KTH Computer Science
and Communication

Electronic Cash and Hierarchical Group Signatures

MÅRTEN TROLIN

Doctoral Thesis
Stockholm, Sweden 2006

TRITA-CSC-A 2006:24
ISSN 1653-5723
ISRN KTH/CSC/A–06/24–SE
ISBN 91-7178-510-8

School of Computer Science and Communication
KTH
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi den 15 december kl 14:00 i D3, Lindstedtsvägen 5, Kungl Tekniska högskolan, Stockholm.

Tryck: Universitetsservice US AB

Abstract

In this thesis we present results in two areas, electronic cash and hierarchical group signatures.

- We investigate definitions of security for previously proposed schemes for electronic cash and strengthen them so that the bank does need to be trusted to the same extent. We give an experiment-based definition for our stronger notion and show that they imply security in the framework for Universal Composability. Finally we propose a scheme secure under our definition in the common reference string (CRS) model if based on a family of trapdoor permutations.

  As a tool we define and prove the existence of simulation-sound non-interactive zero-knowledge proofs (NIZK-PK) in the CRS-model under the assumption that a family of trapdoor permutations exists.

- We propose a scheme for electronic cash based on symmetric primitives. The scheme is secure in the framework for universal composability if based on a symmetric CCA2-secure encryption scheme, a CMA-secure signature scheme, and a family of one-way, collision-free hash functions. In particular, the security proof is *not* in the random-oracle model. Due to its high efficiency, the scheme is well-suited for devices such as smart-cards and mobile phones. We also show how the proposed scheme can be used as a group signature scheme with one-time keys.

- We introduce the notion of *hierarchical group signatures*. This is a proper generalization of group signatures, which allows multiple group managers organized in a tree with the signers as leaves. For a signer that is a leaf of the subtree of a group manager, the group manager learns which of its children that (perhaps indirectly) manages the signer. We give definitions and three different constructions.

  Our first construction uses general methods and is secure if based an a family of trapdoor permutations. It is not intended for actual use, but serves as a proof of concept of our definition. Our second construction is almost practical, and we prove its security in the random oracle model under the strong RSA assumption and the DDH assumption. Our third construction is practical and secure under the strong RSA assumption and the DDH assumption in the random oracle model. The third construction is optimistic in that a group manager may need to interact with other parties to open a signature of a corrupt signer.

  Finally we show that if a hierarchical group signature scheme is secure, then it realizes an ideal hierarchical group signature scheme in the framework for universal composability.

Sammanfattning

I denna avhandling presenteras resultat inom två områden – elektroniska betalningar och hierarkiska gruppsignaturer.

- En starkare säkerhetsdefinition för digitala pengar presenteras med syfte att användare inte ska behöva lita på banken i samma utsträckning som för äldre definitioner. Denna starkare definition ges dels som en experimentbaserad definition och dels som en definition i ramverket för universell sammansättning (UC). Det visas att säkerhet enligt den experimentbaserade definitionen implicerar UC-säkerhet. Dessutom beskrivs en konstruktion med allmänna metoder som är säker enligt den nya definitionen i modellen med gemensam referenssträng (CRS) om den baseras på en familj lönndörrspermutationer.

  I samband med konstruktionen definieras extraherbara, simuleringssunda, icke-interaktiva kunskapslösa bevis. Existensen av sådana bevis i CRS-modellen bevisas under antagandet att det existerar en familj lönndörrspermutationer.

- Ett system för digitala pengar beskrivs. Systemet bygger på symmetriska primitiv och är säkert i UC-ramverket under antagandet att ett symmetriskt CCA2-säkert kryptosystem, ett CMA-säkert signaturssystem och en familj enkelriktade kollisionsfria hashfunktioner används. Beviset är *inte* i slumporakel-modellen. Tack vare sin höga effektivitet är system väl lämpat för exempelvis smartkort och mobiltelefoner. Det beskrivs vidare hur systemet kan användas som ett system för gruppsignaturer med engångsnycklar.

- Begreppet *hierarkiska gruppsignaturer* introduceras. Detta är en äkta generalisering av gruppsignaturer som tillåter flera gruppchefer organiserade i ett träd med signatörerna som löv. Om en signatör är löv i ett delträd under en gruppchef får denne gruppchef reda på till vilket delträd signatören hör. Definitioner och tre olika konstruktioner ges.

  Den första konstruktionen använder allmänna metoder och är säker om den baseras på en familj lönndörrspermutationer. Denna konstruktion är inte avsedd för praktisk använding, utan ska ses som ett bevis att definitionen är rimlig. Den andra definitionen är nästan praktisk, och dess säkerhet bevisas i slumporakelmodellen under starka RSA-antagandet och DDH-antagandet. Den tredje konstruktionen är praktisk och säker under starka RSA-antagandet och DDH-antagandet. Denna konstruktion är optimistisk i den meningen att en gruppchef kan behöva interagera med andra parter för att öppna en signatur.

  Slutligen bevisas att om ett system för hierarkiska gruppsignaturer är säkert så realiserar det en ideal funktionalitet i UC-ramverket.

Contents

Part I

Background

Chapter 1

Introduction

1.1 About this Thesis

Historically cryptography was only about sending secret messages. The Caesarian cipher, where each letter of the message is replaced the letter three positions ahead in the alphabet, is often mentioned as the first known use of cryptography. Although a quite simple scheme, one can assume it was quite effective at the time, when most people were illiterate and the message, if at all read, would be assumed to be in a foreign language. Other examples can be found in fiction. One example is the secret language of [60] in one the early works by Lindgren.

These schemes have in common that, once the underlying idea is known, they are quite easy to break. One could argue that they are more about obscurity than security. As cryptography has evolved, modern schemes are designed to be secure even if the method is known to an attacker.

Organization of the Thesis

Part I gives the background. Chapter 1 gives a relatively non-technical introduction to the subject. In Chapter 2 we provide definitions of concepts and primitives, almost all of which are well-known from literature. Note that some definitions which are only used locally can be found later in the thesis. A summary of our results is found in Chapter 3.

Our results on electronic cash are presented in Part II. Chapter 5 is based on an unpublished manuscript and Chapter 6 is based on [83].

The contributions on hierarchical group signatures are found in Part III, where parts of Chapter 8 as well as Chapters 9 and 10 are based [84]. The rest of the Chapter is based on an unpublished manuscript. The results of this part are joint work with Douglas Wikström.

How to Read the Thesis

If you only want an introduction to the subject, it suffices to read Chapter 1. If you also want a short description of the results, you can read Chapter 3, possibly using Chapter 2 to look up standard definitions. If you need the detailed descriptions, including the proofs, you will have to read also Parts II and III.

1.2 Confidentiality and Authenticity

Hiding information from eavesdroppers, *confidentiality*, is the traditional reason to use cryptography. An analogy is to send a message in a sealed envelope (or maybe in a locked safe, although it is debatable how realistic such an analogy is). Sometimes we are not primarily interested in hiding information, but rather in ensuring that information isn't modified or counterfeited, *authenticity*. By this we mean that the receiver can be convinced that sender is who he claims to be, and that the message has not been altered in transit. The analogy here is to sign a paper with the message on it. Since signatures are assumed to be hard to forge, a signature identifies the sender.

In an environment where messages are mainly sent electronically, we need methods to achieve confidentiality and authenticity by digital means, and this is one major part of what cryptographic research is about. The traditional approach is to set up a key sk and define a function E to encrypt and a function D to decrypt so that $D_{sk}(E_{sk}(\mathsf{msg})) = \mathsf{msg}$ for any legal message msg. We will call msg the *plaintext* and the encryption $E_{sk}(m)$ the *ciphertext*. Since we want the system to be secure, we want it to be infeasible to compute any useful information about the plaintext from the ciphertext, provided that the key sk is unknown.

It is quite reasonable to assume that the attacker has more information than just the ciphertext. The attacker may have seen ciphertexts for which she knows the plaintext. For example, every plaintext may begin with the same standard header. It may even be the case that the attacker can trick the the sender into encrypting plaintexts of the attacker's choice, or decrypting ciphertexts chosen by the attacker. The attacker may also know a subset of legal messages from which msg is drawn, Depending on the scenario we may require the encryption scheme to remain secure also for such settings,

Consider the functions necessary to ensure that a message isn't counterfeited or modified. The usual approach is to define a function S to create a *message authentication code* (MAC) and a function V to verify that a MAC is valid. The function S takes as input a message msg and a key sk and returns a MAC. The function V takes a key, a message and a MAC, and returns 1 if the MAC is valid and 0 otherwise. It must hold that $V_{sk}(m, S_{sk}(m)) = 1$, and it should be infeasible to compute a message m and a MAC s such that $V_{sk}(m, s) = 1$ without knowledge of sk. Also here the attacker may have access to side information such as MACs on messages of his choice.

1.3 Public Key Cryptography

Asymmetric Encryption Schemes

In the above definitions, the same key is used for encryption and decryption. For a long time, this was the only known way to perform cryptography. In the middle of the 1970s, a major breakthrough was made when methods to perform *asymmetric* cryptography were discovered. Asymmetric schemes use two keys, the *public key*, *pk* and the *secret key* (sometimes called *private key*), *sk*. The public key is used to encrypt, and the private key to decrypt so that $D_{sk}\left(E_{pk}(m)\right) = m$. The public key can be published, since it is used only for encryption, but the private key must be kept secret.

Let us now compare this with symmetric encryption schemes to see what the differences may mean in practice. Assume ten people work at the same company, and that they want to be able to send encrypted messages to each other. First consider a symmetric encryption scheme. One solution is to have a single common key that everything is encrypted with, but there are several drawbacks with this approach. Someone who gets hold of the key (for example by bribing one of the employees) is able to read all messages sent. Also any employee can read any message, even it wasn't meant for him. If an employee quits, a new key has to be set up and distributed in a secure manner. A second solution is to have one key between every pair of employees. Then only the intended recipient can read his messages, and if one employees sells (or accidently discloses) his keys, only the messages sent or received by that employee can be read. However, the number of keys necessary for such a system is high. Our ten employees need a total of 45 keys. Although this number may not seem very high, we must take into account that agreeing on a symmetric key is a cumbersome task. It is not advisable to send the keys electronically, since they can be eavesdropped, and if a key is sent by mail, there is always the risk that someone opens the envelope and gets the key. The only safe way is to meet in person. Now consider a company with 1000 employees. Then a total of $499,500$ keys are necessary! It is obvious that symmetric encryption schemes have certain drawbacks.

Let us consider using asymmetric cryptography to solve the problem. Each of the ten employees generates a key pair consisting of a secret and a public key. The public keys are published, say in the company phone book. If Alice wants to send a message to Bob, she looks up Bob in the phone book, encrypts using his public key and sends the message. Bob uses his secret key to decrypt, and no-one else can read the message. If the company hires new employees, then each of them generates a key pair. No keys have to be exchanged under secure conditions.

Digital Signatures

Also authenticity can be achieved by asymmetric means. When a MAC is used, the same key is used for computing the MAC and verifying it. Hence only the intended recipient can check the validity of the message. Furthermore, ability to verify implies ability to compute a MAC, making it hard to use a signature as proof in case of a dispute. Therefore, in many situations, it is desirable to have a scheme in which it is possible to verify without being able to sign. Using asymmetric techniques we can construct a scheme where the *signing* is performed using the secret key *sk* and the *verification* with the public key *pk*. Now it must hold that $V_{pk}\left(m, S_{sk}(m)\right) = 1$. This is also what we expect from real-world signing schemes – anyone can look at a signature and check whether it has been written by the putative sender (by comparing it with other signatures written by the same person), but no-one but the sender else should be able to produce such a signature.

A digital signature is in one sense more secure than a physical signature on paper. When a paper with the message written on it is signed, it is hard to ensure that the message is not altered afterwards. A forger may add new text to a signed document or combine pages from two or more signed documents into a new document. A secure digital signature scheme withstands attacks of this type, since the signature is tied to the message and becomes invalid if the message is modified.

1.4 Zero-Knowledge Proofs

In some cases a player, which we call the *prover*, needs to prove to a different player, the *verifier*, that she knows a certain secret such as a secret key or a password. The most obvious way to do this is by simply passing the secret to the verifier. This would be a *proof system*. Often it is not permissible to pass the secret to the verifier, since the verifier could use the secret himself. What we need is a *zero-knowledge* proof system, which convinces the verifier without exposing the actual secret.

A general proof system may require communication between the prover and the verifier. A non-interactive proof system is a proof system where the communication consists of only a single message from the prover to the verifier. Clearly there is no need for a direct link between the two parties for such a protocol.

It may seem counter-intuitive that knowledge of a secret can be proved using a non-interactive protocol without actually revealing the secret, but it is known that, informally speaking, such a non-interactive zero-knowledge proof exists for any type of (computational) secret.

We often write NIZK instead of non-interactive zero-knowledge proof.

1.5 One-Way Functions and Trapdoor Functions

Two of the most important building blocks for cryptographic functions are *one-way functions*, i.e., functions that are easy to compute but hard to invert, *trapdoor functions*, i.e., functions that are one-way functions with the additional property that there is a secret which makes the function easy to invert. Take, for example, multiplication. It is easy to multiply two numbers, but no method is known that factors a numbers into its prime factors in reasonable time. It should be noted that the existence of one-way and trapdoor functions is a classical open problem, and a proof of their existence would be a major breakthrough. However, there are functions that have been subject to intensive research for more than thirty years, and no evidence contradicting the hypothesis that they are trapdoor functions have been found. It is therefore reasonable to assume that they are indeed trapdoor functions. From functions that are assumed to be trapdoor functions, it is possible to build cryptographic primitives, e.g., encryption and signature schemes.

1.6 Building Cryptographic Protocols

To achieve more complex tasks, such as setting up a secure channel between parties who have not previously met, or creating digital coins, we need to describe how to combine primitives to get the functionality we need. The result is called a protocol, and the protocol describes how the participants should act. A protocol can be seen as a set of algorithms, one for each participant.

A protocol may be interactive or non-interactive. An interactive protocol is used when the parties can send messages to each other in an interleaved manner. An example may be a user logging on to a web-site. In a non-interactive protocol the sender creates the message on his own, and only then sends it to the receiver. Encrypting and signing emails are a typical examples of non-interactive protocols.

1.7 Efficient vs. Practical Protocols

Naturally we want our protocols to be as efficient as possible. However, in different contexts effiency may have different meanings. The common definition of an efficient algorithm is that the execution time is bounded by a polynomial in the size of the input. For example, the grade school algorithm for multiplication is polynomial time, since the number of steps needed is less than $2n^2$, where n is the number of digits of each factor.

An example of an algorithm that is not polynomial is factoring by exhaustive search. To factor an n-bit number m we may need to check each number up to $\sqrt{m}$, that is, $2^{n/2}$ different numbers. Even if we assume that we can check divisibility in a single step, we still need an exponential number of steps before we are guaranteed to have a result.

It is clear that this definition of efficient algorithms does not cover everything we need from an algorithm to be usable in practice. If we design an algorithm that runs in n^{30} steps, it would still be considered efficient according to the above definition. However, the algorithm would be impossible to use in practice except for extremely small inputs.

The protocols in this thesis fall into one of two categories. The first category are protocols which uses general methods, such as the existence of zero-knowledge proofs. Such protocols should only be viewed as evidence that a certain kind of protocols exist, and may give some insight into how a practical protocol may be constructed. The second category are protocols with an explicit description. These protocols must be specified in such detail that it is possible to analyze their running time precisely and not only show that it is bounded by some polynomial. Being practical is not a strict definition. In some cases, we want a protocol that can be executed on devices with little computing power such as smart-cards or mobile phones. In other cases it is enough if the protocol runs reasonably fast on a personal computer, and in still other cases the protocol will run on a server with large storage capabilities.

1.8 Optimistic Protocols

We would prefer protocol that performs well all of the time. It may be the case that we do not succeed in constructing such a protocol. Then we may settle for a protocol which performs well if all parties are honest, but may perform worse, while still being secure, if there are dishonest parties. The protocol may need to query an external trusted party when inconstistencies are detected. Such protocols are called *optimistic* for obvious reasons.

1.9 Security of Cryptographic Primitives and Protocols

Obviously we want the cryptographic primitives we use to be secure. However, we need to define precisely what we mean by security of a primitive. Let us consider an encryption scheme. One definition of security is that the scheme is secure if an attacker who sees a ciphertext cannot recover the plaintext. However, in some scenarios this is not enough, since the attacker may have access to additional information. Maybe the attacker knows that the plaintext is either “yes” or “no”, and maybe the attacker has seen encryptions of other plaintexts. Maybe the attacker even has seen encryptions of “yes” and “no”. A good encryption scheme should remain secure even under these circumstances. For example, to remain secure even if the attacker knows encryptions of “yes” and “no”, the encryption must be probabilistic, so that when the same plaintext gives different ciphertexts when encrypted several times.

Designing protocols that are as secure as the primitives used is not trivial. It may very well be the case that a protocol turns out to be insecure although all components used are secure. Also in the case of protocols, the term “secure” must be properly defined. Take, for example, a scheme for electronic cash involving customers, merchants and a bank. Naturally a customer should not be able to counterfeit money, but what happens if a customer and a merchant collaborates to forge electronic coins? Or maybe when two customers together try to create a coin that appears to be valid to the merchant but which is rejected by the bank? Obviously there are many subtle details when deciding what kind of security we want from a protocol. Therefore it is important to make a clear definition of security and to prove that the protocol fulfils those definition under some plausible assumptions.

Computational Assumptions

It is seldom possible to construct schemes that are unconditionally secure. The most common approach is to design a scheme so that it is secure if some well-examined problem is hard to solve. The assumptions which we use in this thesis are the *strong RSA assumption* and the *decisional Diffie-Hellman assumption.* Informally the strong RSA assumption says that, given an RSA modulus $\mathbf{N}$, i.e., a product of two primes, and $\mathbf{g} \in \mathbb{Z}_{\mathbf{N}}$, it is hard to find a number $e > 1$ and $\mathbf{t}$ such that $\mathbf{t}^e = \mathbf{g} \bmod \mathbf{N}$. Put differently, it is assumed to be hard to draw any root of an element of the group $\mathbb{Z}_{\mathbf{N}}$.

The decisional Diffie-Hellman assumption is about the following problem. Consider a prime q, and let g be a generator for the multiplicative group $\mathbb{Z}_q$. Suppose we are given a tupel (a, b, c) which have either been constructed by $a = g^x$, $b = g^y$, $c = g^{xy}$, or a, b, c can be random variables. The task is to decide which of the two cases holds. The decisional Diffie-Hellman assumption states that it is hard to solve this problem.

Experiment-Based Security vs. Simulation-Based Security

Let us consider a scheme for digital signature, and what we would expect from such a scheme in terms of security. One way to describe the desired properties is the following. An adversary attacking a signature scheme is successful if it constructs a signature without having access to the private key. It is now possible to construct an experiment to test whether an adversary is indeed successful. The experiment gives some information to the adversary which it would have access to in a real-world scenario. In the case of a signature scheme this would be the public key. The experiment may also allow the adversary to access some kind of additional information. It may, as an example, be allowed to request signatures on certain messages. Then the adversary outputs a message-signature, and it wins if the signature is valid and no signature on the message has been given to the adversary by the experiment.

If the scheme is secure, then there cannot exist an efficient adversary which wins the experiment, other than possibly with very low probability. To prove this, one assumes the existence of such an adversary, and shows that such an adversary can be used to perform some task which is assumed to be impossible.

A different approach is to define how a signature scheme would behave if we had access to a completely honest party, called the *ideal functionality*, with which everyone could communicate freely. To construct signatures, the ideal functionality would verify that the player requesting the signature is allowed to do this, and if so, output a string representing the signature. To verify a signature, the message and signature is sent to the ideal functionality. If the trusted party has produced the signature, then it answers that the signature is valid, and otherwise it responds that it is invalid. Intuitively this is what one would expect from a signature scheme.

Now we can define a signature scheme as secure if it is indistinguishable from the ideal signature scheme. To prove that a specific scheme fulfills this definition of security, we assume there exists an environment which interacts with either the real protocol or the ideal functionality, and which is able to determine which one it interacts with. Then we show how to use such an environment to break some assumption.

1.10 Anonymity

Assume the cash you withdraw had your name on it. What would that mean? In most cases it would not mean anything. No-one would be interested in knowing that it was you who bought that pack of chewing gum. You might feel a little bit uncomfortable if you knew that a curious trainee working in the pharmacy can keep track of what medicine you use. If the government can figure out your political viewpoint by monitoring what newspapers you purchase and what events you buy tickets to, you have reason to be really worried.

We often take anonymity for granted. If you purchase a newspaper with cash, it is not possible to trace the purchase back to you by looking at the coins you paid with. If you buy a couple of tokens for the metro, it is not possible to see if two trips were paid by tokens purchased at the same time. The simple reason neither coins nor metro tokens are traceable is that they do not have a serial number. The reason they do not have a serial number is that their low value do not make them an interesting target for counterfeiter – the cost of producing a fake coin or metro token probably exceeds its value.

Now you may argue that these transactions are not at all anonymous – if you go and buy the newspaper in person, anyone can see what you bought. However, the important point here is that it requires considerable resources to track a person that way, and it is impossible to do in an automated way on a large scale.

When the physical coins and metro tokens are replaced with electronic counterparts, the scenario is changed. The cost of copying an electronic coin, which is nothing but a sequence of zeros and ones, is next to nothing. Therefore even low-value coins need some kind of serial number to detect duplicates, and that potentially makes them traceable. One of the challenges when designing protocols for transactions that people assume to be anonymous is to make them anonymous also when performed electronically.

Before we can design anonymous protocols, we must decide what we mean by anonymity. One definition of anonymity is that a transaction cannot be connected to the identity of any involved party. This definition, however, is weaker than the anonymity of real-world transactions, because it does not say anything about connecting transactions. Assume, for example, that you use your electronic coins first to buy a train ticket that is mailed to your home and then to buy a political newsletter. If the coins are anonymous only in the above sense, the identity of the buyer of the newsletter may still be revealed if the two purchases can be connected. Clearly it is unlinkability that we should strive for.

If a protocol involves several parties, in the case of electronic coins a customer, a merchant and the bank, we may settle for anonymity only towards the merchant to make the protocol more efficient. In other words, the merchant cannot link two purchases, but once the coin reaches the bank, the bank can see who withdrew the coin. Another concept is *revocable anonymity.* Here some trusted third party (who could, for example, be a judge) can extract the identity from a coin, but otherwise the coin is anonymous to both the bank and the vendor.

Although anonymity is desirable from the user's point of view, protocols that ensure anonymity tend to be less efficient than non-anonymous protocols. Also from a legal point of view anonymity might be problematic. If electronic coins are achieved through black-mailing or other illegal activities, anonymity works in favor of the criminal.

In an anonymous scheme for electronic coins the bank cannot monitor the flow of coins. It will detect irregularities only after a long period of time (if ever). This may be one reason why the schemes for electronic cash that are in use are non-anonymous.

