, \mathbf{C}', g_1, y_1, \nu$ and private input $H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$, t', and t.

c) Protocol 16.34 on public input $\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{z}, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \mathbf{C}', e'$ and private input $r, s, r', s', t, t', e, H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})$.

Lemma 16.37. *Protocol 16.36 is a* $[0, 2^{\kappa_c} - 1] \times \mathbb{Z}_{q_2}$*-*$\Sigma$*-protocol.*

Proof. The completeness follows from the completeness of the subprotocols.

Using Lemma 16.13 we can find $\rho_0, \rho'_0, \ldots, \rho_{\delta-1}, \rho'_{\delta-1}, \rho_\delta, \gamma_0, \ldots, \gamma_\delta$ with $\gamma_0 = y_{\alpha_0}$, and ζ'', τ'' such that

$$\begin{aligned}(u_l, v_l, u'_l, v'_l) &= (E_{(\gamma_l, g_3)}(\gamma_{l+1}, \rho_l), E_{(\gamma_l, g_3)}(1, \rho'_l)) \text{ for } l = 0, \ldots, \delta - 1 \ , \\ C_\delta &= E_Y^{\mathsf{cs}}(\gamma_\delta, \rho_\delta) \ , \quad \text{and} \\ (\mu, \nu) &= (y_1^{\zeta''} g_1^{\tau''}, y_1^{\tau''} g_1^{H^{\mathsf{CHP}}(\gamma_1, \ldots, \gamma_\delta)}) \ .\end{aligned}$$

Using Lemma 16.17 we can find $\omega, \zeta''', \tau'''$ such that

$$\mathbf{C}' = \mathbf{y}^{\zeta'''} \mathbf{g}^{\omega} \quad \text{and} \quad \nu = y_1^{\tau'''} g_1^{\omega} \ .$$

We may assume that $\omega = H^{\mathsf{CHP}}(\gamma_1, \ldots, \gamma_\delta)$, since otherwise Case 1 in Section 16.1 is satisfied. Using Lemma 16.35 we can find $\rho, \zeta, \rho', \zeta', \tau, \tau', \varepsilon, \omega^*$ and $\mathbf{r}, \mathbf{r}' \in \mathrm{QR}_{\mathbf{N}}$ such that

$$\begin{aligned}(\mathbf{u}, \mathbf{v}) &= (\mathbf{g}^{\zeta} \mathbf{y}^{\rho}, \mathbf{g}^{\rho} \mathbf{r}) \ , \\ (\mathbf{u}', \mathbf{v}') &= (\mathbf{g}^{\zeta'} \mathbf{y}^{\rho'}, \mathbf{g}^{\rho'} \mathbf{r}') \ , \\ \mathbf{C} &= \mathbf{y}^{\tau} \mathbf{g}^{\varepsilon} \ , \\ \mathbf{C}' &= \mathbf{y}^{\tau'} \mathbf{g}^{\omega^*} \ , \quad \text{and} \\ \mathsf{Vf}^{\mathsf{cs}}_{\mathrm{id}, H^{\mathsf{Sh}}_{(\mathbf{N}, \mathbf{g})}, (\mathbf{N}, \mathbf{h}, \mathbf{z}, e')}(\omega, (\varepsilon, \mathbf{r}, \mathbf{r}')) &= 1 \ .\end{aligned}$$

We may assume that $\omega = \omega^*$, since otherwise Case 5 in Section 16.1 is satisfied. This concludes the proof of special-soundness.

On input $(b, c) \in [0, 2^{\kappa_c} - 1] \times \mathbb{Z}_{q_2}$ the special zero-knowledge simulator chooses $\mu, \nu \in G_{q_1}$ and $\mathbf{C}' \in \mathrm{QR}_{\mathbf{N}}$ randomly and invokes the special zero-knowledge simulator of each subprotocol on input b or c as appropriate. Since the distribution of $(\mu, \nu, \mathbf{C}')$ is statistically close to the corresponding elements in a real execution and the subprotocols are special honest verifier statistical zero-knowledge, then so is the combined protocol. □

16.5.1 A Computationally Convincing Proof of Knowledge

We are finally ready to prove the main results of this chapter.

Proposition 16.38. *Protocol 16.36 is honest verifier statistical zero-knowledge.*

Proof. This is an immediate consequence of 16.37. □

Proposition 16.39. *Protocol 16.36 is a computationally convincing proof of knowledge with regards to the distribution of the special parameters* $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{y}, g_{\mathbf{N}}, y_{\mathbf{N}})$ *and* $\overline{g} = (q_0, g_1, y_1, g_2, y_2, g_3, y_3)$*, under the DL-assumption and the strong RSA-assumption.*

Proof. We know from Lemma 16.37 that the protocol is a Σ-protocol for the relation $R = R_{\mathrm{HGS}} \vee R_{\mathrm{DL}} \vee R_{\mathrm{SRSA}}$. From Lemma 2.40 follows that the protocol is a proof of knowledge for the relation $R_{\mathrm{HGS}} \vee R_{\mathrm{DL}} \vee R_{\mathrm{SRSA}}$. Let $\mathcal{X}$ be the extractor. Then we have for every prover P^* and constant c that if $\Pr_{(\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}}[\delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \kappa^{-c}$ then

$$\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \in R \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] \tag{16.40}$$

is overwhelming.

We argue that $\mathcal{X}$ is an extractor for a computationally convincing proof of knowledge for the relation R_{HGS}. The requirement on the running time of $\mathcal{X}$ in Definition 2.28 follows immediately. Suppose that the requirement on the output of $\mathcal{X}$ in Definition 2.28 does not hold. Then there exists an adversary P^*, a constant c, and an infinite index set $\mathcal{N}$ such that

$$\begin{aligned} \Pr_{(\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}}[\delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] &\geq \kappa^{-c} \\ \Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \notin R_{\mathrm{HGS}} \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] &\geq \kappa^{-c} \ . \end{aligned}$$

The union bound and the fact that the probability in Equation (16.40) is overwhelming implies that

$$\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \in R_{\mathrm{DL}} \vee R_{\mathrm{SRSA}} \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}]$$

is at least $\frac{1}{2\kappa^c}$ and we conclude that $\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \in R_{\mathrm{DL}} \vee R_{\mathrm{SRSA}}] \geq \frac{1}{2\kappa^{2c}}$.

Denote by $t(\kappa)$ the expected running time of $\mathcal{X}^{P^*}$. We define an adversary A that given input $(\mathbf{\Gamma},\overline{g})$ simulates $\mathcal{X}^{P^*}$ except that it halts after $4\kappa^{2c}t(\kappa)$ steps. If the simulation is not completed it outputs $\perp$. Otherwise it outputs the output of $\mathcal{X}^{P^*}$. Thus, the running time of A is polynomial.

Markov's inequality implies that the probability that the simulation is interrupted is at most $\frac{1}{4\kappa^{2c}}$. The union bound then implies that

$$\Pr[A(\mathbf{\Gamma},\overline{g}) \in R_{\mathrm{DL}} \vee R_{\mathrm{SRSA}}] \geq \frac{1}{4\kappa^{2c}} \ .$$

It follows from Lemma 3.12 and Lemma 3.5 that this contradicts either the DL-assumption, Definition 3.4, or the strong RSA-assumption, Definition 3.11, and the proposition follows. □

It turns out that in the analysis of the hierarchical group signature scheme we need a somewhat stronger statement. Consider the protocol where the prover and verifier are given as special input not only $(\mathbf{\Gamma}, \overline{g})$, but also a tree T, a pair of maps $sk : \mathcal{V}(T) \to \mathbb{Z}_{q_3}$ and $pk : \mathcal{V}(T) \to G_{q_3}$ and a set of leaves $\mathcal{L} \subset \mathcal{L}(T)$. The protocol is identical to π_{hgs} except that the verifier checks that there does not exist a path $\alpha_0, \ldots, \alpha_\delta$ in the tree T such that

$$(D_{1/sk(\alpha_0)}(u_0, v_0), \ldots, D_{1/sk(\alpha_{\delta-1})}(u_{\delta-1}, v_{\delta-1})) = (pk(\alpha_1), \ldots, pk(\alpha_\delta)) \ .$$

Denote the resulting protocol by π'_{hgs}. The following result follows similarly to the proposition above.

Proposition 16.40. *Let T be a tree with all leaves at the same depth. Then Protocol 16.36 is a computationally convincing proof of knowledge with regards to the distribution of $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{y}, g_{\mathbf{N}}, y_{\mathbf{N}})$, $\overline{g} = (q_0, g_1, y_1, g_2, y_2, g_3, y_3)$, and (pk, sk), under the DL-assumption and the strong RSA-assumption.*

16.6 Complexity Analysis

We now analyze the number of exponentiations needed for some typical parameters. Table 16.1 shows the number of exponentiations necessary for each protocol.