1.11 Payment Systems

When making purchases, the most common ways to pay for the goods is either by cash or by a payment card or check. Cash has the property that it is anonymous and that it is possible to verify that it is valid by just looking at it and without calling the bank. This *offline* property of cash is important, and very desirable. It reduces communication costs, it makes the scheme more robust since it doesn't require the bank to be available, and it is fast. The merchant can deposit the cash with his bank, use it as change, buy goods, pay salary etc. Unfortunately cash also has the not so nice property that it can be stolen. A payment card or check, on the other hand, is not itself a proof that the customer has the money to pay. The issuer must be contacted to verify that the customer has the necessary funds, but once the transaction is completed, it cannot be stolen like cash. Since the merchant's name is part of the payment, no-one else can get credited for the transaction.

Digital payment systems try to mimic these properties. Systems for digital cash try to keep the anonymity of the customer, possibly with a trusted party that can revoke the anonymity. However, since a digital coin is just a bit-string, it can be copied and spent twice. The most common way to deal with this is to design the system so that the identity of the owner is revealed if the same coin is spent twice. Another solution is to make the system online, but then part of the motivation to use coins is lost.

Systems for digital cash often require that the merchant deposits the cash with the bank after the transaction rather than reuse it. However, digital cash may also have the useful property that in cannot be stolen while at the merchant, since the merchant's name is part of the transaction.

If digital cash does not completely correspond to cash in the real world, payment card transactions are easier to make purely electronic. In many cases this simply means that the physical signature on the receipt is replaced by a digital signature by the cardholder. Here, however, we can ask for more and make payment card transactions anonymous to the merchant.

The goal then is to design a system such that two transactions cannot be linked by the merchants. The system will still be non-anonymous to the issuer, since the issuer must be able to charge the correct account. A trivial way to achieve anonymity towards the merchant is to give each cardholder not just one card number, but several one-time numbers. The bank keeps a list of which number belongs to which cardholder, and the cardholder makes sure each number is only used once. Provided that the card numbers are generated randomly, such a system would be anonymous to the merchants.

1.12 Group Signatures

In this section we discuss a more general approach to the problem of creating anonymous credit cards. We use *group signatures*. In a group signature scheme, there are *group members* and a *group manager*. Group members can sign documents on behalf of the group, but the only information that someone other than the group manager gets is that *someone* in the group signed the document. The group manager, however, is able to determine the identity of a signer. As the alert reader has already seen, this is exactly what we need to make payment cards anonymous. The group members are the cardholders, and the issuer is the group manager. When making a payment, the cardholder produces a group signature on the transaction. The merchant verifies that the signature is produced by someone in the group of cardholders, but does not get any additional information. When the transaction is passed on to the card issuer, the issuer, who acts as group manager, extracts the identity of the cardholder to debit the correct account.

The scheme described above with group signatures works for payment cards when there is just one issuer, and every merchant sends all transactions directly to that issuer. In reality this is not the case. There is not just one but several issuers cooperating within a network. Rather than sending the transaction directly to the issuer, the merchant sends it to the network, which routes it to the issuer.

The obvious way to solve the problem is to set up a group signature scheme for each issuer. With this solution we lose some anonymity, since the merchant learns the name of the issuer, and in some cases this can give quite a lot of information. Therefore we would like a variant of group signatures where there are group managers that only get partial information about the identity of the signer. More specifically, in the case of payment cards, we need a scheme such that the signature is anonymous to the merchant, the network can see which issuer the card belongs to, and the issuer sees the identity of the cardholder. Naturally this can be generalized so that there are several intermediate group managers that get more and more detailed information about the identity. In this thesis we describe such an extension of group signatures. Because of the hierarchical way information about the identity is revealed, we call the scheme *hierarchical group signatures*.

1.13 Acknowledgments

Now that the scientific part of the thesis is ready, the hardest part remains, namely to remember all the people who helped me finish the thesis. Many of my colleagues have helped me proof-read manuscripts, have listened to and commented on seminars, or have taken time to discuss topics related or unrelated to my research. Although it is impossible to mention everyone by name, I will do my best.

First of all I would like to thank my advisor Johan Håstad for all the help and excellent ideas. Without this support, my work had not been possible. I would also like to thank Mikael Goldmann for good advice and useful tips. I have had many interesting and rewarding discussions with my fellow students. These discussions have often given me new insights and new perspective on my work. I would like to thank Gustav Hast for valuable discussions on the protocol for electronic cash as well on some of the ideas behind the group signature scheme. Of course Douglas Wikström, with whom I co-authored the paper that is the basis for the chapter on group signatures, deserves a special acknowledgment.

Part of the thesis was written while I was visiting University of Latvia in Riga. I am very grateful to prof Jānis Bārzdiņš and prof Rūsiņš Freivalds, who made this visit possible.

Chapter 2

Notation and Definitions

2.1 Basic Notation

We write $[a, b]$ to denote the set $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$ or the set $\{x \mid a \leq x \leq b\}$. In each case the meaning will be clear from the context. We use $SQ_{\mathbf{N}}$ to denote the subgroup of squares in $\mathbb{Z}_{\mathbf{N}}^*$. Elements in $SQ_{\mathbf{N}}$ are written in bold-face, e.g., $\mathbf{z} \in SQ_{\mathbf{N}}$. We write $\emptyset$ to denote both the empty set and the empty string. 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.

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 $\mathbb{Z}_q$.

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 $\lfloor \log_2 q \rfloor$, whichever 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.

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.

Let T be a tree with root ω. We denote by $\mathcal{L}(T)$ its set of leaves. Abusing notation we let T be both the tree and the set of nodes. By $\mathsf{level}_T(v)$ we denote the level of v in the tree T where the root is on level 0.

By $r \leftarrow_R S$ we mean that r is chosen randomly in S. A function $f : \mathbb{N} \rightarrow [0, 1]$

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 $\kappa_0 < \kappa \in \mathbb{N}$. We say that a function $f : \mathbb{N} \rightarrow [0, 1]$ is non-negligible whenever it is not negligible, and we say that a function f is overwhelming if $1 - f(\kappa)$ is negligible. When we say that a number is k-bit, we implicitly mean that it has a leading one (i.e., that it is in the interval $[2^{k-1}, 2^k - 1]$). We write $\lceil r \rceil$ to denote the smallest integer $n \geq r$ and we write $\lfloor r \rfloor$ to denote the largest integer $n \leq r$.

All adversaries in this paper are modeled as polynomial time Turing machines with *non-uniform* auxiliary advice string. We denote the set of such adversaries by PT^*.

Given two distributions X and Y the statistical distance between X and Y is defined as $\mathsf{dist}(X, Y) = \sum_\alpha |\Pr[X = \alpha] - \Pr[Y = \alpha]|$. Two ensembles $\{X_n\}_{n \in \mathbb{N}}$ and $\{Y_n\}_{n \in \mathbb{N}}$ are *statistically close* if $\mathsf{dist}(X_n, Y_n)$ is negligible in n.

We say that a distribution ensamble $\mathcal{D} = (D_i)_{i=1}^{\infty}$ is efficiently sampleable if there exists a probabilistic polynomial time algorithm $T_{\mathcal{D}}(1^i)$ that outputs a random sample distributed according to D_i.

2.2 Security Models

In this thesis we use both the classical experiment-based notion of security and simulation-based security. In experiment-based definitions a number of experiments are defined, and the scheme is said to be secure if no adversary can win in any of the experiments. For simulation-based definitions we use the framework for universal composabilty to give security results.

The Framework for Universal Composability

A detailed description of the framework for universal composability, UC-framework, is given by Canetti in [30]. For completeness we here give a brief summary.

A protocol is described by giving an *ideal functionality*, which is an interactive Turing machine. The players of the ideal protocol may send messages to the ideal functionality, and the ideal functionality may send messages to the players, but no other communcation is allowed. All commmunication is assumed to take place over secure and authenticated channels. There is an adversary, called the ideal adversary, or simulator, denoted $\mathcal{S}$, which may corrupt players and delay messages.

A function is either immediate or non-immediate, where immediate functions are calls to the functionality which are immediately followed by a response sent to the calling party. One can think of immediate functions as local computations, and of non-immediate functions as interactive protocols. It will be stated for each ideal functionality which functions are immediate.

An implementation of a protocol is a graph of interactive Turing machines, which communicate over secure and authenticated channels. The adversary A may corrupt players and learn their internal state and control their input and output, and it may delay messages. If we only allow the adversary to corrupt players before the execution is initiated, we say that the adversary is *static*.

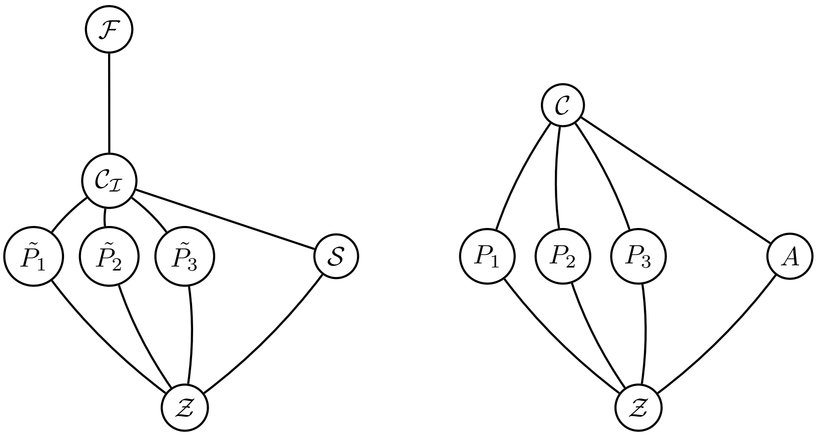


Figure 2.1: The environment $\mathcal{Z}$ interacting with the ideal protocol (to the left) and with the real protocol (to the right).

Security is defined by introducing an *environment* $\mathcal{Z}$, whose task is to distinguish between the ideal and the real protocol. The ideal protocol is run with dummy players, which do nothing but forward messages from the environment to the functionality and back. The environment is allowed to send arbitrary messages to the players and the adversary. It may instruct the adversary to corrupt players. Its goal is to decide whether it runs with the ideal functionality or the practical protocol. We say that a protocol *securely realizes* an ideal functionaly if for each real adversary A there exists an ideal adversary $\mathcal{S}(A)$ such that no environment can tell whether it runs with the real or ideal protocol.

As the name suggests, one of the powers of the framework for universal composability is that protocols can be combined so that the new protocols is secure if the building blocks are secure. Let us be a little more precise. A hybrid model is a real protocol with access to an ideal functionality. Informally the *composition theorem* says that if a protocol π securely realizes a functionality $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model, and π' securely realizes $\mathcal{F}'$, then π securely realizes $\mathcal{F}$ also if $\mathcal{F}'$ is replaced by π'.

We use tilde-notation for dummy players, i.e., $\tilde{P}_1$.

We use a model where the ideal functionality is linked to the players through an ideal communication network $\mathcal{C}_\mathcal{I}$. The communication network forwards a message msg from a player P as (P, msg) to the ideal functionality. When $\mathcal{C}_\mathcal{I}$ receives (P, msg) from the functionality for a non-immediate function, it informs the ideal adversary that P has been sent a message, and does not forward it until $\mathcal{S}$ has approved. The message itself it never forwarded to $\mathcal{S}$. Hence the ideal adversary is allowed to delay the delivery of such a message, but not change its content. For immediate functions the response is forwarded to P without involving the ideal adversary.

The use of a communication network replaces the need for session identifiers of [30], since each protocol uses its own communication network. The difference is purely formal, but it makes the description more simple.

The Common Reference String (CRS) model

It is often hard to construct a provably secure scheme without any setup assumption. One such assumption is that every player has access to a common string, which is called the Common Reference String, or CRS. In an honest execution of the protocol, the string is chosen randomly before the protocol is initiated.

Let us briefly describe the proof technique for a scheme in the CRS model. Let A be an adversary breaking some security property of the scheme. By constructing the CRS such that some trapdoor is known, we may be able to use A to solve some presumebly hard problem. It is important that the CRS created in such a way has a distribution which is identical, or almost identical, to an honestly created CRS.

The Random Oracle Model

Experiments 2.4.1 and 2.4.2 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 practical 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.

The random oracle hypothesis, first explicitly stated in a paper by Bellare and Rogaway [10], 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 [32] 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 [32] 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.

If a scheme is secure in the CRS model, then it is also secure in the random oracle model, since we can use $H(0), H(1), \ldots$ as the common reference string, where H is the hash function modeled as a random oracle.

2.3 Computational Assumptions

In this section we formalize the Decisional Diffie-Hellman assumption and the strong RSA assumption.

The Decisional Diffie-Hellman Assumption

The Decision Diffie-Hellman problem in a group G of order n says that it is infeasible to distinguish between the distributions $(g^{\alpha}, g^{\beta}, g^{\gamma})$ and $(g^{\alpha}, g^{\beta}, g^{\alpha\beta})$, where $\alpha, \beta, \gamma \in \mathbb{Z}_n$ are randomly distributed.

Experiment 2.3.1 (Decision Diffie-Hellman, $\mathbf{Exp}^{\mathsf{ddh}-b}_{G_n,A}(\kappa)$)**.**

```
  $g \leftarrow_R G_n$
  $\alpha, \beta, \gamma \leftarrow_R \mathbb{Z}_n$
  $(X, Y, Z) \leftarrow \left(g^{\alpha}, g^{\beta}, g^{b\gamma+(1-b)\alpha\beta}\right)$
  return $A(g, X, Y, Z)$
```

Assumption 2.3.1 (Decision Diffie-Hellman Assumption over the Group G_n)**.** *For all $A \in \mathrm{PT}^*$ the advantage*

$$\mathbf{Adv}^{\mathsf{ddh}}_{G_n,A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ddh}-0}_{G_n,A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ddh}-1}_{G_n,A}(\kappa) = 1]|$$

is negligible.

The Strong RSA Assumption

The *strong RSA assumption* is a potentially stronger assumption the standard RSA assumption. Informally the strong RSA assumption says that no algorithm can, given an RSA modulus $\mathbf{N}$ and an element $\mathbf{g} \in SQ_{\mathbf{N}}$, find *any* non-trivial root of $\mathbf{g}$. The standard RSA assumption, on the other hand, does not allow the adversary itself to pick which root to find.

We write $\mathsf{Kg}^{\mathsf{rsa}}$ for the algorithm that on input 1^{κ} generates two random $\kappa/2$-bit primes $P = 2P' + 1$ and $Q = 2Q' + 1$, where P' and Q' are also prime, and returns (P, Q). Thus, $\mathsf{Kg}^{\mathsf{rsa}}$ generates a κ-bit RSA modulus $\mathbf{N} = PQ$.

Formally we define the assumption using the below experiment.

Experiment 2.3.2 (Strong RSA, $\mathbf{Exp}^{\mathsf{srsa}}_{A}(\kappa)$)**.**

```
  $(p, q) \leftarrow \mathsf{Kg}^{\mathsf{rsa}}(\kappa)$
```

```
$\mathbf{g} \leftarrow_R SQ_{pq}$
$(\mathbf{z}, e) \leftarrow A(pq, \mathbf{g})$
**if** $(\mathbf{z}^e = \mathbf{g}) \wedge (e \notin \{-1, 1\})$ **then**
  **return** $1$
**else**
  **return** $0$
**end if**
```

Assumption 2.3.2 (The Strong RSA Assumption)**.** *For all $A \in \mathrm{PT}^*$ the advantage*

$$\mathbf{Adv}_A^{\mathsf{srsa}}(\kappa) = \Pr[\mathbf{Exp}_A^{\mathsf{srsa}}(\kappa) = 1]$$

is negligible.

2.4 Security of Cryptographic Primitives

Let us now review the definitions for the cryptographic primitives we use. The definitions use experiment-based security. In some cases we give the adversary access to oracles for certain functions. We let $\mathfrak{Q}_O$ be the set of queries to oracle O and we let $\mathfrak{R}_O$ be the set of responses. Except for Definition 2.4.22 and Theorem 2.4.23, the definitions in this chapter are well-known standard definitions.

One-Way, Trapdoor, and Collision-Free Functions

A function is called a *one-way function* if it is infeasible to find the preimage of a random element. A one-way function is a *trap-door* function if there exists some secret which makes the function efficiently invertible. A function is collision-free if it is infeasible to find two values that map to the same output. Obviously a *trapdoor permutation* is a trapdoor function which is also a permutation.

Formally we consider collections of functions to define the above concepts. The properties are defined with regards to a function drawn at random function from the collection. A common method to design sampleable function collections is to make the function depend on a key and sample the key. We abuse notation by letting f denote both a function and its description.

Definition 2.4.1 (Collection of Functions)**.** *A collection of functions F is a set of functions, all of which have the same domain and image.*

We refer to the common domain of the functions of F as $\mathrm{Dom}(F)$, and to their common image as $\mathrm{Im}(F)$.

Definition 2.4.2 (Useful)**.** *A family of function collections $\mathcal{F} = (F_i)_{i=1}^{\infty}$ is useful if*

- *the uniform distribution on $\mathcal{F}$ is efficiently sampleable.*
- *the uniform distribution on $(\mathrm{Im}(F_i))_{i=1}^{\infty}$ is efficiently sampleable.*

- *for every $f \in F_i$ there exists an algorithm which outputs $f(x)$ on input $x \in \mathrm{Dom}(F_i)$ and whose running time is a polynomial in i.*

When every function $f \in F_i$ is a permutation on $\mathrm{Dom}(F_i)$ we say that $\mathcal{F}$ is a family of permutations.

In this thesis we only consider useful families of functions unless it is explicitly mentioned otherwise.

Remark 2.4.3. We have chosen to require the image to be sampleable. This allows us to use the same definitions for functions with bounded and unbounded input. We discuss this in more detail in conjuction with hash functions.

A family of function collections $\mathcal{F} = (F_i)_{i=1}^{\infty}$ is one-way if the functions are hard to invert in the following sense. Consider the below experiment.

Experiment 2.4.1 (One-Way Function, $\mathbf{Exp}^{\mathsf{ow}}_{\mathcal{F},A}(\kappa)$)**.**

```
$f \leftarrow_R F_\kappa$
$y \leftarrow_R \mathrm{Im}(f)$
$x \leftarrow A(\mathsf{guess}, f, y)$
**if** $f(x) = y$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Definition 2.4.4 (One-Way Function)**.** *A function family $\mathcal{F}$ is* one-way *if the advantage*

$$\mathbf{Adv}^{\mathsf{ow}}_{\mathcal{F},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{ow}}_{\mathcal{F},A}(\kappa) = 1]$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Experiment 2.4.2 (Collision-Free Function, $\mathbf{Exp}^{\mathsf{col}}_{\mathcal{F},A}(\kappa)$)**.**

```
$f \leftarrow_R F_\kappa$
$(m_0, m_1) \leftarrow A(\mathsf{guess}, f)$
**if** $(m_0 \neq m_1) \wedge (f(m_0) = f(m_1))$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Definition 2.4.5 (Collision-Free Function)**.** *A function family $\mathcal{F}$ is* collision-free-way *if the advantage*

$$\mathbf{Adv}^{\mathsf{col}}_{\mathcal{F},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{col}}_{\mathcal{F},A}(\kappa) = 1]$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Let F_i and F_i^{-1} be collections of functions, where $\mathrm{Im}(F_i^{-1}) \subseteq \mathrm{Dom}(F_i)$ and $\mathrm{Dom}(F_i^{-1}) = \mathrm{Im}(F_i)$ for $i \in \mathbb{N}$. Let $T_i \subset F_i \times F_i^{-1}$ such that $\forall f \in F_i\ \exists f^{-1} : (f, f^{-1}) \in T_i$ for any i. If $f(f^{-1}(y)) = y$ for any $(f, f^{-1}) \in T_i$, $y \in \mathrm{Im}(f)$, and $\mathcal{F} = (F_i)_{i=1}^{\infty}$ is one-way, then $\mathcal{T} = (T_i)_{i=1}^{\infty}$ is a family of collections of *trapdoor functions*. Obviously $\mathcal{T}$ is a family of *trapdoor permutations* if every $f \in F_i$ is a permutation.

Hard-Core Bit

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 [16]. Goldwasser and Micali [50] use this notion to construct an indistinguishable encryption scheme.

Let $\mathcal{B} : \{0,1\}^* \rightarrow \{0,1\}$ 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 $\mathcal{F} = (F_i)_{i=1}^{\infty}$ be a family of one-way functions. Consider the following experiment.

Experiment 2.4.3 (Hard-Core Bit, $\mathbf{Exp}^{\mathsf{hardcorebit}}_{(\mathcal{F},\mathcal{B}),A}(\kappa)$)**.**

```
$f \leftarrow_R F_\kappa$
$x \leftarrow_R \mathrm{Dom}(f)$
$b \leftarrow A(\mathsf{guess}, f, f(x))$
**if** $b = \mathcal{B}(x)$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Remark 2.4.6. Here we require that there exists an efficient algorithm which draws an element at random from the function domain, which differs from the definition of *useful* above, where the requirement was on the function image. Therefore our defintion for hard-core bits only makes sense for functions with finite domain.

Definition 2.4.7 (Hard-Core Bit)**.** *The function $\mathcal{B}$ is a hard-core bit for $\mathcal{F}$ if the advantage*

$$\mathbf{Adv}^{\mathsf{hardcorebit}}_{(\mathcal{F},\mathcal{B}),A}(\kappa) = \left| \Pr[\mathbf{Exp}^{\mathsf{hardcorebit}}_{(\mathcal{F},\mathcal{B}),A}(\kappa)] - \frac{1}{2} \right|$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

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

Hash Functions

A collection of functions H_κ is a collection of hash functions if $\mathrm{Dom}(H_\kappa) = \{0,1\}^*$ and $\mathrm{Im}(H_\kappa) \subseteq \{0,1\}^\kappa$. We will usually consider families of hash functions $\mathcal{H}$ which are one-way and collision-free in the sense described above.

Had we chosen that the function domains must be sampleable for the experiments for a one-way function, then the above definition would not suffice. In such a case, one could make a more cumbersome description of the hash-function. For each security parameter κ, the hash function of length κ is represented by a function ensemble $(H_\kappa^{(m)})_1^\infty$, such that $H_\kappa^{(m)}$ has $\{0,1\}^m$ as domain. Then the definitions which require sampling from the domain can be applied directly.

Since hard-core bits are defined only for functions with sampleable domain, they cannot be applied to hash functions. In this thesis we never need a hard-core bit for a hash function, and we use the more simple definition as just described.

Pseudo-Random Functions

Let $\mathcal{R}_\kappa$ be a collection of functions from $\{0,1\}^\kappa$ to $\{0,1\}^\kappa$, and let $\mathcal{R} = \{\mathcal{R}_i\}_{i=1}^\infty$. Let $\mathcal{U}_\kappa$ be the family of all functions from $\{0,1\}^\kappa$ to $\{0,1\}^\kappa$. Informally $\mathcal{R}$ is said to be pseudo-random if is infeasible to distinguish a function from $\mathcal{R}$ from a random function. The following experiment is used to formalize this.

Experiment 2.4.4 (Pseudo-Random, $\mathbf{Exp}_{\mathcal{R},A}^{\mathsf{prf}-b}(\kappa)$)**.**

```
**if** $b = 0$ **then**
    $f \leftarrow_R \mathcal{R}_\kappa$
**else**
    $f \leftarrow_R \mathcal{U}_\kappa$
**end if**
**return** $A^{f(\cdot)}(\mathsf{guess}, 1^\kappa)$
```

Definition 2.4.8 (Pseudo-Random Function)**.** *A function family $\mathcal{F}$ is* pseudo-random *if the advantage*

$$\mathbf{Adv}_{\mathcal{R},A}^{\mathsf{prf}}(\kappa) = |\Pr[\mathbf{Exp}_{\mathcal{R},A}^{\mathsf{prf}-0}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{R},A}^{\mathsf{prf}-1}(\kappa) = 1]| \ .$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Signature Schemes

A signature scheme $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$ consists of

- a key generation algorithm $\mathsf{SSKg}(1^\kappa)$ which outputs a key pair (spk, ssk).
- a signing algorithm $\mathsf{Sig}_{ssk}(m)$ which outputs a signature σ.
- a verification algorithm $\mathsf{Vf}_{spk}(m, \sigma)$ which outputs 1 if σ is valid signature on m and 0 otherwise.

The signature scheme is correct if for (pk, sk) generated by SSKg and any message msg it holds that $\mathsf{Vf}_{pk}(\mathsf{msg}, \mathsf{Sig}_{sk}(\mathsf{msg})) = 1$. It is secure against chosen-message attacks, CMA-secure [52], if it is infeasible to produce a valid message-signature pair for *any* message not previously signed, even if the adversary has access to a signing oracle $\mathsf{Sig}_{sk}(\cdot)$. Formally we use the following experiment for the definition. Recall that $\mathfrak{Q}_O$ is the set of queries passed to oracle O.

Experiment 2.4.5 (Chosen Message Attack, $\mathbf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa)$)**.**

```
$(pk, sk) \leftarrow \mathsf{SSKg}(\kappa)$
$(\mathsf{msg}, \sigma) \leftarrow A^{\mathsf{Sig}_{sk}(\cdot)}(pk)$
**if** $(\mathsf{msg} \notin \mathfrak{Q}_{\mathsf{Sig}_{sk}(\cdot)}) \wedge (\mathsf{Vf}_{pk}(\mathsf{msg}, \sigma) = 1)$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Definition 2.4.9 (CMA-Secure Signature Scheme)**.** *A signature scheme $\mathcal{SS}$ is CMA-secure if the advantage*

$$\mathbf{Adv}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = 1]$$

is negligible for any $A \in \mathrm{PT}^*$.