Since Protocol 16.36 corresponds to the proof of a signature, this is what we need to evaluate to find the number of exponentiations of the complete protocol. Since the bulk of computations stem from the $(\delta + 3)$ executions of the double-decker exponentiation proofs that are based on cut-and-choose techniques we consider how to speed up these exponentiations. It can be noted that all exponentiations in these protocols are fixed-based exponentiations. In [31] a technique to use pre-computation to speed up such computations is given. The idea is to represent the exponent in basis b and pre-compute g^{b^i} for $i = 1, 2, \ldots, \log_b m$ where m is the maximum value of the exponent, in our case usually 2^κ. By pre-computing also cross-products $g^{b_i} g^{b_j}$ we can speed up the algorithm by a factor of (almost) two by paying in larger storage requirements.

For each call to Protocol 16.6 or 16.24 the verifier will need to perform precomputations for two new bases. All other bases in these two cut-and-choose protocols are fixed throughout the execution, and thus only requires one precomputation phase, which also may be stored between executions.

A realistic example may be $\kappa = 1024$ and $\kappa_c = 160$. By choosing parameters appropriately, one fixed-base exponentiation then takes about 0.075 of the time for a general exponentiation, and the setup phase takes about $\frac{4}{3}10$ general exponentiations with a storage requirement of about 2.5 Mb per base involved. By evaluating the expressions in Table 16.1 and adding a setup phase for 10 bases, we

Prot.	Prover	Verifier	Setup
16.2	$25\delta - 2$	$30\delta - 3$	
16.4	$2 + (16.6)$	$2 + (16.6)$	
16.6	$7.5\kappa_c$	$7.5\kappa_c$	2 bases
16.8	5	8	
16.10	3	6	
16.12	$7\delta + 2 + (16.2)+$ $(\delta + 1) \cdot (16.4)$ $(16.8) + (16.10)$	$(16.2)+$ $(\delta + 1) \cdot (16.4)+$ $(16.8) + (16.10)$	
16.16	10	18	
16.18	2	3	
16.20	2	3	
16.22	(16.20)	(16.20)	
16.26	$6 + (16.16) + (16.24)$	$(16.16) + (16.24)$	
16.24	$7.5\kappa_c$	$7.5\kappa_c$	2 bases
16.28	$12 + 2 \cdot (16.18) + (16.20)$	$8 + 2 \cdot (16.18) + (16.20)$	
16.30	4	9	
16.32	7	7	
16.14	6	12	
16.34	$20 + 2 \cdot (16.18)+$ $(16.20) + (16.22)+$ $2 \cdot (16.26) + (16.28)+$ $(16.30) + (16.32) + (16.14)$	$2 \cdot (16.18)+$ $(16.20) + (16.22)+$ $2 \cdot (16.26) + (16.28)+$ $(16.30) + (16.32) + (16.14)$	
16.36	$6 + \delta + (16.12)+$ $(16.16) + (16.34)$	$\delta + (16.12)+$ $(16.16) + (16.34)$	

Table 16.1: Number of exponentiations in each subprotocol

get that generating and verifying a signature takes time equivalent to about 1000 general exponentiations. This can be improved by using more efficient exponentiation algorithms, see, e.g., [109]. We have, for example, not used the fact that many exponentiations are simultaneous multiple exponentiations.

We now look at the size of a signature. Also here the cut-and-choose protocols contribute the most. For each protocol execution, $7\kappa_c$ values need to be stored, each of which has κ bits. This gives a total of $7\kappa_c(\delta + 3)$ κ-bit numbers. With the same parameters as above this gives a signature size of about 1 Mb.

In the above computations we have ignored the fact that some computations involve exponents slightly larger than κ bits, but the numbers still gives a good picture of the amount of computation necessary. One can conclude that on a standard PC a signature can be created in about a minute.

Chapter 17

Conclusions of Part III

We have introduced the new notion of hierarchical group signatures, and given formal definitions. We have also given two constructions. The first is secure under the existence of a family of trapdoor permutations, and the second is secure under standard concrete complexity assumptions. Although we believe that our work is a good starting point, much work on hierarchical group signatures remain.

Our definitions only consider static groups. Thus, an important research problem is to formalize hierarchical group signatures for dynamic groups, i.e., when group managers and signers can be added and removed. It is far from obvious what the semantics of such a scheme should be. Another important problem is to find a more practical scheme, even for static groups. Finally, our solution postulates a trusted key generator. Future work should try to eliminate this deficiency.

As part of the construction of the proof of knowledge of Chapter 16 we have constructed a reasonably practical zero-knowledge proof of knowledge that a committed value is a signature of an encrypted message. We are not aware of any previous method to do this efficiently.

There are many interesting variations of group signatures, some of which have not yet been properly formalized and studied. These variations are not always of immediate practical value, but the problem of realizing them is still an interesting cryptographic problem that inspire the development of new methods and techniques.

Bibliography

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[2] M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-centers. In *Advances in Cryptology – Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer Verlag, 1998.

[3] M. Abe, R. Cramer, and S. Fehr. Non-interactive distributed-verifier proofs and proving relations among commitments. In *Advances in Cryptology – Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 206–223. Springer Verlag, 2002.

[4] M. Abe and S. Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In *Advances in Cryptology – Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 317–334. Springer Verlag, 2004. (Full version at Cryptology ePrint Archive, Report 2004/118, `http://eprint.iacr.org`, May, 2004.).

[5] M. Abe and H. Imai. Flaws in some robust optimistic mix-nets. In *Australasian Conference on Information Security and Privacy – ACISP2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 39–50. Springer Verlag, 2003.

[6] R. J. Anderson and S. Vaudenay. Minding your p's and q's. In *Advances in Cryptology – Asiacrypt '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 26–35. Springer Verlag, 1996.

[7] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology – Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer Verlag, 2000.

[8] G. Ateniese and G. Tsudik. Some open issues and directions in group signatures. In *Financial Cryptography '99*, volume 1648 of *Lecture Notes in Computer Science*, pages 196–211. Springer Verlag, 1999.

[9] M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. Cryptology ePrint Archive, Report 2003/240, 2003. `http://eprint.iacr.org/`.

[10] R. C. Baker and G. Harman. The difference between consecutive primes. *Proceedings of the London Mathematical Society*, 72(3):261–280, 1996.

[11] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature scheme. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer Verlag, 1997.

[12] O. Baudron, P.A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. Practical multi-candidate election scheme. In *20th ACM Symposium on Principles of Distributed Computing – PODC*, pages 274–283. ACM Press, 2001.

[13] D. Beaver. Foundations of secure interactive computation. In *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer Verlag, 1991.

[14] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

[15] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Advances in Cryptology – Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer Verlag, 1992.

[16] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *SIAM Journal on Computing*, 39(1):214–233, 1992.

[17] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology – Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer Verlag, 2003.

[18] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security (CCS) '93*, pages 62–73. ACM Press, 1993.

[19] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *RSA Conference 2005, Cryptographers' Track 2005*, 2005. (Full version at Cryptology ePrint Archive, Report 2004/077, `http://eprint.iacr.org`, May, 2004.).

[20] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10. ACM Press, 1988.

[21] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *26th ACM Symposium on the Theory of Computing (STOC)*, pages 544–553. ACM Press, 1994.

[22] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *5th ACM Symposium on Principles of Distributed Computing – PODC*, pages 52–62. ACM Press, 1986.

[23] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[24] M. Blum. Coin flipping by telephone. In *Advances in Cryptology – Crypto '81*, pages 11–15, 1981.

[25] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *20th ACM Symposium on the Theory of Computing (STOC)*, pages 103–118. ACM Press, 1988.

[26] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[27] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology – Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[28] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology – Crypto '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 425–439. Springer Verlag, 1997.

[29] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology – Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[30] F. Boudot and J. Traoré. Efficient publicly veriable secret sharing schemes with fast or delayed recovery. In *2nd International Conference on Information and Communication Security (ICICS)*, volume 1726 of *Lecture Notes in Computer Science*, pages 87–102. Springer Verlag, 1999.

[31] E.F. Brickell, D.M. Gordon, K.S. McCurly, and D.B. Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology – Eurocrypt '92*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer Verlag, 1992.

[32] J. Camenisch. Efficient and generalized group signature. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer Verlag, 1997.

[33] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks 2004*, volume 3352 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[34] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[35] J. Camenisch and M. Michels. A group signature scheme with improved effiency. In *Advances in Cryptology – Asiacrypt '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160–174. Springer Verlag, 1999.