Commitment Schemes

A (non-interactive) commitment scheme $\mathcal{COM}$ consists of

- the commitment algorithm Commit, which takes as input a message $\mathsf{msg} \in \{0,1\}^\kappa$ and outputs a pair (c, r).
- the reveal algorithm Reveal, which takes a commitment c, a message msg, and the secret r and determines whether or not c is a commitment to msg under commitment secret r. The commitment is correct if $\mathsf{Reveal}(c, \mathsf{msg}, r) = 1$ for $(c, r) \leftarrow \mathsf{Commit}(\mathsf{msg})$.

A commitment scheme is secret, sometimes called hiding, if the adversary gains no useful information about the committed value from $(c, r) \leftarrow \mathsf{Commit}(m)$. It is binding if it is infeasible to find c, (m', r') and (m, r) such that $\mathsf{Reveal}(c, m', r') = 1$ but $m\S \neq m'$.

The following two experiments defines secrecy and binding of a commitment scheme.

Experiment 2.4.6 (Secrecy, $\mathbf{Exp}^{\mathsf{secrecy}-b}_{\mathcal{COM},A}(\kappa)$)**.**

```
$(\mathsf{msg}_0, \mathsf{msg}_1, state) \leftarrow A(\mathsf{choose}, 1^\kappa)$
$(c, r) \leftarrow \mathsf{Commit}(\mathsf{msg}_b)$
$d \leftarrow A(\mathsf{guess}, c, state)$
```

```
  **return** $d$
```

Definition 2.4.10 (Secret Commitment Scheme)**.** *A commitment scheme $\mathcal{COM}$ has secrecy if the advantage*

$$\mathbf{Adv}^{\mathsf{secrecy}}_{\mathcal{COM},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{secrecy}-0}_{\mathcal{COM},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{secrecy}-1}_{\mathcal{COM},A}(\kappa) = 1]|$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Experiment 2.4.7 (Binding, $\mathbf{Exp}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa)$)**.**

```
$(c, r_0, \mathsf{msg}_0, r_1, \mathsf{msg}_1) \leftarrow A(\mathsf{guess}, 1^\kappa)$
if $(\mathsf{Reveal}(c, \mathsf{msg}_0, r_0) = \mathsf{Reveal}(c, \mathsf{msg}_1, r_1) = 1) \wedge (\mathsf{msg}_0 \neq \mathsf{msg}_1)$ then
  return $1$
else
  return $0$
end if
```

Definition 2.4.11 (Binding Commitment Scheme)**.** *A commitment scheme $\mathcal{COM}$ is binding if the advantage*

$$\mathbf{Adv}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa) = 1]$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

One could give stronger a definitions, but in our case the above experiments suffice. As an example, they do not rule out malleability, i.e., the existence of an adversary which, after seeing one commitment, creates another commitment to a related value, which can be opened after the first commitment has been opened.

It is known that secret and binding commitment schemes exist if there exists a family of one-way permutations [49]. The construction even gives a perfectly binding scheme, i.e., even an unbounded adversary cannot decommit to more than one value.

Encryption Schemes

Symmetric Encryption Schemes

A symmetric encryption scheme consists of

- the key generation algorithm $\mathsf{Kg}(1^\kappa)$ which outputs a secret key sk.
- the encryption algorithm $E_{sk}(\mathsf{msg})$ which takes as input a key and a plaintext. It outputs a ciphertext.
- the decryption algorithm $D_{sk}(c)$ which takes as input a key and a ciphertext. It outputs a plaintext or $\perp$ if the ciphertext is invalid.

A symmetric encryption scheme is *correct* if $D_{sk}(E_{sk}(\mathsf{msg})) = \mathsf{msg}$ for $sk \leftarrow \mathsf{Kg}(1^\kappa)$ and every legal message msg.

Asymmetric Encryption Schemes

A public key, or asymmetric, encryption scheme consists of the following three algorithms.

- The key generation algorithm $\mathsf{Kg}(1^\kappa)$ outputs a key pair (pk, sk).
- The encryption algorithm $E_{pk}(\mathsf{msg})$ takes as input a public key and a plaintext. It outputs a ciphertext.
- The decryption algorithm $D_{sk}(c)$ takes as input a secret key and a ciphertext. It outputs a plaintext or $\perp$ if the ciphertext is invalid.

A public key encryption scheme is *correct* if $D_{sk}(E_{pk}(\mathsf{msg})) = \mathsf{msg}$ for $(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$ and every legal message msg.

Indistinguishable Public Key Encryption Schemes

Informally an asymmetric encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$ is called indistinguishable if it is infeasible to distinguish between the encryptions of two plaintexts of the same length. The experiment below formalizes this assumption.

Experiment 2.4.8 (Indistinguishability, $\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa)$)**.**

```
$(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$
$(\mathsf{msg}_0, \mathsf{msg}_1, state) \leftarrow A(\mathsf{choose}, pk)$
$c \leftarrow E_{pk}(\mathsf{msg}_b)$
$d \leftarrow A(\mathsf{guess}, c, state)$
return $d$
```

Definition 2.4.12 (Indistinguishable Encryption Scheme)**.** *An encryption scheme $\mathcal{CS}$ is indistinguishable if the advantage*

$$\mathbf{Adv}^{\mathsf{ind}}_{\mathcal{CS},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS},A}(\kappa) = 1]|$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

The notion of indistinguishability is equivalent to the well-known definition of semantic security for non-uniform adversaries [66], which informally says that no information about the plaintext can be efficiently computed from the ciphertext. We use the terms "indistinguishable encryption scheme" and "semantically secure encryption scheme" interchangeably in this text.

Chosen Ciphertext Security for Asymmetric Encryption Schemes

Given an asymmetric encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$ the following experiment is used to define chosen ciphertext security (CCA2). Recall that $\mathfrak{Q}_O$ denotes the set of questions answered by the oracle for O.

Experiment 2.4.9 (Asymmetric CCA2, $\mathbf{Exp}^{\mathsf{cca2}-b}_{\mathcal{CS},A}(\kappa)$)**.**

```
$(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$
$(\mathsf{msg}_0, \mathsf{msg}_1, state) \leftarrow A^{D_{sk}(\cdot)}(\mathsf{choose})$
$c \leftarrow E_{sk}(\mathsf{msg}_b)$
$d \leftarrow A^{D_{sk}(\cdot)}(\mathsf{guess}, c, state)$
if $c \in \mathfrak{Q}_{D_{sk}(\cdot)}$ then
  return $0$
else
  return $d$
end if
```

Definition 2.4.13 (CCA2-Secure Public Key Encryption Scheme)**.** *A public key encryption scheme $\mathcal{CS}$ is CCA2-secure if the advantage*

$$\mathbf{Adv}^{\mathsf{cca2}}_{\mathcal{CS},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{cca2}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{cca2}-1}_{\mathcal{CS},A}(\kappa) = 1]| \ .$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Chosen Ciphertext Security for Symmetric Encryption Schemes

Given a symmetric encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$ the following experiment is used to define chosen ciphertext security (CCA2).

Experiment 2.4.10 (Symmetric CCA2, $\mathbf{Exp}^{\mathsf{sym}-\mathsf{cca2}-b}_{\mathcal{CS},A}(\kappa)$)**.**

```
$(sk) \leftarrow \mathsf{Kg}(1^\kappa)$
$(\mathsf{msg}_0, \mathsf{msg}_1, state) \leftarrow A^{E_{sk}(\cdot), D_{sk}(\cdot)}(\mathsf{choose})$
$c \leftarrow E_{sk}(\mathsf{msg}_b)$
$d \leftarrow A^{E_{sk}(\cdot), D_{sk}(\cdot)}(\mathsf{guess}, c, state)$
if $c \in \mathfrak{Q}_{D_{sk}(\cdot)}$ then
  return $0$
else
  return $d$
end if
```

Definition 2.4.14 (CCA2-Secure Secret Key Encryption Scheme)**.** *A secret key encryption scheme $\mathcal{CS}$ is CCA2-secure if the advantage*

$$\mathbf{Adv}^{\mathsf{sym}-\mathsf{cca2}}_{\mathcal{CS},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{sym}-\mathsf{cca2}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{sym}-\mathsf{cca2}-1}_{\mathcal{CS},A}(\kappa) = 1]|$$

is negligible for any adversary $A \in \mathrm{PT}^*$.

Proofs of Knowledge

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

Definition 2.4.15 (Polynomially Bounded). *A relation $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ is polynomially bounded if there exists a polynomial $p(\cdot)$ such that $|y| \leq p(|x|)$ for all $(x, y) \in \mathcal{R}$.*

Definition 2.4.16 (**NP**-Relation). *A relation $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ is an* **NP***-relation if it is polynomially bounded and there exists a deterministic polynomial machine M such that $M(x, y) = \mathcal{R}(x, y)$.*

Definition 2.4.17 (Complexity Class **NP**). *A language $L_{\mathcal{R}} \subset \{0,1\}^*$ belongs to* **NP** *if there exists an* **NP***-relation $\mathcal{R}$ such that $x \in L_{\mathcal{R}}$ if and only if there exists an $y \in \{0,1\}^*$ such that $(x, y) \in \mathcal{R}$.*

Every relation $\mathcal{R}$ considered in this thesis corresponds to a language $L_{\mathcal{R}} \in \mathbf{NP}$ in the sense of the definition.

We use non-interactive zero-knowledge proofs of knowledge, or NIZKs, in our construction. Given a language $L \in \mathbf{NP}$ with witness relation R and $x \in L$, a NIZK (P, V) enables a prover P to prove to a verifier V that she knows a witness w such that $(x, w) \in R$.

A proof system is said to be zero-knowledge if there exists a simulator which produces proofs indistinguishable from real proofs, and the condition for it to be called non-interactive should be obvious. A Non-interactive zero-knowledge proofs (NIZK) is complete if for any $(x, w) \in R$ it holds that $V(x, P(x, w)) = 1$ and sound if for any algorithm A the probability that $V(x, \pi) = 1$ and $x \notin L$ is negligible, where $(x, \pi) \leftarrow A(\xi)$ and ξ is the common reference string. A NIZK is a proof of knowledge (NIZK-PK) if there exists an extractor which, if allowed to choose the CRS, can extract a witness.

In cryptographic proofs one often performs hypothetic experiments where the adversary is run with simulated NIZKs. If an 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.

Non-interactive zero-knowledge proofs (NIZK) were introduced by Blum, Feldman, and Micali [15]. Several works have since refined and extended the notion in various ways. Following [9] we employ the definition of adaptive zero-knowledge for NIZK introduced by Feige, Lapidot, and Shamir [43] and we use the notion of simulation soundness introduced by Sahai [77]. The notion of simulation soundness is strengthened by De Santis et al. [79].

Definition 2.4.18 (NIPS). *A triple $(p(\kappa), P, V)$ is an* efficient adaptive non-interactive proof system *(NIPS) for a language $L \in$* **NP** *with witness relation R if $p(\kappa)$ is a polynomial and P and V are probabilistic polynomial time machines such that the following properties hold.*

1. *Completeness. $(x, w) \in R$ and $\xi \in \{0,1\}^{p(\kappa)}$ implies $V(x, P(x, w, \xi), \xi) = 1$.*

2. *Soundness. For all 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.

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 NIZK which is adaptive zero-knowledge (for a single statement) in the sense of Feige, Lapidot, and Shamir [43].

Experiment 2.4.11 (Adaptive Indistinguishability, $\mathbf{Exp}^{\mathsf{ad-ind-0}}_{(P,V,S),A}(\kappa)$)**.**

```
$\xi \leftarrow_R \{0,1\}^{f(\kappa)}$
$(state, x, w) \leftarrow A(\mathsf{setup}, \xi)$
**while** $(x, w) \in R$ **do**
  $(state, x, w) \leftarrow A(\mathsf{choose}, P(x, w, \xi))$
**end while**
**return** $A(\mathsf{guess}, state)$
```

Experiment 2.4.12 (Adaptive Indistinguishability, $\mathbf{Exp}^{\mathsf{ad-ind-1}}_{(P,V,S),A}(\kappa)$)**.**

```
$(\xi, \mathsf{simstate}) \leftarrow S(1^\kappa)$
$(state, x, w) \leftarrow A(\mathsf{setup}, \xi)$
**while** $(x, w) \in R$ **do**
  $(state, x, w) \leftarrow A(\mathsf{choose}, S(x, \xi, \mathsf{simstate}))$
**end while**
**return** $A(\mathsf{guess}, state)$
```

The advantage in the experiment is defined

$$\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ad-ind-0}}_{(P,V,S),A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ad-ind-1}}_{(P,V,S),A}(\kappa) = 1]|$$

and the notion of adaptive zero-knowledge is given below.

Definition 2.4.19 (Adaptive Zero-Knowledge (cf. [43]))**.** *A NIPS (P, V) is adaptive zero-knowledge (NIZK) if there exists a polynomial time Turing machine S such that the advantage $\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa)$ is negligible for any adversary $A \in \mathrm{PT}^*$.*

Let us now formalize simulation soundness. Recall that $\mathfrak{R}_O$ is the set of responses by oracle O.

Experiment 2.4.13 (Simulation Soundness, $\mathbf{Exp}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa)$ (cf. [79]))**.**

```
$(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$
$(x, \pi) \leftarrow A^{S(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})}(\mathsf{guess}, \xi)$
**if** $(\pi \notin \mathfrak{R}_{S(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})}) \wedge (x \notin L) \wedge (V(x, \pi, \xi) = 1)$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Definition 2.4.20 (Simulation Soundness (cf. [77, 79]))**.** *A NIZK (P, V) with polynomial time simulator S for a language L is unbounded simulation sound if*

$$\mathbf{Adv}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa) = 1]$$

is negligible for all $A \in \mathrm{PT}^*$.

De Santis et al. [79] extend the results in [43] and [77] and prove the following result.

Theorem 2.4.21. *If there exists a family of trapdoor permutations, then there exists a simulation sound NIZK for any language in* **NP** *in the CRS-model.*

We abbreviate "efficient non-interactive adaptive zero-knowledge unbounded simulation sound proof" by NIZK.

It is important to note that the above definition does not require that it is possible to extract the witness, i.e., they are not proofs of knowledge. To our knowledge, there are no results on the existence of simulation-sound proofs of knowledge, although signatures of knowledge [33] are similar.

One must be careful when formalizing extractability. As for simulation soundness, we want to give the adversary the ability to request simulated proofs for theorems of its choice, and if it outputs a valid proof, the extractor should be able to extract a witness. In the original definitions of NIZK proofs of knowledge [80, 79], *soundness* and *validity*, i.e., the requirement on extractability, are two separate properties. Such a definition would be hard to use when designing protocols. In a protocol, we need to produce a single CRS which is used both for simulation and for extraction. Therefore it makes sense to combine the two properties in a single experiment.

Experiment 2.4.14 (Extractable Simulation Soundness, $\mathbf{Exp}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa)$)**.**