[36] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *Advances in Cryptology – Crypto '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 413–430. Springer Verlag, 1999.

[37] J. Camenisch and A. Mityagin. Mix-network with stronger security. Unpublished Manuscript.

[38] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer Verlag, 1997.

[39] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. Springer Verlag, 1997.

[40] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[41] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE Computer Society Press, 2001. (Full version at Cryptology ePrint Archive, Report 2000/067, `http://eprint.iacr.org`, October, 2001.).

[42] R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `http://eprint.iacr.org/`.

[43] R. Canetti, O. Goldreich, and S. Halevi. The random oracle model revisited. In *30th ACM Symposium on the Theory of Computing (STOC)*, pages 209–218. ACM Press, 1998.

[44] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM Symposium on the Theory of Computing (STOC)*, pages 449–503. ACM Press, 2002.

[45] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[46] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19. ACM Press, 1988.

[47] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 470–484. Springer Verlag, 1991.

[48] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology – Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer Verlag, 1991.

[49] L. Chen and T. P. Pedersen. New group signature schemes. In *Advances in Cryptology – Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer Verlag, 1994.

[50] J. Cohen and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–382. IEEE Computer Society Press, 1985.

[51] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Verlag, 1994.

[52] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer Verlag, 1997.

[53] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer Verlag, 1998.

[54] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *6th ACM Conference on Computer and Communications Security (CCS)*, pages 46–51. ACM Press, 1999.

[55] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology – Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer Verlag, 2002.

[56] I. Damgård and J. Groth. Non-interactive and reusable non-malleable commitment schemes. In *35th ACM Symposium on the Theory of Computing (STOC)*, pages 426–437. ACM Press, 2003.

[57] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography – PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer Verlag, 2001.

[58] I. Damgård and J. Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–267. Springer Verlag, 2003.

[59] Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In *Advances in Cryptology – Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer Verlag, 2000.

[60] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[61] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd ACM Symposium on the Theory of Computing (STOC)*, pages 542–552. ACM Press, 1991.

[62] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero-knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[63] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 427–438. IEEE Computer Society Press, 1987.

[64] A. Fiat and A. Shamir. How to prove yourself. practical solutions to identification and signature problems. In *Advances in Cryptology – Crypto '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–189. Springer Verlag, 1986.

[65] P. Fouque and J. Stern. Fully distributed threshold RSA under standard assumptions. Cryptology ePrint Archive, Report 2001/008, 2001. `http://eprint.iacr.org/`.

[66] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology – Auscrypt '92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer Verlag, 1992.

[67] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 1997.

[68] J. Furukawa. Efficient, verifiable shuffle decryption and its requirements of unlinkability. In *Public Key Cryptography – PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 319–332. Springer Verlag, 2004.

[69] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *Financial Cryptography 2002*, volume 2357 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 2002.

[70] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer Verlag, 2001.

[71] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[72] O. Goldreich. *Foundations of Cryptography – Basic Tools*. Cambridge University Press, 2001.

[73] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *21st ACM Symposium on the Theory of Computing (STOC)*, pages 25–32. ACM Press, 1989.

[74] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[75] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology – Crypto '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer Verlag, 1990.

[76] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[77] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[78] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[79] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic mixing for exit-polls. In *Advances in Cryptology – Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 451–465. Springer Verlag, 2002.

[80] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer Verlag, 2003.

[81] J. Groth. Personal communication, 2004.

[82] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. Construction of a pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[83] J. Håstad, A. W. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides $O(n)$ bits. *Journal of Computer and System Sciences*, 47:376–404, 1993.

[84] M. Hirt and K. Sako. Efficient reciept-free voting based on homomorphic encryption. In *Advances in Cryptology – Eurocrypt 2000*, Lecture Notes in Computer Science, pages 539–556. Springer Verlag, 2000.

[85] S. Hamdy J. Buchmann. A survey on IQ cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15. Walter de Gruyter, 2001.

[86] M. Jakobsson. Flash mixing. In *19th ACM Symposium on Principles of Distributed Computing – PODC*, pages 83–89. ACM Press, 1998.

[87] M. Jakobsson. A practical mix. In *Advances in Cryptology – Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 448–461. Springer Verlag, 1998.

[88] M. Jakobsson and A. Juels. Millimix: Mixing in small batches. Technical Report 1999-33, Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), June 1999.

[89] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *Advances in Cryptology – Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer Verlag, 2000.

[90] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. In *20th ACM Symposium on Principles of Distributed Computing – PODC*, pages 284–292. ACM Press, 2001.

[91] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *11th USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[92] M. Jakobsson and D. M'Raihi. Mix-based electronic payments. In *Selected Areas in Cryptography – SAC '98*, volume 1556 of *Lecture Notes in Computer Science*, pages 157–173. Springer Verlag, 1998.

[93] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *Advances in Cryptology – Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[94] A. Kiayias and M. Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders. Cryptology ePrint Archive, Report 2004/076, 2004. `http://eprint.iacr.org/2004/076`.

[95] A. Kiayias and M. Yung. The vector-ballot e-voting approach. In *Financial Cryptography 2004*, volume 3110 of *Lecture Notes in Computer Science*, pages 72–89. Springer Verlag, 2004.

[96] S. Kim, S. Park, and D. Won. Group signatures for hierarchical multigroups. In *Information Security Workshop – ISW '97*, volume 1396 of *Lecture Notes in Computer Science*, pages 273–281. Springer Verlag, 1998.

[97] N. Koblitz. *Algebraic Aspects of Cryptography*. Springer Verlag, 1998.

[98] N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. `http://eprint.iacr.org/`.

[99] C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263. Springer Verlag, 1997.

[100] A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 331–350. Springer Verlag, 2001.

[101] A. Lysyanskaya and Z. Ramzan. Group blind digital signatures: A scalable solution to electronic cash. In *Financial Cryptography '98*, volume 1465 of *Lecture Notes in Computer Science*, pages 184–197. Springer Verlag, 1998.

[102] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[103] R. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1980.

[104] S. Micali, M. Rabin, and J. Kilian. Zero-knowledge sets. In *44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Computer Society Press, 2003.

[105] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, 1988.

[106] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer Verlag, 1991.

[107] M. Michels and P. Horster. Some remarks on a reciept-free and universally verifiable mix-type voting scheme. In *Advances in Cryptology – Asiacrypt '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 125–132. Springer Verlag, 1996.

[108] M. Mitomo and K. Kurosawa. Attack for flash MIX. In *Advances in Cryptology – Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 192–204. Springer Verlag, 2000.

[109] B. Möller. *Public-Key Cryptography – Theory and Practice.* PhD thesis, Technische Universität Darmstadt, 2003.

[110] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *22th ACM Symposium on the Theory of Computing (STOC)*, pages 427–437. ACM Press, 1990.

[111] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM Symposium on the Theory of Computing (STOC)*, pages 427–437. ACM Press, 1990.

[112] A. Neff. A verifiable secret shuffle and its application to e-voting. In *8th ACM Conference on Computer and Communications Security (CCS)*, pages 116–125. ACM Press, 2001.

[113] A. Neff. Verifiable mixing (shuffling) of elgamal pairs. manuscript, 2005. Available at `http://www.votehere.com/documents.html`.

[114] L. Nguyen, R. Safavi-Naini, and K. Kurosawa. A provably secure and efficient verifiable shuffle based on a variant of the paillier cryptosystem. Cryptology ePrint Archive, Report 2005/162, 2005. `http://eprint.iacr.org/`.

[115] L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. Cryptology ePrint Archive, Report 2005/199, 2005. `http://eprint.iacr.org/`.

[116] J. Buus Nielsen. Universally composable zero-knowledge proof of membership. http://www.brics.dk/~buus/, April 2005.

[117] V. Niemi and A. Renvall. How to prevent buying of votes in computer elections. In *Advances in Cryptology – Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 164–170. Springer Verlag, 1994.

[118] A. M. Odlyzko. Discrete logarithms: The past and the future. *Designs, Codes, and Cryptography*, 19(2/3):129–145, 2000.

[119] National Institute of Standards and Technology (NIST). Secure hash standard. Federal Information Processing Standards Publication 180-2, 2002. http://csrc.nist.gov/.

[120] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *1st International Conference on Information and Communication Security (ICICS)*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444. Springer Verlag, 1997.

[121] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Verlag, 1999.

[122] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology – Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer Verlag, 1994.

[123] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[124] K. Peng, C. Boyd, and E. Dawson. Simple and efficient shuffling with provable correctness and ZK privacy. In *Advances in Cryptology – Crypto 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 188–204. Springer Verlag, 2005.

[125] B. Pfitzmann. Breaking an efficient anonymous channel. In *Advances in Cryptology – Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 332–340. Springer Verlag, 1995.