```
$(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$
$(x, \pi) \leftarrow A^{S(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})}(\mathsf{guess}, \xi)$
$w \leftarrow S(\mathsf{extract}, x, \pi, \xi, \mathsf{simstate})$
**if** $(\pi \notin \mathfrak{R}_{(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})}) \wedge ((x, w) \notin R) \wedge (V(x, \pi, \xi) = 1)$ **then**
  **return** 1
**else**
  **return** 0
**end if**
```

Definition 2.4.22 (Extractable Simulation Soundness)**.** *A NIZK (P, V) with polynomial time simulator S for a language L is unbounded extractable simulation sound if*

$$\mathbf{Adv}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa) = 1]$$

is negligible for all $A \in \mathrm{PT}^*$.

In Section 5.6 we prove the following theorem. A dense encryption scheme is an encryption such that a random string is a valid public key with non-negligible probability.

Theorem 2.4.23. *Given an* **NP***-language* L *there exists a proof system* (P, V) *for* L *which is extractable, adaptively indistinguishable, and unbounded simulation sound in the CRS-model if there exists a family of trapdoor permutations and a dense encryption scheme.*

Chapter 3

Contributions

In this chapter we present our contributions and give a brief description of our results.

3.1 Electronic Cash

Introduction and Background

Anonymous electronic cash was introduced by Chaum et al. [34]. The underlying problem can be described as follows. We want the coins to be anonymous, but we want to catch a malicious users which spends the same coins more than once, e.g., but copying the digital coin and using it at two different merchants. This may at first seem like a contradiction. The solution proposed by Chaum was to design the scheme so that a coin spent once does not reveal the user's identity, but the identity can be extracted if the coin is double-spent. This is still the paradigm which most schemes use.

This thesis contribute in two ways. We discuss ways to improve the security definitions in order to protect the user from a malicious bank (or corrupt employee within the bank). Our second contribution is a scheme which is highly efficient but sacrifices some of the anonymity to the bank.

An Improved Security Definition

Background

From the point of view of the bank, a scheme for electronic cash is secure if it is infeasible to construct valid coins by other means than withdrawing them. From the point of view of the merchant, a coin that has been spent should always be accepted by the bank. Finally, to be secure for a user, the anonymity property should hold even if the bank conspires with other users and merchants. In addition, the bank should not be able to claim that the user has withdrawn more coins than she has, or falsely accuse the user of double-spending.

In the recent years, several papers have focused on giving precise security definitions for tasks such as group signatures [9, 11] and ring signatures [12]. We suggest a definition of security for schemes for electronic cash. In addition to an experiment-based definition we construct an ideal functionality for electronic cash and show that security in the experiment setting implies simulation-based security in the framework for universal composability [30] using the ideal functionality.

Strengthening the Definition

We point out that previous definitions do not rule out a corrupt bank cheating a user. The scenario is that the bank claims that a user has withdrawn a coin, but the user denies this. We argue that the protocol should include a mechanism to solve such an issue. Our definition addresses this issue by requiring a proof of withdrawal from the bank.

We define six algorithms which form a scheme for electronic cash, key generation for the bank, key generation for a user, spending a coin, identification of a double-spender, verification of a withdrawn coin, and verification of a spent coin. In addition we require the existence of a protocol between a user and the bank to withdraw a coin.

Our security definition is based on four properties, *unforgeability*, stating that valid coins can only be issued by the bank, *anonymity*, ensuring a user stays anonymous even if the complete system conspires against her, *non-frameability*, requiring that no honest user can be accused of double-spending even by a corrupt bank, and *exculpability*, ensuring that no user can be falsely accused of withdrawing a coin. Previously considered security properties, as well as the property mentioned above, follow from security under these four experiments. The fact the security in the UC-model follows from the four experiments is another argument that our definition cover the intuitive meaning of security for electronic cash.

Let us now describe the experiments in more detail.

Unforgeability. In the unforgeability experiment, the adversary is given an honestly generated bank public key, and the ability to add corrupt users to the system. The corrupt users can interact with an honest bank to withdraw coins, not necessarily following the withdrawal protocol. After the execution, the honest bank holds a number of withdrawal proofs, and the adversary has created a number of spent coins. The adversary is successful if the number of spent coins exceeds the number of withdrawal proofs, and no two spent coins are deemed to be double-spent.

Anonymity. The adversary in the anonymity experiment is allowed to construct the bank public key itself, reflecting that anonymity should hold also against a fully corrupt bank. The adversary can add honest users to the scheme, and it can issue coins to the users by playing the bank's part in the withdrawal protocol. Then it chooses two coins. The experiment spends one of the coins and hands the spent coin to the adversary without telling the adversary which of the coins was spent. If the adversary correctly guesses which coin has been spent, then it wins the experiment.

Non-Frameability. The non-frameability experiment is defined as follows. The adversary chooses a bank public key and asks the experiment to add honest users to the system. The adversary can issue coins to the honest users by playing the bank's part in the withdrawal protocol and also spend the coins, possibly forcing a user to double-spend in the process. The adversary outputs a list of coins. If the number of double-spendings in the list exceeds the number of double-spendings explicitly requested by the adversary, the adversary is successful in the experiment. This reflects the requirement that no honest user may be held responsible for more double-spendings than she actually is guilty of.

Exculpability. In the exculpability experiment, the adversary constructs the bank public key and is allowed to add honest users to the system. It can also play the bank's part in the withdrawal protocol to issue coins to the honest users. The adversary outputs a list of withdrawal proofs, and it wins if the list contains a valid proof, or coin, which the coin holder cannot spend.

A Construction

We construct a scheme using general methods, which is secure under our definition in the common reference string (CRS) model assuming the existence of a family of trapdoor permutations. The scheme is not intended for practical use, but should rather be considered a proof of concept.

A Practical Scheme for Electronic Cash

Background

Most payment schemes involve trapdoor functions such as variants of ElGamal encryption or RSA groups. A real-life electronic cash scheme would probably be implemented on a portable device with low computational power such as a smart-card or a mobile phone. For such schemes it is important that the amount of computation is low, especially on the user side. The difference between zero, one or two exponentiations in the payment protocol is significant, whereas many schemes require tens, or in some cases hundreds, of exponentiations. The merchant terminal is more comparable to a low-end PC, but also in this case it is desirable to reduce the amount of computations necessary to one or a few exponentiations.

Outline of Scheme

In Chapter 6 we propose a scheme which relies on symmetric primitives such as symmetric encryption, hash functions and pseudo-random functions. The only computations performed by the user during payment is evaluation of pseudo-random functions, and the merchant verifies a signature. The scheme has been implemented on a mobile platform [89]. The scheme is inspired by the scheme by Sander and Ta-Shma [78].

When a user withdraws a coin, the bank encrypts the user's identity. Then the user uses a pseudo-random function to create a list of values and sends the hash value of the pseudo-random values to the bank. The coin, consisting of the encrypted identity and the hash values, is then inserted as a leaf into a Merkle hash tree. After a certain amount of time, the bank builds the tree and publishes the root. To spend a coin, the user reveals half of the preimages of the hash values together with a path from the coin up to a published root. The merchant verifies the correctness of the preimages, and in addition verifies that the chain of hash values is valid.

If a user double-spends a coin, then she has revealed the preimage of more than half of the hash values. If this happens the bank decrypts the encrypted identity. From only the revealed preimages of a double-spent coin, it may be possible to successfully spend the coin a third time. In other words, a user double-spending a coin risks being held responsible for additional purchases. This gives additional incentive not to double-spend.

The anonymity of the scheme follows from the security of the encryption scheme, and unforgeability of coins follows since the hash function is collision-free.

As an additional feature of our scheme the payment protocol is non-interactive. In other words, the user produces a coin that can only be deposited by the designated merchant. This enables a user to prepare a coin for a certain merchant. In addition, anyone can verify that the coin has been prepared for that merchant. As an example, a parent can give a coin to their child which can be spent only at a certain store.

We show security for our scheme in the framework for universal composability (UC) [30]. We stress that our proof of security is in the plain model and not in the random oracle model. We only assume that the encryption scheme is CCA2-secure, that the hash functions are one-way and collision-free, and that the pseudo-random functions are indistinguishable from random functions. We believe that the current scheme is the first scheme for electronic cash with a security proof in the UC-model and also the first scheme that does not use the random-oracle model for its security proof.

3.2 Hierarchical Group Signatures

Background

Recall group signatures and the payment card application of group signatures from Section 1.12. 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.

A More Complex Application

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.

Introducing the Hierarchical Notion

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 3.1.

In the original notion the group manager can always identify a signer, but nobody else can distinguish between signatures 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

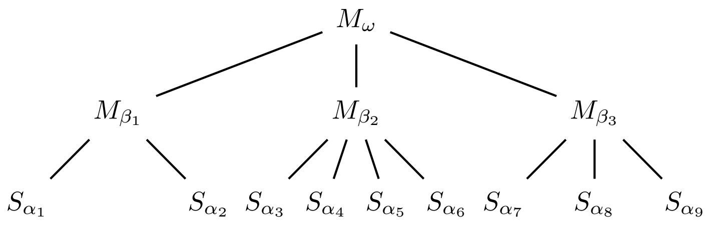


Figure 3.1: A tree of group managers and signers where $\omega = \{\beta_1, \beta_2, \beta_3\}$, $\beta_1 = \{\alpha_1, \alpha_2\}$, $\beta_2 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6\}$, and $\beta_3 = \{\alpha_7, \alpha_8, \alpha_9\}$.

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.

Definitions

Like schemes for group signatures, a scheme for hierarchical group signatures consists of the key generation algorithm HGKg, the signing algorithm HGSig, the verification algorithm HGVf, and the opening algorithm HGOpen. As opposed to regular group signature scheme, the opening algorithm may return an empty response $\perp$ if the group manager trying to open the signature is not an ancestor (directly or indirectly) of the signer.

Let us now discuss the experiments which define security for a scheme for hierarchical group signatures. We use the same experiments as Bellare et al. [9], although we modify them to fit our needs. Informally the anonymity experiments examines whether the signatures leaks any knowledge of its signer's identity other than what is required from a correct scheme, and the traceability experiment roughly corresponds to existential forgery for plain signatures.

Hierarchical Anonymity. In the experiment for hierarchical anonymity, the adversary is first given the public key of the scheme and the private keys of all signers. Then it may request the private keys of group managers of its choice, and it has access to an HGOpen oracle, which opens signatures on behalf of a group manager of the adversary's choice. The adversary picks two signers and a message.

The experiment lets one of the two signers sign the message, without informing the adversary of who the signer is. Still having access to the HGOpen oracle, the adversary guesses which signer produced the signature. It wins if it successful without having corrupted a group manager which by definition can distinguish between the two signers and having queried the HGOpen oracle on the challenge signature.

Hierarchical Traceability. In the full traceability experiment, the adversary is given the private keys of all group manager. It may corrupt signers of its choice, and receives the private key for the corrupted signers. During the process it has access to a HGSig oracle.

At the end of the experiment the adversary outputs a signature for a message for which the oracle has not produced a signature. The adversary wins if the signature cannot be opened or if it opens to a non-corrupted signer. In addition, the adversary wins if two group managers on the same level can open the signature.

The essence of the traceability experiment is that a group manager should always be able to trust the output of the HGOpen algorithm.

Constructions

We give two constructions for hierarchical group signature schemes. The first one uses general methods and is shown to be secure if a family of trapdoor permutation is used. The second construction is explicit and is shown to be secure under the strong RSA assumption and the decision Diffie-Hellman assumption.

A Construction Using General Methods

As building blocks for our scheme we use a scheme for (non-hierarchical) group signatures, a public-key encryption scheme, and a non-interactive zero-knowledge proof system (NIZK). These schemes are known to exist under the assumption of trapdoor permutations.

The key generator constructs a key pair for the encryption scheme for each group manager. The private key is handed to the group manager and the public key is added to the public key of the scheme. Each signer is given a secret key for the group signature scheme, and the public key is included in the public key of the hierarchical group signature scheme.

To sign a message a signer first constructs a signature for the group signature scheme. Then it constructs two list of ciphertexts. The first list is a chain where the signer's identity is encrypted with the public key of its parent, the parent's public key is encrypted with its parent's public key and so on until the root is reached.

In addition all public keys in the list are encrypted with a separate key. Finally a NIZK proving that everything is formed as described here is produced.

Verification simply consists of verifying the proof of knowledge. When a group manager opens a signature it decrypts the ciphertext on its level and verifies against the additional list that it should be able to open the signature. From the plaintext it can identify the next step in the chain. Since it is checked that the correct key is used for decryption, only one group manager on each level can open the signature.

An Almost Practical Construction

Next we give a scheme which is secure under the strong RSA assumption and the decision Diffie-Hellman assumption. Although the scheme is described explicitly, it is still slow, requiring about a minute to compute a signature on an ordinary computer.

Each group manager is given an ElGamal private key and the public key is made part of the scheme public key. Each signer is given a Cramer-Shoup signature on the public keys on the path from the root to the signer. A hierarchical group signature is constructed by computing a commitment to the Cramer-Shoup signature, producing a chain of ciphertexts as in the general construction, and computing a NIZK that the signature is formed correctly.

Informally the scheme has traceability, since it is possible to extract a Cramer-Shoup signature from a forger hierarchical group signature. An adversary breaking the traceability experiment can be used to break the CMA-security of the Cramer-Shoup signature scheme. Anonymity holds, since neither the commitment, nor the ElGamal ciphertexts reveals any information about the identity of the signer.

An Optimistic Scheme

There is room for improvement in terms of efficiency of the scheme above. We now relax the definitions somewhat in order to be able to give such an improvement. In the original scheme, a group manager can always open its signatures. In our relaxation, which we call *optimistic*, a group manager may not be able to open its own signatures. However, there is a trusted party which can always identify the signer. For this reason we introduce a new function, $\mathsf{HGTrustOpen}$, and replace HGOpen by $\mathsf{HGOptOpen}$.

The term optimistic is used since for honest signers, the protocol works just as ordinary hierarchical group signatures. The inconvenience of interacting with a trusted party only occurs for a dishonest signer, which will itself be exposed in the process.

The security definitions are quite similar to those of ordinary hierarchical group signatures. The main difference is that in the traceability experiment, $\mathsf{HGTrustOpen}$ replaces HGOpen. This reflects that $\mathsf{HGTrustOpen}$ is in some sense the same as HGOpen, since both algorithms are required to every time output a valid response. The adversary wins if it breaks traceability in the same sense as for non-optimistic signatures. In addition, it wins if a group manager opens the signature to something else than $\mathsf{HGTrustOpen}$ and $\perp$. That is, a group manager may fail to open a signature, but it may not point out an honest signer.

Our construction works as follows. Each signer α is given a prime p_α. The prime is constructed so that its binary representation identifies a path from the root to the signer. The private key of each signer is a p_αth root of an element $\mathbf{y}$ in an RSA group, and the group managers are given ElGamal keys. To sign the signer commits to its root and creates a chain of ciphertexts from itself to the root. Then it produces a NIZK that the signatures are formed as described above.

Informally traceability holds, since in the random oracle model an RSA root can be extracted from a valid signature. Therefore an adversary breaking the traceability can be turned into a machine which computes root of an element in an RSA group, contradicting the strong RSA assumption. Anonymity holds since no knowledge about the signer leaks from the commitment, the ciphertexts, or the NIZK.

Universal Composability

In Section 12 we give an ideal functionalities for the proper hierarchical group signatures from [84] as well as for the optimistic signatures of this paper. We then show both for proper and optimistic hierarchical group signatures that a scheme which is secure according to the experiment-based definition also securely realizes its ideal functionality for a static adversary.

The ideal functionality constructs signatures with the actual signing algorithm. Since the environment has access to the secret keys of corrupt group managers, we need to ensure that signatures are opened correctly with regard to these keys. On the other hand, signatures should be obviously indistinguishable in all other cases. We solve this by using the legimite keys of the corrupt group managers, but modifying the other keys so that signatures which should be indistinguishable are identically distributed.

Verification of signatures is along the same lines as in [31]. A signature is deemed invalid if the actual verification algorithm returns 0, or if it opens to an uncorrupt party. This corresponds to [31] where the ideal verification algorithm considers a signature valid if it passes the real verification algorithm and the signer is corrupt.

In Section 12 we show that a scheme which is secure in the experiment-based definition is also secure in the UC-setting.

Part II

Electronic Cash

Chapter 4

Introduction and Background

4.1 About This Part

In this part we present two new results on electronic cash. The first contribution is an improved definition of security for electronic cash together with a realization under general assumptions. Although the protocol is not practical, it shows that it is possible to construct a protocol which fulfills our stronger definition. The second result is a practical scheme which does not ensure the same privacy towards the bank, but which is highly efficient.

4.2 Previous Work

The concept of electronic cash, e-cash, was introduced by Chaum et al. [34], and several subsequent schemes have been proposed [20, 44, 86, 78, 73, 69, 68, 25]. In an e-cash scheme there are three types of participants – the bank, merchants, and users. The users can withdraw coins from the bank and spend them at merchants. An e-cash scheme either be *online* or *offline*. In the former case the bank is involved in every transaction, whereas in the second case payments can be performed without contacting the bank. Obviously offline schemes are preferable to online schemes. However, an electronic coin, being nothing but a string of numbers, can be copied and spent more than once, and in an offline scheme such double-spendings cannot be detected during the actual purchase. Rather than preventing double-spending, offline schemes are designed so that double-spenders are detected and identified.

Privacy is a crucial ingredient of e-cash schemes. It is desirable that merchants cannot learn the identity of the user, or even determine whether two payments were made by the same user or not. Many schemes also provide the same privacy towards the bank. However, anonymity also works in favor of criminals using the scheme for illegal activities protected by the privacy offered. To protect against such events some schemes offer the possibility for trusted third parties to trace a payment.

Most schemes require a merchant to deposit a coin after the purchase. A few schemes allow a coin to be transferred between users in several steps before it is deposited at the bank [72, 73]. Such schemes are said to have *transferable coins.* Another possible feature is divisability, i.e., that a coin may be spent only in part [73, 69, 68].

In [21], the possibility of later revoking the anonymity of the coins is added, which may be desirable for legal reasons. Sander and Ta-Shma [78] present a system where the bank does not have a secret key. The scheme in Chapter 6 is based on the ideas of that system. The similarities and differences between our system and the system introduced by Sander and Ta-Shma are discussed in more detail in Section 6.1.

4.3 Group Signatures and E-cash Schemes

Group signatures bear many resemblances to electronic cash. Group signatures are indistinguishable to anyone but the group manager in very much the same way payments are indistinguishable in anonymous e-cash schemes. One important difference is that there is no concept of double-spending for group signatures.

In Chapter 5.2 and Chapter 6.6 we compare our two schemes to group signatures.

Chapter 5

A Stronger Definition for Anonymous Electronic Cash

5.1 Introduction

In this chapter we propose a definition of security for schemes for electronic cash which gives a stronger protection for the user against the bank than the schemes mentioned in Section 4.2. In particular, after a withdrawal the bank acquires a proof in case the user claims the withdrawal never took place.

5.2 Protocol Definition and Security Model

While some security properties of electronic cash are obvious and dealt with from the very first scheme, others are more subtle. Naturally a scheme must not allow for a user to forge coins, and a double-spender must be detected. The schemes [25, 87] require that a corrupt bank cannot accuse an honest user of double-spending, whereas this requirement is not explicit in some other papers, e.g., [68, 61]. However, to our knowledge, no scheme discusses the possibility of a corrupt bank falsely claiming that an honest user has withdrawn a coin, or rejecting a deposition from a merchant of a legally spent coin. The tendency seems to be to, apart from anonymity, protect the interests of the bank rather than those of the user.

We give a definition requiring that the bank be able to prove withdrawals. Thus, after executing the withdrawal protocol, the output of the bank should be a proof of withdrawal and the output of the user should be a valid coin. However, the neither part may benefit from aborting the protocol prematurely. While there exist protocols which address this issue, so called fair exchange [14], they are either based on gradual release of information and thus not very practical, or require the presence of a third trusted party. Since we would like a definition that can be instantiated with a practical protocol, we use a different approach. After the execution, the bank receives a withdrawal proof, and the user receives a coin secret data which can be used to spend the coin. The honest bank would send the withdrawal proof to the user, who can use it as a coin. Should the bank fail to do this, the user can challenge the transaction and force the bank to prove that a coin has indeed been withdrawn. Since the proof can be used a coin, the scheme is fair also from the point of view of the user. We call the bank's output the coin public data and we call the user's output the coin user secret data.

We require that spent coins be publicly verifiable to avoid the possibility of the bank rejecting a deposition and to ensure that a merchant cannot deny having received a payment. In particular the bank can verify a spent coin. Therefore there is no need for an interactive deposition protocol. The merchant simply hands the spent coin to the bank.

Our security definition is based on four experiments, *unforgeability*, stating that valid coins can only be issued by the bank, *anonymity*, ensuring a user stays anonymous even if the complete system conspires against her, *non-frameability*, requiring that no honest user can be accused of double-spending even by a corrupt bank, and *exculpability*, ensuring that no user can be falsely accused of withdrawing a coin. Previously considered security properties, as well as the property mentioned above, follow from security under these four experiments. The fact that security in the UC-model follows from the four experiments is another argument that our definition covers the intuitive meaning of security for electronic cash.

Even with all merchants being fully corrupt, a scheme should stay secure in the sense that coins cannot be forged and anonymity still holds. If spent coins are anonymous and cannot be forged even with the merchants' secret keys revealed, there is no reason to keep them secret. Therefore the merchants do not have any secret keys in our definitions, and the merchants do not take part in any protocol. In particular, the spending algorithm is non-interactive, i.e., to spend a coin, the user applies an algorithm to spend it, and gives the merchant's identifier as parameter.

With a non-interactive spending algorithm, the merchant's consent is not necessary in order to spend a coin. In practice, a user would require some sort of contract before handing the merchant a coin, but we feel this is best handled outside of our protocol. In standard banking systems it is indeed possible to wire money without the recipient's approval, although it is usually not very sensible to do so.

We assume the existence of a PKI, i.e., given a public key there exists a method to obtain the identity of the holder of the key. Since we have a PKI and assume the existence of trapdoor permutations, we can construct secure and authenticated communication. We do not explicitly define a protocol to register a user. If the protocol requires some secret information to be passed from the bank to the user, this can be done in the withdrawal protocol, since we have the existence of secure and authenticated channels. Therefore there is no loss of generality in not having a registration protocol.

In this paper we discuss payment schemes containing all basic properties, but there are many possible extensions. Examples of such alternative definitions include the presence of a trusted third party which can identify coin spenders, even when they have not double-spent. Such schemes are called *fair*. Another extension is the possibility to transfer a coin between users in several steps before it is deposited at the bank, and divisible coins. We leave it as an open problem to adjust the definition to handle also such cases.

We construct a scheme using general methods, which is secure under our definition in the common reference string (CRS) model assuming the existence of a family of trapdoor permutations. The scheme is not intended for practical use, but it is rather a proof of concept.

Participants

The participants are the bank $\mathcal{B}$ and users $\mathcal{U}_i$. Each merchant has an identity mid, but the merchants do not take active part in any of our protocols.

Algorithms and Protocols

We now define the algorithms and protocols of which a scheme for electronic cash consists. For non-interactive algorithms the definition is straight-forward. We define two-party protocols as a pair of algorithms, where each participant executes one algorithm. The algorithms take as input a message and a state. The state is initialized with the private input of the party. On startup of a protocol the initiating party executes the algorithm with $\emptyset$ as message. Each algorithm outputs a pair $(\mathsf{msg}, \mathit{state})$, where msg is handed as message to the other party's algorithm, and *state* is passed as input by the executing party the next round. When a party's algorithm outputs $p = \perp$ the protocol is finished, and the final value of each party's *state* is parsed as private output.

The following algorithm illustrates the execution between two parties using algorithms A and B with private input sk_A, sk_B, respectively.

```
$state_A \leftarrow sk_A$
$state_B \leftarrow sk_B$
while $(\mathsf{msg}_A \neq \perp) \wedge (\mathsf{msg}_B \neq \perp)$ do
    $(\mathsf{msg}_B, state_A) \leftarrow A(\mathsf{msg}_A, state_A)$
    if $\mathsf{msg}_B \neq \perp$ then
        $(\mathsf{msg}_A, state_B) \leftarrow B(\mathsf{msg}_B, state_B)$
    end if
end while
return $(state_A, state_B)$
```

None of our subprotocols involve more than two parties, which allows us to use the simplified notation above for interactive subprotocols. As a matter of fact, the only protocols which involves exactly two parties is the withdrawal protocol.

There is an algorithm for creating a bank key pair and a user key pair. After the user has generated its keys, the public key is inserted into the PKI and hence tied to the user's identity.

The merchants have no secret keys. Instead each merchant has an identity $\mathsf{mid} \in \{0,1\}^{\kappa/2}$, which together with a transaction identity $\mathsf{tid} \in \{0,1\}^{\kappa/2}$ uniquely identifies a transaction. The reason to have fixed-length identities can informally be described as follows. We would like a spent coin to have a fixed length, and we would like a scheme which is secure under general assumptions such as the existence of trapdoor permutations. If a user can create two inputs which result in the same spent coin, then she will not be caught as a double-spender. Therefore it must be infeasible to construct two such inputs. However, this is what is required from a collision-free hash functions. Hence such a scheme could be used to construct a collision-free hash function keyed on all other parameters of the scheme, which would solve the long-standing open problem of the existence of collision-free hash functions assuming only the existence of trapdoor permutations.

On the other hand, should we assume the existence of a collision-free hash function, we could have merchant and transaction identifiers of arbitrary length and hash them to a value of appropriate length.

Algorithm 5.2.1 (Bank Key Generation BKg)**.**
INPUT: $\mathsf{BKg}(1^\kappa)$, where κ is the security parameter.
OUTPUT: $(\mathsf{bpk}, \mathsf{bsk})$, where bpk is a bank public key and bsk is a bank secret key.

Algorithm 5.2.2 (User Key Generation UKg)**.**
INPUT: $\mathsf{UKg}(1^\kappa)$, where κ is the security parameter.
OUTPUT: $(\mathsf{upk}, \mathsf{usk})$, where upk is a user public key and usk is a user secret key.

Protocol Head 5.2.1 (Coin Withdrawal, $(\mathsf{UWithdraw}, \mathsf{BWithdraw})$)**.**
PARTIES: Bank $\mathcal{B}$, User $\mathcal{U}$.
PRIVATE INPUT OF $\mathcal{B}$: Bank public key bpk, bank secret key bsk.
PRIVATE INPUT OF $\mathcal{U}$: Bank public key bpk, user public key upk, user secret key usk.
PRIVATE OUTPUT OF $\mathcal{B}$: Coin public data cpd.
PRIVATE OUTPUT OF $\mathcal{U}$: Coin user secret data cusd.

$(\mathsf{UWithdraw}, \mathsf{BWithdraw})$ is the protocol used when a user withdraws a coin. The private input of the user is a user private key usk, a user public key upk, and a bank public key bpk. The private input of the bank is a bank secret key bsk and a bank public key bpk. The user's private output is the coin user secret data cusd used when spending the coin, whereas the bank's output is interpreted as the coin public data, cpd, which we will sometimes simply refer to as a coin. Normally the bank would hand the public data to the user, but we do not address this in the protocol. If the bank fails to hand the coin public data to the user and still charge the user's account, the user would request a proof of the withdrawal. Since the coin public data is the proof, the bank would be forced to reveal it.

To simplify notation, we use a short notation for an honest execution of the protocol. We define $\mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{upk}, \mathsf{usk})$ to be the result, i.e., coin public data cpd and coin user secret data cusd, of a withdrawal where both parties behave according to the protocol.

Algorithm 5.2.3 (Coin Spending Spend)**.**
INPUT: $\mathsf{Spend}(\mathsf{cpd}, \mathsf{upk}, \mathsf{usk}, \mathsf{cusd}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$, where cpd is coin public data,

upk is a user public key, usk is a user secret key, cusd is a coin user secret data, $\mathsf{mid} \in \{0,1\}^{\kappa/2}$ is a merchant identity, $\mathsf{tid} \in \{0,1\}^{\kappa/2}$ is a transaction identity, and bpk is a bank public key.
OUTPUT: $\mathsf{spentcoin}$, where $\mathsf{spentcoin}$ is a (publicly verifiable) spent coin.

Informally $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$ returns the public key upk of the double-spender if $\mathsf{spentcoin}_1$ and $\mathsf{spentcoin}_2$ are two spendings of the same coin. Otherwise it returns $\perp$.

We require that the spent coins handed to $\mathsf{VfDoubleSpent}$ have been verified, or the output of the algorithm is undefined. This requirement could be removed by including $(\mathsf{mid}, \mathsf{tid})$ for each coin in the call, but this would make the interface unnecessarily complex.

The bank secret key is not used in the below algorithm. If a key is indeed needed, and it is separate from the key used to issue coins, then it can be made public by including it into bpk. It is quite realistic to have double-spendings being publicly verifiable. In case of a double-spending, the bank would still need to able to prove this to a third party. It also makes the definitions less cumbersome.

Algorithm 5.2.4 (Identifying a Double-Spender, $\mathsf{VfDoubleSpent}$)**.**
INPUT: $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$, where the two input parameters $\mathsf{spentcoin}_1$ and $\mathsf{spentcoin}_2$ are two spent coins and bpk is a bank public key.
OUTPUT: upk, a (possibly empty) user public key.

In addition to the above, there are two algorithms which verify the validity of coins produced during withdrawal and spending, $\mathsf{VfCoin}(\mathsf{cpd}, \mathsf{upk}, \mathsf{bpk})$, $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$, The algorithms output 1 if the proof is valid with regards to the additional input parameters and 0 otherwise.

Algorithm 5.2.5 (Verifying a Withdrawal, VfCoin)**.**
INPUT: $\mathsf{VfCoin}(\mathsf{cpd}, \mathsf{upk}, \mathsf{bpk})$, where cpd is the public data of a withdrawn coin, upk a user public key, and bpk a bank public key.
OUTPUT: $b \in \{0,1\}$.

Algorithm 5.2.6 (Verifying a Spent Coin, $\mathsf{VfSpentCoin}$)**.**
INPUT: $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$, where $\mathsf{spentcoin}$ is a spent coin, $\mathsf{tid} \in \{0,1\}^{\kappa/2}$ is a transaction identity, $\mathsf{mid} \in \{0,1\}^{\kappa/2}$ is a merchant identity, and bpk is a bank public key.
OUTPUT: $b \in \{0,1\}$.

Each new coin public data cpd gives the bank the right to charge the account once. To define what is meant by a "new" coin, we must decide on what we mean by a two coin public data cpd_1 and cpd_2 being equal. The most obvious choice would be to require the two bit-string to be equal. However, we allow the scheme to define the equivalence relation in a different way. This equivalence relation is implicitly used also in the security experiment, e.g., when building sets of coin public data. The reason to allow this is that a scheme may allow the bank to reform a cpd into a different cpd' in a certain pattern, e.g., by resigning some data with a probabilistic signing scheme. Rather than require a scheme to take additional steps to withstand such an attack, we allow it to simply define two such coins to be identical and thus not to give the bank the right to charge the account a second time.

Correctness

By correctness we mean that the scheme works as expected when all participants are honest. Proving correctness is often straight-forward, and this property is sometimes not stated explicitly. Here we define correctness for a scheme as defined above.

Experiment 5.2.1 (Correctness, $\mathbf{Exp}^{\mathsf{correct}}_{\mathcal{EC},A}(\kappa)$)**.**