[126] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of mixes. In *Advances in Cryptology – Eurocrypt '89*, volume 434 of *Lecture Notes in Computer Science*, pages 373–381. Springer Verlag, 1990.

[127] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security (CCS)*, pages 245–254. ACM Press, 2000.

[128] The prime pages. `http://primes.utm.edu`, March 2004.

[129] The proth search page. `http://www.prothsearch.net`, March 2004.

[130] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer Verlag, 1991.

[131] P. Ribenboim. *The new book of prime number records.* Springer Verlag, 3 edition, 1996.

[132] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[133] A. Sahai. Non-malleable non-interactive zero-knowledge and adaptive chosen-ciphertext security. In *40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 543–553. IEEE Computer Society Press, 1999.

[134] K. Sako and J. Killian. Reciept-free mix-type voting scheme. In *Advances in Cryptology – Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer Verlag, 1995.

[135] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer Verlag, 2001.

[136] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[137] S. Singh. *The Code Book.* Fourth Estate, London, 1999.

[138] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology – Eurocrypt '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 190–199. Springer Verlag, 1996.

[139] D. Stinson. A polemic on notions of cryptographic security. Manuscript, 2004. `http://www.cacr.math.uwaterloo.ca/~dstinson`.

[140] M. Trolin and D. Wikström. Hierarchical group signatures. Cryptology ePrint Archive, Report 2004/311, 2004. `http://eprint.iacr.org/`.

[141] M. Trolin and D. Wikström. Hierarchical group signatures. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 446–458. Springer Verlag, 2005. (Full version [140]).

[142] Y. Tsiounis and M. Yung. On the security of elgamal based encryption. In *Public Key Cryptography – PKC '98*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer Verlag, 1998.

[143] D. Wagner. A generalized birthday problem. In *Advances in Cryptology – Crypto 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer Verlag, 2002.

[144] D. Wikström. How to break, fix, and optimize "optimistic mix for exit-polls". Technical Report, Swedish Institute of Computer Science (SICS), T2002:24, ISSN 1100-3154, ISRN SICS-T-2002/24-SE, 6 December 2002, `http://www.sics.se`.

[145] D. Wikström. On the security of mix-nets and related problems. Licentiate thesis, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, TRITA-NA-04-06, ISSN 0348-2952, ISRN KTH/NA/R--04/06--SE, ISBN 91-7283-717-9, May, 2004, `http://www.kth.se`.

[146] D. Wikström. Five practical attacks for "optimistic mixing for exit-polls". In *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 160–174. Springer Verlag, 2003.

[147] D. Wikström. A universally composable mix-net. In *1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 315–335. Springer Verlag, 2004.

[148] D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 273–292. Springer Verlag, 2005. (Full version [149]).

[149] D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. Cryptology ePrint Archive, Report 2004/137, 2005. `http://eprint.iacr.org/`.

[150] D. Wikström and J. Groth. An adaptively secure mix-net without erasures. submitted manuscript.

[151] A. C. Yao. Theory and application of trapdoor functions. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Computer Society Press, 1982.

[152] A. Young and M. Yung. Finding length-3 positive Cunningham chains and their cryptographic significance. In *Algorithmic Number Theory – ANTS- III*, volume 1423 of *Lecture Notes in Computer Science*, pages 289–298. Springer Verlag, 1998.

Appendix A

Probability Bounds

Theorem A.1 (Markov). *Let X be a random variable taking non-negative values. Then*

$$\Pr[X \geq a] \leq \frac{\mathrm{E}[X]}{a} \ ,$$

where $\mathrm{E}[X]$ is the expected value of X.

Theorem A.2 (Chernoff). *Let $X_1, \ldots, X_N$ be mutually independent indicator variables and define $X = \sum_{i=1}^{N} X_i$. Then for arbitrary $\gamma > 0$ we have:*

$$\Pr[X < \mathrm{E}[X] - \gamma N] < e^{-2\gamma^2 N} \ .$$

Theorem A.3 (Chernoff). *Let $X_1, \ldots, X_N$ be mutually independent indicator variables and define $X = \sum_{i=1}^{N} X_i$. Then for arbitrary $\gamma > 0$ we have:*

$$\Pr[X < (1-\gamma)\mathrm{E}[X]] < e^{-\frac{\gamma^2 \mathrm{E}[X]}{2}} \ .$$