```
$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$
$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{UKg}(1^\kappa)$
$(\mathsf{cpd}, \mathsf{cusd}) \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{upk}, \mathsf{usk})$
if $\mathsf{VfCoin}(\mathsf{cpd}, \mathsf{upk}, \mathsf{bpk}) = 0$ then
  return 0
end if
$(\mathsf{mid}, \mathsf{tid}) \leftarrow A(\mathsf{guess}, \mathsf{bpk}, \mathsf{upk}, \mathsf{cpd})$
$\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{usk}, \mathsf{cpd}, \mathsf{cusd}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$
if $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk}) = 0$ then
  return 0
end if
return 1
```

Definition 5.2.1 (Correctness)**.** *A scheme for electronic cash $\mathcal{EC}$ is* correct *if the advantage*

$$\mathbf{Adv}^{\mathsf{correct}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{correct}}_{\mathcal{EC},A}(\kappa) = 0]$$

is negligible as a function of κ for any adversary $A \in \mathrm{PT}^*$.

Detection of double-spenders is not included in the definition of correctness. This may seem strange at first, but correctness only stipulates how the protocol works with honest parties, and an honest party does not double-spend. As we will see later, the definition of unforgeability implies that double-spenders are detected.

Security

We describe four experiments, or games, to define security for a scheme for electronic cash. In each experiment the adversary has access to a number of oracles defined below. They operate on the following global parameters.

- $\mathfrak{U}$ contains all public keys inserted into the PKI.
- $\mathfrak{C}$ contains the public keys of corrupt users. Obviously $\mathfrak{C}$ is a subset of $\mathfrak{U}$.

- $(\mathsf{upk}_i, \mathsf{usk}_i)$ is the public and private key of the ith honest user.
- l is the number of coins withdrawn from the bank using the withdrawal oracle. It is initialized to 0.
- ds_i is the number of double-spendings that has been made on behalf of user upk_i using the spending oracle $\mathsf{HonestSpend}$. It is initialized to 0.
- CSK_i is the set of coin user secret data for user upk_i produced when the withdrawal oracle is used. For a new user it is initiated as the empty set.

The oracles are defined as follows.

$\mathsf{HonestUKg}(1^\kappa)$ calls $\mathsf{UKg}(1^\kappa)$ to generate a key pair $(\mathsf{upk}, \mathsf{usk})$. The public key upk is inserted into the PKI and $\mathfrak{U}$. For the ith call to the oracle the key pair $(\mathsf{upk}, \mathsf{usk})$ is stored in the key list as $(\mathsf{upk}_i, \mathsf{usk}_i)$. The public key upk is returned.

$\mathsf{AddCorruptU}(\mathsf{upk})$ inserts the key upk into the PKI and into the sets $\mathfrak{C}$ and $\mathfrak{U}$.

$\mathsf{HonestUWithdraw}(i, j, \mathsf{msg})$ executes one step of withdrawal session j for $\mathcal{U}_i$. More precisely, if session j has not been instantiated for $\mathcal{U}_i$, i.e., $state^i_j$ is not defined, then $state^i_j \leftarrow (\mathsf{upk}_i, \mathsf{usk}_i, \mathsf{bpk})$. Thereafter a call is made to $\mathsf{UWithdraw}(\mathsf{msg}, state^i_j)$ with output $(\mathsf{msg}', state^i_j)$. The message msg' is returned, and $state^i_j$ is stored for use in subsequent calls to the oracle. After the session has finished, $state^i_j$ is parsed as a coin user secret data cusd^i_j. The key set for user i is updated $\mathsf{CSK}_i \leftarrow \mathsf{CSK}_i \cup \{\mathsf{cusd}^i_j\}$.

$\mathsf{HonestBWithdraw}(j, \mathsf{msg})$ executes one step of withdrawal session j for $\mathcal{B}$. More precisely, if session j has not been instantiated, i.e., $state_j$ is not defined, then $state_j \leftarrow (\mathsf{bpk}, \mathsf{bsk})$. Then a call is made to $\mathsf{BWithdraw}(\mathsf{msg}, state_j)$ with output $(\mathsf{msg}', state_j)$. The message msg' is returned, and $state_j$ is stored. After the session has finished, $state_j$ is parsed as a coin cpd and returned. Each time a coin is returned the counter l is incremented.

$\mathsf{HonestSpend}(\mathsf{cpd}, i, j, \mathsf{mid}, \mathsf{tid})$ spends cpd on behalf of $\mathcal{U}_j$ using the secret key from withdrawal session j. The oracle first checks if the secret data from withdrawal session j, cusd^i_j, has been stored in CSK_i and returns $\perp$ if this is not the case. Then it checks if (i, cpd) has been stored by the oracle and sets $\mathsf{ds}_i \leftarrow \mathsf{ds}_i + 1$ if this is the case. Then it stores (i, cpd), calls $\mathsf{Spend}(\mathsf{cpd}$, upk_i, usk_i, cusd^i_j, mid, tid, $\mathsf{bpk})$, and returns the output.

Concurrency

The adversary is given oracle access to the withdrawal protocol without any restrictions on how to access it. In particular it may execute several sessions in parallel. Therefore the scheme must be secure also under concurrent use to pass our definition.

Unforgeability

The property of *unforgeability* informally says that one cannot create valid coins by other means than withdrawing them from the bank. A little more precisely it says that if a coalition of users spend more than they have legally withdrawn, then at least one of them will get caught as a double-spender. Recall that l is the number of withdrawn coins using the withdrawal oracle. Unforgeability corresponds to the property *balance* of [25].

Experiment 5.2.2 (Unforgeability, $\mathbf{Exp}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa)$)**.**

```
$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$
$(\mathsf{spentcoin}_1, \ldots, \mathsf{spentcoin}_k) \leftarrow A^{\mathsf{AddCorruptU}(\cdot),\mathsf{HonestBWithdraw}(\cdot,\cdot)}(\mathsf{bpk})$
if $k \leq l$ then
  return 0
end if
if $\exists i \in [1, k] : \mathsf{VfCoin}(\mathsf{spentcoin}_i, \mathsf{bpk}) = 0$ then
  return 0
end if
if $\exists (i, j) \in [1, k]^2 : \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) \in \mathfrak{C}$ then
  return 0
end if
return 1
```

In the above experiment, there is no method for creating honest user. If there is an adversary which would benefit from this, it could as well create the key pair itself and run $\mathsf{AddCorruptU}$. Coins for the honest user could be withdrawn by playing the user part of the withdrawal protocol honestly.

Definition 5.2.2 (Unforgeability)**.** *A scheme for electronic cash $\mathcal{EC}$ has* unforgeability *if the advantage*

$$\mathbf{Adv}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa) = 1]$$

is negligible as a function of κ for any adversary $A \in \mathrm{PT}^*$.

Non-Frameability

A potential problem could be that a coalition of users and possibly the bank could accuse an honest user of double-spending. We say that a scheme has *non-frameability* if it is infeasible to frame an honest user in such a way.

In experiment below, $\mathcal{DS}$ is the set of (indices of) double-spent coins that implicate the framed $\mathcal{U}_j$. The adversary wins if it creates more double-spendings than ds_j, the number of double-spendings the adversary has made on behalf of $\mathcal{U}_j$ using the spending oracle. Intuitively this means that a user can only be accused of the actual number of double-spending she has performed.

Non-frameability corresponds to *strong exculpability* of [25]. The weak variant would guarantee that a user that has never double-spent cannot be accused of double-spending, but it would not prevent a double-spending user from being set up for additional double-spendings. Which variant to prefer is a matter of taste. One could argue that a user that double-spends has already breached her part of the contract, and it is not necessary to add complexity to the protocol to protect her. On the other hand, a double-spending could occur due to a technical malfunction rather than intentional misconduct, and in such a case it would be unreasonable for the protocol to allow an adversary to create additional double-spendings on behalf of the user. We choose the strong definition.

Since not even a dishonest bank should be able to frame a user, we allow the bank key to be chosen in an adversial way.

Experiment 5.2.3 (Non-Frameability, $\mathbf{Exp}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa)$)**.**

```
  $(\mathsf{bpk}, state) \leftarrow A(\mathsf{setup}, 1^\kappa)$
  $((\mathsf{spentcoin}_i, \mathsf{mid}_i, \mathsf{tid}_i)_{i=1}^{k}, j) \leftarrow$
    $A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, state)$
  **if** $\exists i : \mathsf{VfSpentCoin}(\mathsf{spentcoin}_i, \mathsf{mid}_i, \mathsf{tid}_i, \mathsf{bpk}) = 0$ **then**
    **return** 0
  **end if**
  $\mathcal{DS} \leftarrow \{i : \exists i' > i : \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_{i'}, \mathsf{bpk}) = \mathsf{upk}_j\}$
  **if** $|\mathcal{DS}| > \mathsf{ds}_j$ **then**
    **return** 1
  **else**
    **return** 0
  **end if**
```

We do not provide an oracle to add corrupt users to the PKI, since we are not interested in exposing honest users as double-spenders.

Definition 5.2.3 (Non-Frameability)**.** *A scheme for electronic cash $\mathcal{EC}$ has* non-frameability *if the advantage*

$$\mathbf{Adv}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa) = 1]$$

is negligible as a function of κ for any adversary $A \in \mathrm{PT}^*$.

Anonymity

Informally a scheme for electronic cash is *anonymous* if it is infeasible for any player, including the bank, to decide the identity of a spender. We define anonymity in a very strong sense, namely that not even knowing the private key of the spender helps revealing the identity of the user. We cannot, however, give the adversary the coin secret user data, since in such a case the adversary could itself double-spend the coin and reveal the identity. For the same reason the adversary may not use the $\mathsf{HonestSpend}$ oracle to double-spend one of the challenge coins.

In the experiment we let the adversary choose the bank public key and use oracles to create users and withdraw coins before it selects two coins, one of which will be spent as the challenge. Together with the challenge spentcoin the adversary is given the private keys of all users. This corresponds to the scenario where the private key of a user is exposed. The privacy of the user should be kept also in such a case.

If the keys were given to the adversary in the first stage, it could withdraw coins itself using the protocol, and it would trivially win the experiment by double-spending the challenge coins.

Experiment 5.2.4 (Anonymity, $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{EC},A}(\kappa)$)**.**

```
$(\mathsf{bpk}, state) \leftarrow A(\mathsf{setup}, 1^\kappa)$
$(i_0, j_0, i_1, j_1, \mathsf{mid}, \mathsf{tid}) \leftarrow$
    $A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{choose}, state)$
$\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{cpd}_{i_b}, \mathsf{usk}_{i_b}, \mathsf{cusd}^{i_b}_{j_b}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$
$d \leftarrow A^{\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, state, \mathsf{spentcoin}, (\mathsf{usk}_i)_{i=1}^{|\mathfrak{U}|})$
if $\exists \mathsf{mid}, \mathsf{tid} : (\{\mathsf{cpd}_{i_0}, i_0, j_0, \mathsf{mid}, \mathsf{tid}), (\mathsf{cpd}_{i_1}, i_1, j_1, \mathsf{mid}, \mathsf{tid})\} \cap \mathfrak{Q}_{\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot,\cdot)} \neq \emptyset$ then
    return 0
end if
if $d = b$ then
    return 1
else
    return 0
end if
```

Definition 5.2.4 (Anonymity)**.** *A scheme for electronic cash $\mathcal{EC}$ has* anonymity *if the advantage*

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{EC},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{EC},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{EC},A}(\kappa) = 1]|$$

is negligible as a function of κ for any adversary $A \in \mathrm{PT}^*$.

Exculpability

Exculpability states that the bank should not be able to create proofs of withdrawal, i.e., coins, which the user cannot spend. It should also not be able to produce more proofs than number of withdrawals made by the user.

Experiment 5.2.5 (Exculpability, $\mathbf{Exp}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa)$)**.**

```
$(\mathsf{bpk}, state) \leftarrow A(\mathsf{setup}, 1^\kappa)$
$(j, (\mathsf{cpd}_i)_{i=1}^k, \mathsf{mid}, \mathsf{tid}) \leftarrow$
    $A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, state)$
if $\exists i : \mathsf{VfCoin}(\mathsf{cpd}_i, \mathsf{upk}_j, \mathsf{bpk}) = 0$ then
    return 0
end if
if $k > |\mathsf{CSK}_j|$ then
```

```
    return 1
end if
if $\forall \mathsf{cusd} \in \mathsf{CSK}_j : \mathsf{Spend}(\mathsf{cpd}_1, \mathsf{upk}_j, \mathsf{usk}_j, \mathsf{cusd}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk}) = \perp$ then
    return 1
end if
return 0
```

There is no loss of generality in only checking if cpd_1 is spendable, since the adversary can always reorder the coins to output an unspendable coin first.

Definition 5.2.5 (Exculpability)**.** *A scheme for electronic cash $\mathcal{EC}$ has* exculpability *if the advantage*

$$\mathbf{Adv}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa) = 1]$$

is negligible as a function of κ for any adversary $A \in \mathrm{PT}^*$.

Finally we make the following definition.

Definition 5.2.6 (Secure Scheme for Electronic Cash)**.** *A scheme for electronic cash is* secure *if it has unforgeability, non-frameability, anonymity, and exculpability.*

Comparison to Group Signatures

Electronic cash resembles group signatures in many ways. We refer to Part III for a detailed description of group signatures.

Both group signatures and electronic cash allow users to perform transactions while remaining anonymous. On a high level, the roles of the bank and the group manager are similar. Withdrawing a coin has similarities to joining the group of a group signature scheme, and spending is in some ways similar to signing. The major difference in terms of anonymity is that a group signature can always be opened by the group manager, but a spent coin is anonymous also to the bank. In this sense, a scheme for electronic cash can be seen as a group signature scheme with one-time keys and unrevocable anonymity.

It is not surprising that the security properties of the two tasks are similar in many ways. Let us compare our definition for electronic cash to the definitions for dynamic group signatures in [57, 11].

Unforgeability

Our definition of unforgeability resembles the misidentification attack of [57] and traceability of [11].

Non-Frameability

Non-frameability is similar to the non-frameability property of both group signature definitions in that it requires that a user cannot be framed even if the complete system conspires against her. Although the adversary is given the secret keys of the group manager in [57, 11], our definition is stronger, since we allow the adversary to construct the key itself.

Anonymity

Anonymity of a group signature is different than that of a spent coin, since the opening key can always be used to open group signature. This is reflected in the experiments, which otherwise are quite similar.

Exculpability

The exculpability property is not defined for any group signature scheme to our knowledge. The corresponding property would be that a group manager cannot falsely claim that it has included a certain member into the group. A scenario where this might pose a problem is if group members are allowed to download certain information and there is a price to join the group. Group signatures do not address the potential issue when a member claims that the group manager has not issued him a key.

5.3 Security in the Framework for Universal Composability

We now consider the relation between experiment-based security of a scheme for electronic cash as defined above and security in the framework for universal composability (UC) [30]. We describe an ideal functionality, discuss why it captures the notion of anonymous electronic cash, and show that a scheme that is secure according to Definition 5.2.6 also securely realizes the ideal functionality. We use a model which is described in Section 2.2. The functionality described here has only one non-immediate function – the withdrawal protocol.

The ideal *anonymous electronic cash* functionality $\mathcal{F}_{\text{AnonEC}}$ running with parties $\mathcal{B}$, $\mathfrak{U} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_k\}$ is given below. The ideal adversary $\mathcal{S}$ corrupts a subset of the users and possibly the bank before the start-up. We let $\mathfrak{C}$ be the set of corrupted parties.

We do not differentiate between users and merchants. In fact, any party (except for the bank $\mathcal{B}$) can act both as a merchant and as a user. In addition we assume every user is uniquely identified with an identifier $\mathsf{mid} \in \{0,1\}^{\kappa/2}$.

The ideal functionality stores the following values.

T_{coins} is the table of coins that the bank has issued.

cc is the number of coins that have been withdrawn by corrupt users. It is initialized to 0.

$T_{\text{spent}-\text{coins}}$ contains the coins that have been honestly spent.

$T_{\text{valid}-\text{coins}}$ holds the coins that have been verified to be correct, but which have not been spent by honest users.

T_{double} are the coin pairs that have been determined to be double-spendings.

ds is the number of coins that have been determined to be double-spent by corrupt users. It is initialized to 0.

Functionality 5.3.1 (Anonymous Electronic Cash)**.**

1. Wait for the message $(\mathcal{S}, \texttt{Keys}, (\mathsf{bpk}, \mathsf{bsk}), (\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^{k})$ where $\mathsf{usk}_i = \perp$ if $\mathcal{U}_i \in \mathfrak{C}$ and $\mathsf{bsk} = \perp$ if $\mathcal{B} \in \mathfrak{C}$. Store $(\mathsf{bpk}, \mathsf{bsk})$ and $(\mathsf{upk}_i, \mathsf{usk}_i)$.

2. Then handle incoming messages as follows.

   - **Withdraw.** Upon reception of $(\mathcal{B}, \texttt{AccWithdrawal}, \mathcal{U}_i)$ act as follows.
     - If $\mathcal{U}_i \notin \mathfrak{C}$, then compute $(\mathsf{cpd}, \mathsf{cusd}) \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{upk}_i, \mathsf{usk}_i)$. Store $(\mathcal{U}_i, \mathsf{cpd}, \mathsf{cusd})$ in T_{coins}. Then hand the messages $((\mathcal{B}, \texttt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd}), (\mathcal{S}, \texttt{AccWithdrawal}, \mathcal{U}_i))$ to $\mathcal{C}_{\mathcal{I}}$.
     - If $\mathcal{U}_i \in \mathfrak{C}$, then hand $(\mathcal{S}, \texttt{AccWithdrawal}, \mathcal{U}_i)$ to $\mathcal{C}_{\mathcal{I}}$. Upon reception of response $(\mathcal{S}, \texttt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd})$, store $(\mathcal{U}_i, \mathsf{cpd}, \perp)$ in T_{coins}. Set $\mathsf{cc} \leftarrow \mathsf{cc} + 1$. Hand $(\mathcal{B}, \texttt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd})$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Verify Coin.** Upon reception of $(P, \texttt{VfCoin}, \mathsf{cpd}, \mathcal{U}_i)$, compute $b \leftarrow \mathsf{VfCoin}(\mathsf{cpd}, \mathsf{bpk}, \mathsf{upk}_i)$ and hand $(P, \texttt{VfCoin}, \mathsf{cpd}, \mathcal{U}_i, b)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Spend.** Upon reception of $(\mathcal{U}_i, \texttt{Spend}, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid})$, execute $(\cdot, \texttt{VfCoin}, \mathsf{cpd}, \mathcal{U}_i)$. If the result is 0, then hand $(\mathcal{U}_i, \texttt{Spend}, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid}, \perp)$ to $\mathcal{C}_{\mathcal{I}}$. Otherwise set $(\mathsf{upk}', \mathsf{usk}') \leftarrow \mathsf{UKg}(1^{\kappa})$, $(\mathsf{cpd}', \mathsf{cusd}') \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{upk}', \mathsf{usk}')$, $\mathsf{spentcoin}' \leftarrow \mathsf{Spend}(\mathsf{cpd}', \mathsf{usk}', \mathsf{cusd}', \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$. Store $(\mathcal{U}_i, \mathsf{cpd}', \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin}')$ in $T_{\text{spent}-\text{coins}}$.
   - **Verify Spent Coin.** Upon reception of $(P, \texttt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid})$ proceed as follows.
     - If $\mathcal{B} \in \mathfrak{C}$, then set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$.
     - If $(\cdot, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$ has been stored in $T_{\text{spent}-\text{coins}}$, then set $b \leftarrow 1$.
     - If $(\cdot, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin}) \notin T_{\text{spent}-\text{coins}}$, then do as follows.
       * If there exists $\mathcal{U}_j \in \mathfrak{C}$ such that $(\mathcal{U}_j, \mathsf{cpd}, \perp) \in T_{\text{coins}}$, then set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$. If $b = 1$, then store $(\mathcal{U}_j, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$ in $T_{\text{valid}-\text{coins}}$.
         For each $\mathsf{spentcoin} \in T_{\text{valid}-\text{coins}}$ proceed as follows. First compute $\mathsf{upk} \leftarrow \mathsf{VfDoubleSpent}(\mathsf{spentcoin}, \mathsf{spentcoin}', \mathsf{bpk})$ and let $\mathcal{U}_j$ be such that $\mathsf{upk}_j = \mathsf{upk}$. Insert $(\mathcal{U}_j, \{\mathsf{spentcoin}, \mathsf{spentcoin}'\})$

into T_{double} if $\mathcal{U}_j \in \mathfrak{C}$. Let $\mathsf{ds} \leftarrow \mathsf{ds} + 1$ if at least one such double-spending is found.
If $|T_{\text{valid-coins}}| - \mathsf{cc} > \mathsf{ds}$, then insert $(\mathcal{U}_j, \{\mathsf{spentcoin}, \mathsf{spentcoin}'\})$ into T_{double} for a random $\mathsf{spentcoin}' \in T_{\text{valid-coins}} \setminus \{\mathsf{spentcoin}\}$ and $\mathcal{U}_j \in \mathfrak{C}$ and set $\mathsf{ds} \leftarrow \mathsf{ds} + 1$.

∗ If no such user exists, then set $b \leftarrow 0$.

Hand $(P, \texttt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, b)$ to $\mathcal{C}_{\mathcal{I}}$.

- **Find Double-Spender.** Upon reception of a message $(P, \texttt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$, proceed as follows.
  - If $(\mathcal{U}_j, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin}_1)$ and $(\mathcal{U}_j, \mathsf{cpd}, \mathsf{mid}', \mathsf{tid}', \mathsf{spentcoin}_2)$ for $(\mathsf{mid}, \mathsf{tid}) \neq (\mathsf{mid}', \mathsf{tid}')$ exist in $T_{\text{spent-coins}}$, then set $\mathcal{U} \leftarrow \mathcal{U}_j$.
  - If $(\mathcal{U}_j, \{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}) \in T_{\text{double}}$, then let $\mathcal{U} \leftarrow \mathcal{U}_j$.
  - Otherwise let $\mathcal{U} \leftarrow \perp$.
    Hand $(P, \texttt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathcal{U})$ to $\mathcal{C}_{\mathcal{I}}$.

About the Functionality

Let us discuss why $\mathcal{F}_{\text{AnonEC}}$ captures what one would expect from a secure scheme for anonymous electronic cash.

Withdrawal

For an honest user, the coin is created as in the protocol to ensure correct distribution. The coin is stored in the table for withdrawn coins T_{coins} and returned to the bank. For a corrupt user, the bank engages in the withdrawal protocol (via the simulator). If the result is indeed a coin, then it is stored in the coins table and the counter for coins withdrawn by corrupt users is incremented.

Coin Verification

Coins are deemed valid when the protocol says so. This may seem overly simplified, but since the `Spend` algorithms requires valid coins to be spendable, this covers what one would expect from a valid coin. By the correctness of $\mathcal{EC}$, honestly withdrawn coins always pass the verification.

Spending

Before a coin can be spent, it is verified that it is valid and the spender owns the coin. If so, then a spent coin is created by creating a new user, withdrawing a coin, and spending it. Thus the spent coin has no information about the owner to ensure anonymity. The coin is stored in the table for spent coins $T_{\text{spent-coins}}$.

Verification of Spent Coin

If the bank is honest and the coin exists in the table for spent coins, then it is valid. If it does not exist in the table, it may still be valid, but only if it has been spent by a corrupt party and would implicate a corrupt party if double-spent. This is handled by including the spent coin in the list of potential double-spending by corrupt parties.

If the bank is corrupt, then any coin deemed valid by the protocol is accepted.

Identification of Double-Spenders

The algorithm is constructed so that it may only point out an honest user if it has actually double-spent a coin by sending a Spend command twice for the same coin.

The algorithm also ensures that if more coins are spent than withdrawn by corrupt parties, then a double-spender will be revealed. If the two coins have been spent by corrupt parties, then they may only be cleared from double-spending if there are enough potential double-spendings to cover the surplus of spent coins against withdrawn coins. As a special case a double-spending is never required to be exposed if the corrupt parties have not spent more coin than they have withdrawn.

On the Possibility of Simplifying the Functionality

The functionality $\mathcal{F}_{\text{AnonEC}}$ is rather complex, and it is natural question to ask whether it could be simplified. Let us consider how double-spenders are identified. Let $\mathcal{Z}$ be an environment proceeding as follows:

- $\mathcal{Z}$ runs with one corrupt user $\mathcal{U}_1$.
- $\mathcal{U}_1$ withdraws three coins.
- $\mathcal{U}_1$ spends four times, creating $\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{spentcoin}_3, \mathsf{spentcoin}_4$.

When only three coins have been spent, the functionality does not need to intervene if no double-spending is detected. If the fourth coin does not reveal a double-spending, then the functionality forces a double-spending to be reported. By using the T_{double} table, it is ensured that further queries are answered in a consistent way.

As the above example suggests, the functionality needs to be keep track of which coins have been reported as double-spendings, and which have not. It also needs to determine whether to many coins have been spent, forcing a double-spending if the number of spent coins exceeds the number of withdrawn coins. It seems that the functionality needs to be fairly complex.

The Real Protocol π_{AnonEC}

We describe how the protocol π_{AnonEC} is built from the algorithms of the scheme.

The Bank
The bank $\mathcal{B}$ generates $(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$ and broadcasts bpk. Then it waits for a message $(\mathtt{Keys}, \mathsf{upk}_i)$ for every user $\mathcal{U}_i$. Incoming messages are handled as follows:

- Upon reception of $(\mathtt{AccWithdrawal}, \mathcal{U}_i)$, the bank engages in the withdrawal protocol with $\mathcal{U}_i$. After the protocol has terminated,the bank outputs the tuple $(\mathtt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd})$.

Users
The user $\mathcal{U}_i$ generates $(\mathsf{upk}_i, \mathsf{usk}_i) \leftarrow \mathsf{UKg}(1^\kappa)$ and broadcasts upk_i. Then it waits for a message $(\mathtt{Keys}, \mathsf{upk}_j)$ for every user $\mathcal{U}_j$ and $(\mathtt{Keys}, \mathsf{bpk})$ from $\mathcal{B}$. Incoming messages are handled as follows:

- When challenged in the withdrawal protocol, run it according to the algorithm $\mathsf{UWithdraw}$. After the protocol has terminated, store the output cusd_j.
- Upon reception of $(\mathtt{Spend}, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid})$, set $\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{upk}_i, \mathsf{usk}_i, \mathsf{cusd}_j, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$ for the corresponding coin secret data cusd_j. Output $(\mathtt{Spend}, \mathsf{cpd}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$.

All Parties
Incoming messages are handled as follows:

- Upon reception of the message $(\mathtt{VfCoin}, \mathsf{cpd}, \mathcal{U}_j)$, set $b \leftarrow \mathsf{VfCoin}(\mathsf{cpd}, \mathsf{upk}_j, \mathsf{bpk})$ and return $(\mathtt{VfCoin}, \mathsf{cpd}, \mathcal{U}_j, b)$.
- Upon reception of the message $(\mathtt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid})$, set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$ and return $(\mathtt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, b)$.
- Upon reception of the message $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$, compute $\mathsf{upk} \leftarrow \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$. If $\mathsf{upk} = \perp$, then return $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \perp)$. Otherwise let $\mathcal{U}_j$ be such that $\mathsf{upk}_j = \mathsf{upk}$, and return $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathcal{U}_j)$

Proof of Security

Theorem 5.3.1. *Let $\mathcal{EC} = (\mathsf{BKg}, \mathsf{UKg}, \mathsf{UWithdraw}, \mathsf{BWithdraw}, \mathsf{VfCoin}, \mathsf{Spend}, \mathsf{VfSpentCoin}, \mathsf{VfDoubleSpent})$ be a secure scheme for anonymous electronic cash according to Definition 5.2.6. Then π_{AnonEC} securely realizes $\mathcal{F}_{\text{AnonEC}}$.*

Proof. Defining the Hybrids. We prove the theorem with a hybrid argument. We build a polynomial-size chain of protocols π_1^0, π_1^1, …, π_1^m, π_2^0, π_2^1, …, π_2^m, π_3^0, $\pi_3^1, \ldots, \pi_4^m$ such that $\pi_0^1 = \mathcal{F}_{\text{AnonEC}}$ and $\pi_4^m = \pi_{\text{AnonEC}}$. Then we show that if there exists an adversary A which can distinguish between π_t and π_{t+1} for some t,

$$\begin{tikzcd}
\pi_0^0 \arrow[r] & \pi_1^0 \arrow[r] & \cdots \arrow[r] & \pi_{m-1}^0 \arrow[r] & \pi_m^0 \arrow[dllll] \\
\pi_0^1 \arrow[r] & \pi_1^1 \arrow[r] & \cdots \arrow[r] & \pi_{m-1}^1 \arrow[r] & \pi_m^1 \arrow[dllll] \\
\vdots & \vdots & \ddots & \vdots & \vdots \arrow[dllll] \\
\pi_0^5 \arrow[r] & \pi_1^5 \arrow[r] & \cdots \arrow[r] & \pi_{m-1}^5 \arrow[r] & \pi_m^5
\end{tikzcd}$$

Figure 5.1: The chain of hybrid protocols.

then A can be used to break the security of $\mathcal{EC}$. For simplicity we assume all chains to be of the same length, which can always be arranged by padding. The chain built in this way is shown in Figure 5.1.

1. Let π_1^0 be $\mathcal{F}_{\text{AnonEC}}$. We define π_1^t to be π_1^{t-1} with the difference that the tth call to `Spend` produces a spent coin according to the protocol rather than using a dummy user as in the functionality.

2. Let $\pi_2^0 = \pi_1^m$. We define π_2^t to be π_2^{t-1} with the difference that the tth call to `Spend` there is no call to `VfCoin` before the spent coin is constructed.

3. Let $\pi_3^0 = \pi_2^m$, and define π_3^t to be π_3^{t-1} with the difference that the tth call to `VfSpentCoin` returns the $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$ rather than the value stipulated by the functionality. The table are manipuliated according to the functionality.

4. Let $\pi_4^0 = \pi_3^m$, and define π_4^t to be π_4^{t-1} with the difference that the tth call to `VfDblSpent` is executed according to the protocol rather than to the functionality and no table are manipulated in `VfSpentCoin`..

Building the Simulator. By assumption $\mathcal{Z}$ distinguishes between $\mathcal{F}_{\text{AnonEC}}$ and π_{AnonEC} for any ideal adversary. In particular it distinguishes between the two protocols for the adversary defined as follows.

For each player P_i that the real-world adversary A corrupts, the ideal adversary $\mathcal{S}$ corrupts the corresponding dummy player $\tilde{P}_i$. When a corrupted dummy player $\tilde{P}_i$ receives a message m from $\mathcal{Z}$, the simulator $\mathcal{S}$ lets $\mathcal{Z}'$ send m to P_i. When a corrupted P_i outputs a message m to $\mathcal{Z}'$, then $\mathcal{S}$ instructs the corrupted $\tilde{P}_i$ to output m to $\mathcal{Z}$. This corresponds to P_i being linked directly to $\mathcal{Z}$.

The simulated real-world adversary A is connected to $\mathcal{Z}$, i.e., when $\mathcal{Z}$ sends m to $\mathcal{S}$, $\mathcal{Z}'$ hands m to A, and when A outputs m to $\mathcal{Z}'$, $\mathcal{S}$ hands m to $\mathcal{Z}$. All non-corrupted players are simulated honestly. The corrupted players run according to their respective protocols.

When all parties have broadcasted their keys, $\mathcal{S}$ inspects the internal state of honest parties and intercepts the broadcast of corrupt parties to construct $(\mathsf{bpk}, \mathsf{bsk})$ and $(\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^{k}$ where $\mathsf{bsk} = \perp$ and $\mathsf{usk}_i = \perp$ only if $\mathcal{B}$ and $\mathcal{U}_i$, respectively, is corrupt. It hands $(\mathcal{F}_{\text{AnonEC}}, \mathtt{Keys}, (\mathsf{bpk}, \mathsf{bsk}), (\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^{k})$ to $\mathcal{C}_{\mathcal{I}}$.

When $\mathcal{S}$ receives the message $(\mathtt{AccWithdrawal}, \mathcal{U}_i)$, it instructs $\mathcal{Z}'$ to send $(\mathtt{AccWithdrawal}, \mathcal{U}_i)$ to $\mathcal{B}$. If $\mathcal{U}_i \in \mathfrak{C}$, then on output $(\mathtt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd})$ from $\mathcal{B}$, $\mathcal{S}$ hands $(\mathcal{F}_{\text{AnonEC}}, \mathtt{IssuedNewCoin}, \mathcal{U}_i, \mathsf{cpd})$ to $\mathcal{C}_{\mathcal{I}}$. All other functions are local and need not be simulated for A.

We now handle the cases when $\mathcal{Z}$ distinguishes between π_i^t and π_i^{t+1} for $i = 1, 2, 3, 4$.

1. Assume $\mathcal{Z}$ can distinguish between π_1^t and π_1^{t+1} with non-negligible probability. Then we construct A_{anon} breaking the anonymity of $\mathcal{EC}$ as follows. A_{anon} runs in Experiment 5.2.4 while simulating the protocol to $\mathcal{Z}$ as

   by using its $\mathsf{HonestUKg}$ oracle to set up keys for honest users, interacts with $\mathsf{HonestBWithdraw}$ to simulate withdrawals, and uses the $\mathsf{HonestSpend}$ oracles to construct spent coins. Let the $(t+1)$th $\mathtt{Spend}$ request be on behalf of user $\mathcal{U}_i$ for data $(\mathsf{mid}, \mathsf{tid})$. When executing Experiment 5.2.4, A_{anon} requests a spent coin by either $\mathcal{U}_i$ or a user $\mathcal{U}_j$, which has never before spent a coin. The challenge coin $\mathsf{spentcoin}$ is returned on the $\mathtt{Spend}$ request.

   If the challenge coin is by $\mathcal{U}_i$, then A_{anon} has run π_1^{t+1}, and if the coin is by $\mathcal{U}_j$, then the protocol simulated is π_1^t. Since, by assumption $\mathcal{Z}$ can distinguish between the two, A_{anon} wins the anonymity experiment with non-negligible probability.

2. Assume $\mathcal{Z}$ can distinguish between π_2^t and π_2^{t+1} with non-negligible probability.

   We construct A_{exculp} breaking the exculpability property of $\mathcal{EC}$ by running in Experiment 5.2.5 and simulating the protocol for $\mathcal{Z}$. A_{exculp} uses its oracles to create keys for the users, withdraw coins, and create spent coins. Since $\mathcal{Z}$ can distinguish between π_2^t and π_2^{t+1}, the coin to be spent in call $t+1$ does not pass the $\mathtt{VfCoin}$, but can still be spent. Then A_{exculp} outputs this coin in the guess phase of the experiment. Since the coin cannot be spent, A_{exculp} wins the experiment with non-negligible probability.

3. Assume $\mathcal{Z}$ can distinguish between π_3^t and π_3^{t+1} with non-negligible probability. We can assume $\mathcal{B} \notin \mathfrak{C}$, since otherwise $\mathtt{VfSpentCoin}$ is run identically in the protocol and the functionality. By the correctness of $\mathcal{EC}$, a coin issued by the bank and honestly spent is always accepted.

By the construction of the functionality, a $\mathsf{spentcoin}$ created by corrupt user will be detected as a double-spending if more coins are spent than has been withdrawn. Therefore $\mathcal{Z}$ can distinguish between the protocol and the functionality only if the tth call to Spend will be revealed as a double-spending by the functionality but not by the protocol.

We use $\mathcal{Z}$ to break the unforgeability of $\mathcal{EC}$ as follows. We construct A_{unforge} running in Experiment 5.2.2. The keys of the users are created honestly and then "registered" using the $\mathsf{AddCorruptU}$ oracle. Withdrawals are simulated by interacting with the $\mathsf{HonestBWithdraw}$ oracle. When asked to output forged coins, it outputs $T_{\text{valid}-\text{coins}}$. By the assumption, the table contains more coins than have been withdrawn, thus breaking the unforgeability property of $\mathcal{EC}$.

4. Assume $\mathcal{Z}$ can distinguish between π_4^t and π_4^{t+1} with non-negligible probability.

   For double-spendings that point out a corrupt user as double-spender, the protocol and the functionality are identical. Therefore $\mathcal{Z}$, if able to distinguish between the functionality and the protocol, has found $(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$ such that the functionality does not consider them a double-spending, but the protocol points out an honest party as double-spender.

   We let $A_{\mathsf{non-frame}}$ interact in Experiment 5.2.3 and simulate the protocol for $\mathcal{Z}$ as follows. The bank keys are constructed honestly and the keys of the honest users are created using the $\mathsf{HonestUKg}$ oracle. Withdrawals are performed by interacting with the $\mathsf{HonestUWithdraw}$ oracle, and spent coins are constructed with the $\mathsf{HonestSpend}$ oracle. By construction of the functionality, $\mathcal{Z}$ has found $(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$ such that they were not both constructed using the Spend, but still form a double-spending. $A_{\mathsf{non-frame}}$ breaks the non-frameability of $\mathcal{EC}$ by outputting this pair.

As shown, for each hybrid pair we can construct an adversary breaking a security assumption of $\mathcal{EC}$. Therefore it follows that if $\mathcal{EC}$ is secure, π_{AnonEC} securely realizes $\mathcal{F}_{\text{AnonEC}}$. □

5.4 A Construction

In this section we describe a secure scheme for electronic cash based on general methods. We first define the primitives, then we give the algorithms, and finally we prove that our scheme is secure according to our definition.

Common Reference String Model

Our model is secure in the Common Reference String (CRS) model. In this model every player has access to a random string. The string is chosen at a setup phase which is not discussed explicitly.

Primitives

Our construction uses a signature scheme $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$, a commitment scheme $\mathcal{COM} = (\mathsf{Commit}, \mathsf{Reveal})$, and simulation sound non-interactive zero-knowledge proofs of knowledge (NIZK-PK). We refer to Chapter 2 for precise definitions of these well-known concepts.

We need NIZKs for languages on the form $L = \{x \in \mathrm{Im}(f)\}$. Here the obvious witness relation is $R = \{(x, w) : f(w) = x\}$. Adapting the notation of [23], we use the notation $\mathrm{NIZK}(\omega : f(\omega) = x)$ to denote a NIZK of such a relation. We use Greek letters to denote variables in the witness, i.e., known only to the prover, and Latin letters for variables known both to the prover and to the verifier. We denote the verification algorithm by Vf. It will be clear from context for which relation the proof is.

The Protocol

Here we give the definitions for algorithms and protocols that form a scheme for electronic cash in the CRS-model. We begin by giving an informal description. In order to identify double-spenders, we use Ferguson's [44] trick of letting each coin contain a line $y = ax + \mathsf{upk}$, such that the coordinate of its intersection with the y-axis coincides with the identity upk of the owner. When spending the coin, one point on the line is revealed. Thus one spending of a coin gives no information about its owner. However, since we make sure different spendings reveal different points, the identity can be computed from two spendings of the same coin.

When withdrawing a coin, the user randomly selects the slope a. It computes a commitment $\tilde{a}$ of a and a commitment $\hat{a}$ of $\tilde{a}$ with associated randomness r_a and $r_{\tilde{a}}$.

$$\begin{tikzcd}
a \arrow[r] \arrow[dr] & \tilde{a} \arrow[r] \arrow[dr] & \hat{a} \\
& r_a & r_{\tilde{a}}
\end{tikzcd}$$

A two-step commitment $\hat{b}$ of the user public key upk is also computed. Then $\hat{a}, \hat{b}$ is sent to the bank and signed, and when a coin is spent, $\tilde{a}$ is revealed together with a proof of knowledge that it is correctly formed, i.e., that it is indeed the middle element of a two-step commitment of a and the user knows the associated randomness and bank signature. Intuitively this gives anonymity, since $\tilde{a}, \tilde{b}$ cannot be linked to $\hat{a}, \hat{b}$. It also assures that double-spenders are detected, since it is infeasible to open $\hat{a}, \hat{b}$ in more than one way. It can be noted that the signing mechanism is similar to the blind signature scheme found by Fischlin [46].

We let $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$ be a CMA-secure signature scheme and we let $\mathcal{COM} = (\mathsf{Commit}, \mathsf{Reveal})$ be a binding and hiding commitment scheme. Such signature schemes and commitment schemes exist if one-way functions exist [76, 49], and thus certainly if trapdoor permutations exist.

We use NIZKs for two different relations in the withdrawal and the spending protocols. The NIZKs work in the common reference string model. We will denote the reference string ξ. Each proof system needs its own CRS, so we divide ξ into two parts so that $\xi = \xi_0||\xi_1$ such that both ξ_0 and ξ_1 are long enough. We let $S(\mathsf{setup}, 1^\kappa)$ create both ξ_0 and ξ_1, store the two secrets in $\mathsf{simstate}$, and return $\xi_0||\xi_1$. Now S can simulate and extract proofs for both relations.

We start with the key generation algorithms. Key generation for the bank consists of generating a key for the signature scheme. The key for the user is created by drawing a value at random and computing a commitment to the value. The private key is the random value and coin tosses used in the commitment, and the public key is the commitment. As explained above, we do not include user registration at the bank as part of the protocol.

Definition 5.4.1 ($\mathsf{BKg}(1^\kappa)$)**.**

```
$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{SSKg}(1^\kappa)$
**return** $(\mathsf{bpk}, \mathsf{bsk})$
```

Definition 5.4.2 ($\mathsf{UKg}(1^\kappa)$)**.**

```
$t \leftarrow_R \{0,1\}^\kappa$
$(\mathsf{upk}, r_t) \leftarrow \mathsf{Commit}(t)$
$\mathsf{usk} \leftarrow (t, r_t)$
**return** $(\mathsf{upk}, \mathsf{usk})$
```

Merchant registration is straight-forward. Since our protocol does not use a merchant secret key, registration simply consists of handing the merchant identity to the bank, which registers the merchant.

The coin withdrawal protocol is a two-round protocol with the following steps.

1. The user draws a value a at random and commits to a in two steps, i.e., it computes a commitment $\tilde{a}$ to a, and a commitment $\hat{a}$ to $\tilde{a}$. In the same way it computes a two-step commitment $\hat{b}$ to its public key upk. It also constructs a proof $\pi_{\mathcal{U}}$ of knowledge of a a and coin tosses used in the commitments. It hands $\hat{a}, \hat{b}$ and $\pi_{\mathcal{U}}$ to the bank and stores a together with the coin tosses as the coin user secret data cusd.

2. The bank verifies that the user is allowed to withdraw a coin and that the proof of knowledge is valid. It then signs the user's public key concatenated with $(\hat{a}, \hat{b})$. The coin consists of the signature, $\hat{a}$,$\hat{b}$, the user's public key upk, and the proof $\pi_{\mathcal{U}}$.

More precisely, the withdrawal protocol consists of the following two algorithms.

Definition 5.4.3 ($\mathsf{UWithdraw}(\mathsf{msg}, \mathit{state})$)**.**

```
*Parse state as* $(\mathsf{upk}, \mathsf{usk})$.
$a \leftarrow_R \{0,1\}^\kappa$
$(\tilde{a}, r_a) \leftarrow \mathsf{Commit}(a)$
```

```
$(\hat{a}, r_{\tilde{a}}) \leftarrow \mathsf{Commit}(\tilde{a})$
$(\tilde{b}, r_{\mathsf{upk}}) \leftarrow \mathsf{Commit}(\mathsf{upk})$
$(\hat{b}, r_{\tilde{b}}) \leftarrow \mathsf{Commit}(\tilde{b})$
$\pi_{\mathcal{U}} \leftarrow \mathrm{NIZK}(\alpha, \rho_\alpha, \tilde{\alpha}, \rho_{\tilde{\alpha}}, \tau, \rho_\tau, \rho_{\mathsf{upk}}, \tilde{\beta}, \rho_{\tilde{\beta}} :$
  $\mathsf{Reveal}(\tilde{a}, \alpha, r_\alpha) = 1 \wedge \mathsf{Reveal}(\hat{a}, \tilde{\alpha}, \rho_{\tilde{\alpha}}) = 1 \wedge \mathsf{Reveal}(\mathsf{upk}, \tau, \rho_\tau) = 1 \wedge$
  $\mathsf{Reveal}(\tilde{\beta}, \mathsf{upk}, \rho_{\mathsf{upk}}) = 1 \wedge \mathsf{Reveal}(\hat{b}, \tilde{\beta}, \rho_{\tilde{\beta}}) = 1)$
**return** $((a, \tilde{a}, r_a, r_{\tilde{a}}, \tilde{b}, r_{\mathsf{upk}}, r_{\tilde{b}}), (\mathsf{upk}, \hat{a}, \hat{b}, \pi_{\mathcal{U}}))$
```

Definition 5.4.4 ($\mathsf{BWithdraw}(\mathsf{msg}, \mathit{state})$).

```
*Parse state as* $(\mathsf{bsk})$.
*Parse* $\mathsf{msg}$ *as* $(\mathsf{upk}, \hat{a}, \hat{b}, \pi_{\mathcal{U}})$.
*Quit if user with public key* $\mathsf{upk}$ *is not allowed to withdraw a coin.*
**if** $\mathsf{Vf}(\pi_{\mathcal{U}}) = 1$ **then**
  **return** $(\mathsf{reject}, \emptyset)$
**end if**
$s \leftarrow_R \mathsf{Sig}_{\mathsf{bsk}}(\mathsf{upk}, \hat{a}, \hat{b})$
$\mathsf{cpd} \leftarrow (s, \hat{a}, \hat{b}, \mathsf{upk}, \pi_{\mathcal{U}})$
**return** $(\mathsf{cpd}, \emptyset)$
```

We also need to be able to verify whether or not a coin has been withdrawn by a certain user by verifying the coin's signature and the user's proof of knowledge.

Definition 5.4.5 ($\mathsf{VfCoin}(\mathsf{cpd}, \mathsf{upk}, \mathsf{bpk})$).

```
*Parse* $\mathsf{cpd}$ *as* $(s, \hat{a}, \hat{b}, \mathsf{upk}, \pi_{\mathcal{U}})$.
**return** $\mathsf{Vf}_{\mathsf{bpk}}((\mathsf{upk}, \hat{a}, \hat{b}), s) \wedge \mathsf{Vf}(\pi_{\mathcal{U}})$
```

We define two coin public data $\mathsf{cpd} = (s, \hat{a}, \hat{b}, \mathsf{upk}, \pi_{\mathcal{U}})$ and $\mathsf{cpd}' = (s', \hat{a}', \hat{b}', \mathsf{upk}', \pi_{\mathcal{U}})$ to be equal if $\hat{a} = \hat{a}'$, $\hat{b} = \hat{b}'$, $\mathsf{upk} = \mathsf{upk}'$.

To spend a coin the user first checks that the coin is valid. Then it lets (x, y) be a point on the line $y = ax + \mathsf{upk}$, where a is the coin user secret data and upk the public key of the user. The point x is chosen as the concatenation of the transaction identity and the merchant identity. The user reveals the values $\tilde{a}$ and $\tilde{b}$. The spent coin consists of $(\tilde{a}, \tilde{b}, x, y)$, and a proof of knowledge of a and upk such that (x, y) is indeed a point on the line and of a bank signature on $(\mathsf{upk}, \hat{a}, \hat{b})$ as well as of coin tosses such that $\hat{a}$ is a commitment of $\tilde{a}$ and $\hat{b}$ of $\tilde{b}$.

Definition 5.4.6 ($\mathsf{Spend}(\mathsf{cpd}, \mathsf{usk}, \mathsf{cusd}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$).

```
*Parse* $\mathsf{cpd}$ *as* $(s, \hat{a}, \hat{u}, \mathsf{upk}, \pi_{\mathcal{U}})$
*Parse* $\mathsf{usk}$ *as* $(t, r_t)$
*Parse* $\mathsf{cusd}$ *as* $(a, \tilde{a}, r_a, r_{\tilde{a}}, \tilde{b}, r_{\mathsf{upk}}, r_{\tilde{b}})$
**if** $(\mathsf{Vf}_{\mathsf{bpk}}((\mathsf{upk}, \hat{a}, \hat{b}), s) = 0) \vee (\mathsf{Reveal}(\tilde{a}, a, r_a) = 0) \vee (\mathsf{Reveal}(\hat{a}, \tilde{a}, r_{\tilde{a}}) = 0) \vee$
$(\mathsf{Reveal}(\mathsf{upk}, t, r_t) = 0) \vee (\mathsf{Reveal}(\tilde{b}, \mathsf{upk}, r_{\mathsf{upk}}) = 0) \vee (\mathsf{Reveal}(\hat{b}, \tilde{b}, r_{\tilde{b}}) = 0)$ **then**
  **return** $\perp$
**end if**
```

```
$x \leftarrow \mathsf{mid}||\mathsf{tid}$
$y \leftarrow ax + \mathsf{upk}$
$\pi \leftarrow \mathrm{NIZK}(\iota, \alpha, \rho_\alpha, \hat{\alpha}, \rho_{\tilde{a}}, \rho_{\mathsf{upk}}, \hat{\beta}, \rho_{\tilde{b}}, \sigma, \tau, \rho_\tau :$
  $y = \alpha x + \iota \wedge \mathsf{Reveal}(\tilde{a}, \alpha, \rho_\alpha) = 1 \wedge \mathsf{Reveal}(\hat{\alpha}, \tilde{a}, \rho_{\tilde{a}}) = 1 \wedge \mathsf{Reveal}(\tilde{b}, \iota, \rho_{\mathsf{upk}}) \wedge$
  $\mathsf{Reveal}(\hat{\beta}, \tilde{b}, \rho_{\tilde{b}}) = 1 \wedge \mathsf{Vf}_{\mathsf{bpk}}((\iota, \tilde{\alpha}, \tilde{\beta}), \sigma) = 1 \wedge \mathsf{Reveal}(\iota, \rho_\tau, \tau) = 1)$
$\mathsf{spentcoin} \leftarrow (\tilde{a}, \tilde{b}, x, y, \pi)$
return spentcoin
```

Verification of a spent coin is straight-forward:

Definition 5.4.7 (VfSpentCoin(spentcoin,tid,mid,bpk))**.**

```
Parse spentcoin as $(\tilde{a}, \tilde{b}, x, y, \pi)$.
if $x \neq \mathsf{mid}||\mathsf{tid}$ then
  return 0
end if
return $\mathsf{Vf}(\pi)$
```

Finally we give the algorithm to identify a double-spender. A coin is double-spent if the values $(\tilde{a}, \tilde{b})$ appears twice with different values of x. Finding the double-spender is then simply a task of solving the two equations for upk.

Definition 5.4.8 ($\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$)**.**

```
Parse $\mathsf{spentcoin}_1$ as $(\tilde{a}_1, \tilde{b}_1, x_1, y_1, \pi_1)$.
Parse $\mathsf{spentcoin}_2$ as $(\tilde{a}_2, \tilde{b}_2, x_2, y_2, \pi_2)$.
if $((\tilde{a}_1, \tilde{b}_1) \neq (\tilde{a}_2, \tilde{b}_2)) \vee (x_1 = x_2)$ then
  return $\perp$
end if
$\mathsf{upk} \leftarrow \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}$
return upk
```

5.5 Proof of Security

In this section we prove the following theorem about the scheme $\mathcal{EC}$ = (BKg, UKg, UWithdraw, BWithdraw, VfCoin, Spend, VfSpentCoin, VfDoubleSpent) as defined in Section 5.4.

Theorem 5.5.1. *If there exists a family of trapdoor permutations, then there exists a scheme for electronic cash which is correct and secure in the common reference string model.*

From Theorem 5.3.1 this implies the following.

Theorem 5.5.2. *If there exists a family of trapdoor permutations, then there exists a protocol which securely realizes $\mathcal{F}_{\mathrm{AnonEC}}$ in the CRS model.*

We prove the theorem by showing the five properties about the scheme defined in Section 5.4. Each lemma holds in the CRS-model under the assumption that a family of trapdoor permutations exists, although this is not stated explicitly.

Lemma 5.5.3 (Correctness)**.** *The scheme $\mathcal{EC}$ is correct.*

Proof. Follows by the construction of the algorithms. □

Lemma 5.5.4 (Unforgeability)**.** *The scheme $\mathcal{EC}$ has unforgeability.*

Proof. Let A be an adversary that is successful in Experiment 5.2.2 with non-negligible probability. We show how to use A to construct either a machine A_{cma} breaking the CMA-security of the signature scheme $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$, a machine A_{binding} breaking the binding property of the commitment scheme $\mathcal{COM}$, or a machine $A_{\mathsf{sim-sound}}$ breaking the simulation soundness of the NIZK-PK.

A_{cma} is given a public key pk for the signature scheme as input. It passes pk as parameter bpk to A. The CRS is created using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$. As in the experiment for CMA security, A_{cma} has access to a signature oracle. The $\mathsf{BWithdraw}$ oracle is run honestly using the signature $\mathsf{Sig}_{\mathsf{bsk}}(\cdot)$ produced by calling the signature oracle.

Let k be the number of $\mathsf{spentcoin}$ produced by A. Recall A has made l withdrawals using its oracle. This implies A_{cma} has made l calls to the CMA oracle. For each $\mathsf{spentcoin}_i = (\tilde{a}_i, \tilde{b}_i, x_i, y_i, \pi_i)$, A_{cma} calls $S(\mathsf{extract}, (\hat{a}_i, \hat{b}_i, x_i, y_i), \pi_i, \xi, \mathsf{simstate})$ to extract (among other parameters) σ_i, ι, $\hat{\alpha}$, $\hat{\beta}$, ρ_β such that $\mathsf{Vf}_{\mathsf{bpk}}((\iota, \hat{\alpha}, \hat{\beta}), \sigma) = 1$. We now have the following different cases:

1. Signatures on more than l distinct messages are extracted. Then there exists a message-signature pair $(\iota, \hat{\alpha}, \hat{\beta})$ for which no signature has been generated by the CMA oracle. Hence A_{cma} is successful in breaking the CMA-security of $\mathcal{SS}$ by returning $(\iota, \hat{\alpha}, \hat{\beta}), \sigma$.

2. At least one proof π_i cannot be extracted. In this case $A_{\mathsf{sim-sound}}$ uses π_i to break the extractable simulation soundness of the NIZK-PK in the following way. $A_{\mathsf{sim-sound}}$ takes part in Experiment 2.4.14 while running A. $A_{\mathsf{sim-sound}}$ creates the bank key pair honestly and answers queries to $\mathsf{HonestBWithdraw}$ honestly. When A has output $\mathsf{spentcoin}_i$ with the unextractable proof π_i, $A_{\mathsf{sim-sound}}$ returns $\mathsf{spentcoin}_i$, thus winning in its experiment.

3. All proofs can be extracted, but two proofs yield signatures on the same message $(\iota, \hat{\alpha}, \hat{\beta})$. Since no double-spending is detected, all $(\tilde{\alpha}, \tilde{\beta})$ are distinct. Hence there are two commitments with associated randomness $(\tilde{\alpha}_i, \rho_i)$ and $(\tilde{\alpha}_j, \rho_j)$ such that $\mathsf{Reveal}(\hat{\alpha}, \tilde{\alpha}_i, \rho_i) = \mathsf{Reveal}(\hat{\alpha}, \tilde{\alpha}_j, \rho_j) = 1$. A machine A_{binding} which lets the simulator generate the CRS, generates the bank keys honestly, answers $\mathsf{HonestBWithdraw}$ queries honestly wins Experiment 2.4.7, the binding experiment of the commitment scheme $\mathcal{COM}$, by extracting and outputting $(\hat{\alpha}, \tilde{\alpha}_i, \rho_i, \tilde{\alpha}_j, \rho_j)$.

Thus we have shown that an adversary which breaks unforgeability can be used to either break the CMA security of $\mathcal{SS}$, the extractable simulation soundness of the NIZK-PK, or break the binding property of $\mathcal{COM}$. Hence $\mathcal{EC}$ has unforgeability. □

Lemma 5.5.5 (Non-Frameability)**.** *The scheme $\mathcal{EC}$ has non-frameability.*

Proof. Let A be an adversary that succeeds in Experiment 5.2.3 with non-negligible probability. We show how to use A to construct either a machine A_{secrecy} breaking the secrecy property of the commitment scheme $\mathcal{COM}$, a machine A_{binding} breaking the binding property, or a machine $A_{\mathsf{ext-sim-sound}}$, which breaks the extractable simulation soundness of the NIZK-PK.

The machine A_{secrecy} takes part in Experiment 2.4.6. It creates a CRS using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$. It randomly draws two message msg_0 and msg_1 which it returns to its experiment, receiving a challenge commitment c. Let the polynomial $p(\kappa)$ be an upper bound on the number of calls to $\mathsf{HonestUKg}$ by A. Since A runs in polynomial time, there exists such a polynomial. A_{secrecy} randomly selects $t \in [1, p(\kappa)]$. Intuitively A_{secrecy} guesses that A will frame user $\mathcal{U}_t$. All queries to $\mathsf{HonestUKg}$ are executed honestly except for query t, to which A_{secrecy} responds c.

When A queries $\mathsf{HonestUWithdraw}$ or $\mathsf{HonestSpend}$ for a user different from $\mathcal{U}_t$, the query is answered honestly. For $\mathcal{U}_t$, the NIZK-PK is constructed by invoking the simulator $S(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})$.

First consider the case when A behaves differently on simulated and honest proofs. If this is the case, then we can construct $A_{\mathsf{ad-ind}}$ running in Experiment 2.4.11 or 2.4.12 as follows. $A_{\mathsf{ad-ind}}$ receives the CRS from its experiment. It runs A simulating all oracles honestly, except that the NIZK-PKs are constructed by requesting an honest or simulated proof from its experiment. Note that the view of A is identical to the view of A when used by A_{secrecy}. If A behaves differently on honest and simulated proofs, then $A_{\mathsf{ad-ind}}$ can distinguish between its two experiments, breaking the adaptive indistiguishability of the NIZK-PK.

A outputs a list of spent coins $(\mathsf{spentcoin}_i)_{i=1}^k$. Let $\mathsf{spentcoin}_i, \mathsf{spentcoin}_j$ be spent coins such that $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) \notin \mathfrak{C}$ and at least one spent coin has not been produced by $\mathsf{HonestSpend}$. Since A outputs more double-spent coins than was created by $\mathsf{HonestSpend}$, such a pair of spent coins exists by the pigeon-hole principle. Let $\mathsf{spentcoin}_i = (\tilde{a}_i, \tilde{b}_i, x_i, y_i, \pi_i)$ and $\mathsf{spentcoin}_j = (\tilde{a}_j, \tilde{b}_j, x_j, y_j, \pi_j)$. By the assumption that they form a double-spending, we have that $(\tilde{a}_i, \tilde{b}_i) = (\tilde{a}_j, \tilde{b}_j)$ and $x_i \neq x_j$. With probability $1/p(\kappa)$, i.e., non-negligible, it holds that $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) = \mathsf{upk}_t$. From now on, we will assume that this is the case.

From π_i and π_j the machine A_{secrecy} attempts to extract $(\bar{a}_i,\ \bar{r}_a^{(i)},\ \bar{\mathsf{upk}}_i,\ \bar{r}_{\mathsf{upk}}^{(i)},\ \bar{\mathsf{usk}}_i,\ \bar{r}_{\mathsf{usk}}^{(i)})$ and $(\bar{a}_j,\ \bar{r}_a^{(j)},\ \bar{\mathsf{upk}}_j,\ \bar{r}_{\mathsf{upk}}^{(j)}),\ \bar{\mathsf{usk}}_j, \bar{r}_{\mathsf{usk}}^{(j)})$ such that $\mathsf{Reveal}(\tilde{a}_i,\ \bar{a}_i, \bar{r}_a^{(i)}) = 1$, $\mathsf{Reveal}(\tilde{b}_i,\ \bar{\mathsf{upk}}_i,\ \bar{r}_{\mathsf{upk}}^{(i)}) = 1$, $\mathsf{Reveal}(\bar{\mathsf{upk}}_i,\ \bar{\mathsf{usk}}_i,\ \bar{r}_{\mathsf{usk}}^{(i)}) = 1$ and $\mathsf{Reveal}(\tilde{a}_j,\ \bar{a}_j,\ \bar{r}_a^{(j)}) = 1$, $\mathsf{Reveal}(\tilde{b}_j,\ \bar{\mathsf{upk}}_j,\ \bar{r}_{\mathsf{upk}}^{(j)}) = 1$, $\mathsf{Reveal}(\bar{\mathsf{upk}}_j,\ \bar{\mathsf{usk}}_j,\ \bar{r}_{\mathsf{usk}}^{(j)}) = 1$. We now have the following cases and subcases:

1. None of the proofs were created by the simulator.

a) At least one extraction fails. In such case we can construct a machine $A_{\mathsf{ext-sim-sound}}$ using A and breaking the extractable simulation soundness of the NIZK-PK as follows. $A_{\mathsf{ext-sim-sound}}$ takes part in Experiment 2.4.14. When A asks for a spent coin, $A_{\mathsf{ext-sim-sound}}$ uses its simulation oracle to form the NIZK-PK of the spent coin. The un-extractable NIZK-PK of A is output by $A_{\mathsf{ext-sim-sound}}$, which wins the extractable simulation soundness experiment with non-negligible probability.

b) Both extractions succeed but return $\bar{a}_i \neq \bar(a)_j$ or $\bar{\mathsf{upk}}_i \neq \bar{\mathsf{upk}}_j$. Let us assume the first inequality holds, since the other case is analogous. In such a case the extracted values can be used by the machine A_{binding} to break the binding property of $\mathcal{COM}$ by letting A_{binding} run A while generating the keys and simulating the oracle honestly and output $(\tilde{a}, \bar{r}_a^{(i)}, \bar{a}_i, \bar{r}_a^{(j)}, \bar{a}_j)$ in Experiment 2.4.7.

c) Both extractions succeed and return consistent values. Since the NIZK-PK also proves that $y_l = ax_l + \mathsf{upk}$, it follows that $\bar{\mathsf{upk}} = \mathsf{upk}_t$. The machine A_{secrecy} breaking the secrecy of $\mathcal{COM}$ by running in Experiment 2.4.6 is constructed as follows. Recall that $\mathsf{msg}_0, \mathsf{msg}_1$ are drawn by A_{secrecy} when genereating a key for $\mathcal{U}_t$. Now A_{secrecy} finds d such that $\mathsf{msg}_d = \bar{\mathsf{cusd}}$ and returns d. Since A is successful with non-negligible probabilty, the so is A_{secrecy}.

If no such d is found, then a machine A_{binding} wins in Experiment 2.4.7 in the following way. In runs as described with the difference that $(c, r) \leftarrow \mathsf{Commit}(\mathsf{usk})$. It then outputs $(\mathsf{upk}_t,\ \bar{\mathsf{cusd}},\ \bar{r}_{\mathsf{upk}},\ c,\ r)$, which forms a double opening of a commitment. Hence A_{binding} is successful with non-negligible probabilty.

2. One proof was created by the simulator. Without loss of generality we assume that the simulator created π_j, and let $a_j, r_a^{(j)}, \mathsf{upk}_j, r_{\mathsf{upk}}^{(j)}$ be the values used when responding to the oracle query.

a) The extraction of the proof π_i fails. If this is the case, then $A_{\mathsf{ext-sim-sound}}$ proceeds as in Step 1a to break the simulation soundness of the NIZK-PK.

b) The extraction of π_i succeeds but yields $(\bar{a}_i,\ \bar{r}_a^{(i)},\ \bar{\mathsf{upk}}_i,\ \bar{r}_{\mathsf{upk}}^{(i)}) \neq (a_j,\ r_a^{(j)},\ \mathsf{upk}_j, r_{\mathsf{upk}}^{(j)})$. Then, as in Step 1b, the binding property of $\mathcal{COM}$ is broken.

c) The extraction π_i succeeds and gives consistent values. Then, as in Step 1c, the secrecy of $\mathcal{COM}$ is broken.

3. Both proofs were created by the simulator. Since, by assumption, at least one coin was not created by an oracle query, this cannot happen.

We have shown that if A breaks the non-frameability property, then at least one of the machines A_{secrecy}, A_{binding}, and $A_{\mathsf{ext-sim-sound}}$ is successful with non-negligible probability. Since this breaks the assumption, the scheme $\mathcal{EC}$ has non-frameability.

□

Lemma 5.5.6 (Anonymity). *The scheme $\mathcal{EC}$ has anonymity.*

Proof. Assume A wins in the anonymity experiment 5.2.4 with non-negligible probability. We show how to construct either A_{secrecy} breaking the secrecy of the commitment scheme $\mathcal{COM}$ or a machine $A_{\mathsf{ad-ind}}$ breaking the adaptive indistinguishability of the NIZK-PK.

We define two variants of the scheme $\mathcal{EC}$. We let $\mathcal{EC}'$ be $\mathcal{EC}$ with the modification that the CRS is created by the simulator, $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$ and that the NIZK-PK in the Spend algorithm is generated by the simulator. We let $\mathcal{EC}''$ be $\mathcal{EC}'$ with the difference that the commitment scheme of UWithdraw used to produce $\tilde{a}$ is replaced by a commitment scheme with perfect secrecy.

Since a spentcoin in $\mathcal{EC}''$ contains no information about the spender of a coin, the advantage of A when attacking $\mathcal{EC}''$ is 0. We now have the following two cases.

1. The advantage of A when attacking $\mathcal{EC}'$ is non-negligible. We show how to use A to construct A_{secrecy} which successfully attacks the secrecy of the commitment scheme $\mathcal{COM}$. A_{secrecy} takes part in Experiment 2.4.6 while simulating Experiment 5.2.4 to A. All calls to HonestUKg and HonestSpend are answered honestly. When A outputs $(i_0, i_1, \mathsf{mid}, \mathsf{tid})$, A_{secrecy} outputs $(\mathsf{upk}_{i_0}, \mathsf{upk}_{i_1})$ to its experiment, receiving a commitment c in response. Then A_{secrecy} uses c as $\tilde{a}$ when creating the challenge spentcoin and constructs the rest of the coin honestly. (Since $\mathcal{EC}'$ only uses simulated NIZK-PKs, not knowing the message of c is not a problem.) A outputs a bit d, which A_{secrecy} outputs in its experiment.

   From the construction it follows that A_{secrecy} is successful when A is, and hence breaks the secrecy of $\mathcal{COM}$ with non-negligible probability.

2. The advantage of A when attacking $\mathcal{EC}'$ is negligible. In such case we can use A to construct $A_{\mathsf{ad-ind}}$ breaking the adaptive indistinguishability of the NIZK-PK. $A_{\mathsf{ad-ind}}$ takes part in Experiment 2.4.11 and 2.4.12 while executing Experiment 5.2.4 for A. All parts of Experiment 5.2.4 are executed honestly, except that NIZK-PKs of HonestSpend are created by requesting a proof for $A_{\mathsf{ad-ind}}$ in the choose phase. If A is successful, $A_{\mathsf{ad-ind}}$ responds that it is interacting with Experiment 2.4.11, and otherwise that it is interacting with Experiment 2.4.12. Since A is successful only when NIZK-PKs are genuine, $A_{\mathsf{ad-ind}}$ has a non-negligible advantage.

We have shown that a machine breaking the anonymity of $\mathcal{EC}$ can be made into a machine either breaking the secrecy of the commitment scheme or a machine breaking the adaptive indistinguishability of the proof system. Since such machines contradicts the assumptions, we conclude that $\mathcal{EC}$ has anonymity.

□

Lemma 5.5.7 (Exculpability). *The scheme $\mathcal{EC}$ has exculpability.*

Proof. Let A be an adversary which wins in Experiment 5.2.5 with non-negligible probability. Thus A either creates the coin public data which the owner cannot spend or creates a coin which has not been withdrawn. Let us consider the first case. We use A to construct either a machine A_{secrecy} breaking the secrecy property of the commitment scheme $\mathcal{COM}$, A_{binding} breaking the binding property of the $\mathcal{COM}$, or a machine $A_{\mathsf{ext-sim-sound}}$ breaking the simulation soundness of the NIZK-PK.

We define the scheme $\mathcal{EC}'$ being equal to $\mathcal{EC}$ with the difference that the CRS is setup using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$ and the NIZK-PK of $\mathsf{UWithdraw}$ is created using the simulator.

First assume A has negligible probability of breaking the exculpability property of $\mathcal{EC}'$. Then we can use A to construct $A_{\mathsf{ad-ind}}$ in the following way. $A_{\mathsf{ad-ind}}$ takes part in Experiment 2.4.11 or 2.4.12. It invokes A, answering all queries honestly except that the NIZK-PK is created by requesting a proof in the choose phase of $A_{\mathsf{ad-ind}}$. Hence, if $A_{\mathsf{ad-ind}}$ is run in Experiment 2.4.11, it will run $\mathcal{EC}$ for A, but if it is run in Experiment 2.4.12, it will run $\mathcal{EC}'$. If A is successful, then $A_{\mathsf{ad-ind}}$ returns 0, and otherwise it returns 1. From the construction of $A_{\mathsf{ad-ind}}$ it follows that it breaks the adaptive indistinguishability of the NIZK-PK.

Now assume A has non-negligible probability in winning the exculpability experiment against $\mathcal{EC}'$. We use A in a similar way, letting $(\xi, \mathsf{simstate})$ be constructed by the simulator. Proofs of knowledge for $\mathcal{U}_t$ are constructed using the simulator.

1. The NIZK-PK $\pi_{\mathcal{U}}$ of cpd output by A has been constructed by the simulator. This implies that $\pi_{\mathcal{U}}$ was created by $\mathsf{HonestUWithdraw}$ for a certain user and coin secret key $\mathsf{usk}_i, \mathsf{cusd}_i$. Hence the coin can be spent using $\mathsf{usk}_i, \mathsf{cusd}_i$, contradicting the assumption that the exculpability property is broken.

2. The NIZK-PK $\pi_{\mathcal{U}}$ of cpd output by A has not been constructed by the simulator. In this case we can construct A_{secrecy} breaking the secrecy of the commitment scheme as follows. Let $p(\kappa)$ be an upper bound on the number of calls to $\mathsf{HonestUKg}$. Since A is polynomial, such a bound exists. Let $t \leftarrow_R [1, p(\kappa)]$. Informally A_{secrecy} guesses that A will frame $\mathcal{U}_t$. A_{secrecy} randomly chooses τ_0, τ_1 and requests a challenge commitment c on one of them from its experiment. It answers queries honestly, except that when asked to generate the public key for $\mathcal{U}_t$, it returns the challenge commitment c as upk_t.

   With probability $1/p(\kappa)$ A produces a coin cpd that can be verified to belong to $\mathcal{U}_t$. Assume this is the case. Then A_{secrecy} uses the extractor to extract τ, r_τ such that $\mathsf{Reveal}(\mathsf{upk}_t, \tau, r_\tau) = 1$. We now have three cases.

   a) There exists d such that $\tau_d = \tau$. Then A_{secrecy} returns d and breaks the secrecy of the commitment scheme $\mathcal{COM}$ with non-negligible probability.

   b) No such d exists. Then two openings of commitment upk_t has been found, allowing us to construct A_{binding} breaking the binding property of $\mathcal{COM}$ as follows. A_{binding} runs A_{secrecy} (which in turn runs A) as above. When the challenge commitment c is created for τ_b, A_{binding} stores $c = \mathsf{upk}_t$ and the associated randomness r_c. After A_{secrecy} has extracted τ, r_τ, A_{binding} outputs $\tau_b, \mathsf{upk}_t, r_c, \tau, r_\tau$. Since $\tau \neq \tau_b$, A_{binding} breaks the binding property of $\mathcal{COM}$.

c) The extraction fails. Such an adversary A can be used by $A_{\mathsf{ext-sim-sound}}$ breaking the extractable simulation soundness of the NIZK-PK of the withdrawal protocol which is constructed as follows. $A_{\mathsf{ext-sim-sound}}$ runs in Experiment 2.4.14, using ξ as CRS. When A asks for a withdrawn coin, $A_{\mathsf{ext-sim-sound}}$ uses a simulated proof from its experiment, constructing the other parts of the coin honestly. The other oracles are simulated honestly. When the unextractable proof π is constructed, it is output by $A_{\mathsf{ext-sim-sound}}$, which then is successful in its experiment. Since the view of A is the same as in the above cases, the probability of A constructing such a proof is non-negligible.

Let us now consider the case where A outputs more coins than executions of the withdrawal protocol. Since we can assume that all coins can be spent, otherwise the first case would hold, two distinct coins $\mathsf{cpd}_1 = (s_1, \hat{a}_1, \hat{b}_1, \mathsf{upk}_1, \pi_1)$, $\mathsf{cpd}_2 = (s_2, \hat{a}_2, \hat{b}_2, \mathsf{upk}_2, \pi_2)$ can be spent with the same coin secret data cusd. By the definition of equal coins, $(\hat{a}_1, \hat{b}_1) \neq (\hat{a}_2, \hat{b}_2)$. Since the probability that two honestly created coins can be spend using the same cusd is negligible, with overwhelming probability at least one of the coins has been constructing without using the withdrawal oracle of A. Without loss of generality we let cpd_1 be this coin. Then case 2 above holds if we use cpd_1 in place of cpd.

We can conclude that a machine breaking the exculpability property of $\mathcal{EC}$ implies a machine breaking one of the assumptions. Therefore $\mathcal{EC}$ has exculpability.

□

5.6 Proof of Theorem 2.4.23

We prove now that every **NP**-language has a NIZK-PK by giving a construction and showing that it has the necessary properties.

Proof. (Theorem 2.4.23) We give a construction of an extractable simulation sound proof system based on an unbounded simulation sound proof system. The idea behind the construction is the same as for [80], which is also used in [33], namely to encrypt the witness using a semantically secure encryption scheme where the public key is derived from the common reference string. Extraction is performed by letting the extractor choose the CRS in such a way that it knows the private key.

Let L be a language with witness relation R, i.e., $x \in L$ exactly when there exists w such that $(x, w) \in R$. We define a proof system (P, V) with simulator S and prove that it is an unbounded simulation sound zero-knowledge proof of knowledge. Note that S plays the role both of the simulator and the extractor.

In all the below experiments, we let the common reference string ξ consist of two parts, ξ_0 and ξ_1, where ξ_0 is long enough to be used as CRS for a NIZK of [79]. We let *pk* be a public key for the encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$ of the appropriate length defined by ξ_1, and we let *sk* be the corresponding secret key. We also let $L'_{\xi_1} = \{(x, c) \mid x \in L \wedge (x, D_{sk}(c)) \in R\}$ with witness relation $R'_{\xi_1} = \{((x, c), (w, r)) \mid (x, w) \in R \wedge E_{pk,r}(w) = c\}$. Let (P'_{ξ_1}, V'_{ξ_1}) be a unbounded simulation sound proof system for L'_{ξ_1}, and let S'_{ξ_1} be its simulator guaranteed to exist by [79].

Definition 5.6.1 (Prover $P(x, w, \xi)$).

```
$pk \leftarrow \xi_1$
$(c, r) \leftarrow E_{pk}(w)$
$\pi' \leftarrow P'_{\xi_1}((x, c), (w, r), \xi_0)$
$\pi \leftarrow (c, \pi')$
return $\pi$
```

Definition 5.6.2 (Verifier $V(x, \pi, \xi)$).

```
Parse $\pi$ as $(c, \pi')$
return $V'_{\xi_1}((x, c), \pi', \xi_0)$
```

Definition 5.6.3 (Simulator $S(\mathsf{tag}, \mathsf{params})$).

```
if $\mathsf{tag} = \mathsf{setup}$ then
    Parse $\mathsf{params}$ as $1^\kappa$
    $(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$
    $\xi_1 \leftarrow pk$
    $(\xi_0, \mathsf{simstate}') \leftarrow S'(\mathsf{setup}, 1^\kappa)$
    $\mathsf{simstate} \leftarrow (sk, \mathsf{simstate}')$
    $\xi \leftarrow (\xi_0, \xi_1)$
    return $(\xi, \mathsf{simstate})$
else if $\mathsf{tag} = \mathsf{simulate}$ then
    Parse $\mathsf{params}$ as $(x, \xi, \mathsf{simstate})$
    Parse $\mathsf{simstate}$ as $(sk, \mathsf{simstate}')$
    $c \leftarrow E_{pk}(0)$
    $\pi' \leftarrow S'(\mathsf{simulate}, (x, c), \xi_0, \mathsf{simstate}')$
    $\pi \leftarrow (c, \pi')$
    return $\pi$
else if $\mathsf{tag} = \mathsf{extract}$ then
    Parse $\mathsf{params}$ as $(\pi, x, \xi, \mathsf{simstate})$
    Parse $\mathsf{simstate}$ as $(sk, \mathsf{simstate}')$
    Parse $\pi$ as $(c, \pi')$
    $w \leftarrow D_{sk}(c)$
    return $w$
else
    return $\perp$
end if
```

We prove, in order, the properties adaptive indistinguishability and extractable simulation soundness of our construction.

Adaptive Indistinguishability Assume (P,V) is not adaptively indistinguishable. Let A be an adversary such that $\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa)$ is non-negligible. We will use A to construct $A_{\mathsf{ad-ind}}$, breaking either the adaptive indistinguishability of (P',V'), or $A_{\mathsf{sem-sec}}$, breaking the semantic security of $\mathcal{CS}$.

Let $(\xi_0,\mathsf{simstate}') \leftarrow S'(\mathsf{setup},1^\kappa)$, and let ξ_1 be chosen at random. Let pk be the public key defined by ξ_1. Consider the proof system $(\tilde{P},\tilde{V})$, which is identical to (P,V) except that instead of outputting (c,π'), the prover $\tilde{P}$ outputs $(c,S'(\mathsf{simulate},(x,c),\xi_0,\mathsf{simstate}'))$. Assume A wins the Experiments 2.4.11, 2.4.12 with non-negligible probability when Experiment 2.4.11 is run with P and S is replaced by $\tilde{P}$ in Experiment 2.4.12. Then $A_{\mathsf{ad-ind}}$ running in Experiment 2.4.11 can use A as follows. When A asks for a proof, simulated or honest, of $(x,w) \in R$, then $A_{\mathsf{ad-ind}}$ computes $(c,r) \leftarrow E_{pk}(w)$ and asks its experiment for a proof of $(x,c),(w,r)$. When the answer π' is received, it prepends c and returns the answer to A. If $A_{\mathsf{ad-ind}}$ is run with P', then this is what P would return, and if run with S', then the answer is that of $\tilde{P}$. When A returns its guess, the same guess is forwarded by $A_{\mathsf{ad-ind}}$. By construction $A_{\mathsf{ad-ind}}$ is successful when A is.

If A does not distinguish between P and $\tilde{P}$, then it distinguishes between $\tilde{P}$ and S with non-negligible probability. Let us define a chain of machines $\tilde{P}_0,\ldots,\tilde{P}_k$ such that $\tilde{P}_t$ answers the t first queries as S and the remaining queries as $\tilde{P}$, i.e., $\tilde{P}_0 = S$ and $\tilde{P}_k = \tilde{P}$ for k such that A makes at most k queries. Then A can distinguish between $\tilde{P}_t$ and $\tilde{P}_{t+1}$ for some t with non-negligible probability. Fix such a t. We show how such a machine A can be used by $A_{\mathsf{sem-sec}}$ in the following way. $A_{\mathsf{sem-sec}}$ receives a public key pk as input in Experiment 2.4.8, and lets ξ_1 be the CRS corresponding to pk and defines $(\xi_0,\mathsf{simstate}') \leftarrow S'(\mathsf{setup},1^\kappa)$. When A make query $t+1$ on (x,w), then $A_{\mathsf{sem-sec}}$ requests an encryption c of either w or 0 from its experiment, and returns $(c,S'(\mathsf{simulate},(x,c),\xi_0,\mathsf{simstate}'))$ to A. If A responds that it is executed with $\tilde{P}_t$, then $A_{\mathsf{sem-sec}}$ guesses that 0 was encrypted, and otherwise that w was encrypted. By construction $A_{\mathsf{sem-sec}}$ is successful when A is.

Since, by assumption, (P',V') has adaptive indistinguishability and $\mathcal{CS}$ is semantically secure, the existence of either $A_{\mathsf{ad-ind}}$ or $A_{\mathsf{sem-sec}}$ with the above properties is a contradiction. Hence (P,V) has adaptive indistinguishability.

Extractable Simulation Soundness Assume (P,V) does not have extractable simulation soundness, and let A be an adversary which wins in Experiment 2.4.14 with non-negligible probability. We describe how to construct an adversary $A_{\mathsf{sim-sound}}$ which breaks the simulation soundness of (P',V').

$A_{\mathsf{sim-sound}}$ runs A in Experiment 2.4.14, while taking part in Experiment 2.4.13 itself. When $A_{\mathsf{sim-sound}}$ receives the CRS ξ it uses it as ξ_0 in Experiment 2.4.14, while ξ_1 is generated as in the definition of S. $A_{\mathsf{sim-sound}}$ answers queries to S by the algorithm in Definition 5.6.3, using its oracle S' where necessary. When A outputs (x,π) on the call $A(\mathsf{guess},\xi)$, $A_{\mathsf{sim-sound}}$ parses π as (c,π') and outputs $((x,c),\pi')$ on its call $A_{\mathsf{sim-sound}}(\mathsf{guess},\xi)$. If $w \leftarrow D_{sk}(c)$ is not a witness of x, then $(x,c) \notin L'$. Thus $A_{\mathsf{sim-sound}}$ wins in its experiment exactly when A wins. Thus $A_{\mathsf{sim-sound}}$ breaks the simulation soundness of (P',V'), which is a contradiction.

We conclude that (P, V) is extractable simulation sound. $\square$

5.7 Future Work

It remains an open problem to construct a practical scheme which is secure in our sense under some well-established number-theoretical assumptions such as the strong RSA assumption and the Decision Diffie-Hellman assumption.

Chapter 6

Practical Universally Composable Electronic Cash

6.1 Introduction

For the schemes in Section 4.2 most of the execution time of an actual implementation is used for exponentiations or other tasks involving trapdoor functions. The scheme described in this chapter aims at reducing the number of such time-consuming operations.

Our scheme does not offer the same anonymity towards the bank as many other schemes. In particular it does not satisfy the definition of the previous chapter. Its high efficiency is due to the fact that on the user side, only symmetric primitives, such as evaluation of pseudo-random functions and computation of hash functions, are performed. It is commonly believed that there exist efficient algorithms for the primitives needed, e.g., AES and SHA-256.It is an interesting question whether a scheme that does not involve trapdoor functions can offer the anonymity towards the bank in the same strong sense as, e.g., [34]. For a more thorough discussion on this, see Section 6.7.

Relations to Group Signatures

Although proposed as a scheme for electronic cash, our scheme has some similarities with group signatures. The bank has the ability to open a coin to extract the identity in the same way the group manager can open a signature. As a matter of fact, our scheme can be seen as a group signature scheme with one-time keys. This is discussed in further detail in Section 6.6.

Comparison with Sander-Ta-Shma

As in the scheme by Sander and Ta-Shma [78], in our scheme the bank builds a hash tree and the merchant uses the published root when verifying a coin. In their scheme a zero-knowledge protocol is used by the user to prove ownership of a preimage of a hash value and a path to a certified root. Focusing on efficiency, we avoid the zero-knowledge proof by letting the user reveal preimages of $\kappa/2$ out of κ hash values. The cost for this efficiency increase is that the bank always can identify the payer.

6.2 Notation and Definitions

Notation

String concatenation is denoted by $||$. For two integers a and b their concatenation $a||b$ is the number created by concatenating their binary representations, e.g., $a||b = 2^{k_b} + b$, if b is a k_b-bit number.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_k\}$, $i_j < i_{j+1}$, be a subset of $[1, n]$. For a list of values $v = (v_1, v_2, \ldots, v_n)$ we define $v_{\mathcal{I}} = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$. Let f be a function, $S = \{s_1, s_2, \ldots, s_m\}$ a set, and $v = (v_1, v_2, \ldots, v_n)$ a vector. We define $f(S) = \{f(s_1), f(s_2), \ldots, f(s_m)\}$ and $f(v) = (f(v_1), f(v_2), \ldots, f(v_n))$.

Merkle Trees and Hash Chains

Consider the task of proving that a value belongs to a set of certified values. One way to achieve this is to create a binary tree with the values as leaves by setting the value of every inner node to the hash value of the concatenation of the values of its two children and publish the root in a certified way. This tree is called a *Merkle tree* [64].

From a Merkle tree a *hash chain* from each leaf up to the root of the tree can be constructed. For each step the chain contains a value and an order bit which says whether the given value should be concatenated from the left or from the right.

An example of a Merkle tree is given in Figure 6.1. From the tree in the figure we can construct a hash chain from c_{121} up to the root as $(c_{121}, \omega, v_2, r, v_{11}, l, c_{122}, r)$. The values in the chain from c_{121} have been circled in the figure. Note that v_1 and v_{12} are not part of the chain, since these values are computed during verification.

Definition 6.2.1 (Hash chain). *A hash chain h of length d is a vector $h = (v, h_0, h_1, o_1, h_2, o_2, \ldots, h_{d-1}, o_{d-1})$ where $o_i \in \{l, r\}$. A hash chain is said to be valid under a hash function H if $h_0 = h'_0$, where $h'_{d-1} = v$ and*

$$h'_{i-1} = \begin{cases} H(h_i||h'_i) & \text{if } o_i = l \\ H(h'_i||h_i) & \text{if } o_i = r \end{cases}$$

for $i = d-1, d-2, \ldots, 1$. This is written $\mathsf{isvalid}_H(h) = 1$, or $\mathsf{isvalid}(h) = 1$ if it is clear from the context which hash function is used. We also define $\mathsf{root}(h) = h_0$ and $\mathsf{leaf}(h) = v$.

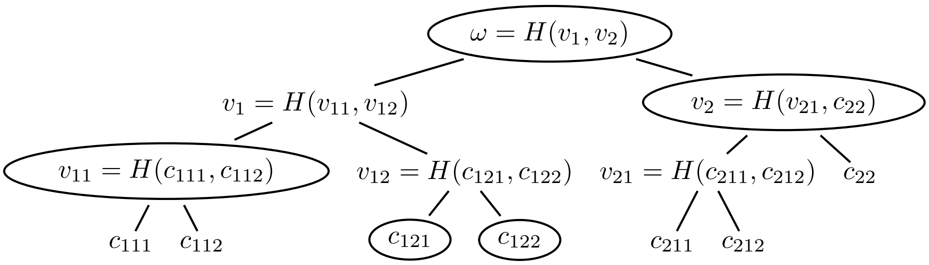

$\omega = H(v_1, v_2)$
$v_1 = H(v_{11}, v_{12})$
$v_2 = H(v_{21}, c_{22})$
$v_{11} = H(c_{111}, c_{112})$
$v_{12} = H(c_{121}, c_{122})$
$v_{21} = H(c_{211}, c_{212})$
c_{22}
c_{111}
c_{112}
c_{121}
c_{122}
c_{211}
c_{212}

Figure 6.1: A Merkle tree with the values stored in the hash chain from c_{121} to the root marked.

Once a Merkle tree has been built for a set of values and its root value has been published, constructing a hash chain for a value not in the set implies finding a collision for the hash function. Since this is assumed to be infeasible, Merkle trees give a method of proving membership.

We define the randomized function $\mathsf{buildtree}_H(S)$ with input a set $S = \{s_1, s_2, \ldots, s_n\}$ to build and output a hash tree of depth $\lceil \log_2 n \rceil$ where the leaves have values $s_1, \ldots, s_n$ in random order. When n is a power of two, all leaves have equal depth $d-1$, and otherwise some leaves have depth $d-2$. The function $\mathsf{getchain}_T(s)$ returns the hash chain from the first leaf with value s to the root in the tree T and $\emptyset$ if no such leaf exists.

6.3 The Protocol

Security Parameters

Two security parameters, κ_1 and κ_2, are used in the protocol. The parameter κ_1 can be thought of as key length for the symmetric cipher, and κ_2 is the number of bits needed so that each merchant can be identified by a κ_2-bit number with $\kappa_2/2$ number of ones.

The Players

The players in the protocols are denoted $\mathcal{B}, P_1, \ldots, P_m$. To simplify the description we also write P_0 for $\mathcal{B}$. Except for the bank, any player may act as a customer, i.e., withdraw and spend coins, as well as a merchants, i.e., accept payments and deposit coins. We abuse notation and let P_i represent both the identity of the player and the Turing machine taking part in the protocol.

We let $\mathcal{I}$ be a public map from identities to $[\kappa_2]$ such that $\mathcal{I}(P_i)$ has cardinality $\kappa_2/2$ and $\mathcal{I}(P_i) \neq \mathcal{I}(P_j)$ for $P_i \neq P_j$. $\mathcal{I}$ can be thought of as a collision-free hash function which maps its input to $\{0,1\}^{\kappa_2}$ with the additional property that the number of 1's in the output is always exactly $\kappa_2/2$. We define

$$\mathsf{span}_{\mathcal{I}}\left(\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}\right) = \left\{ P \mid \mathcal{I}(P) \subseteq \bigcup_{j=1}^{k} \mathcal{I}(P_{i_j}) \right\} .$$

Given preimages of a coin corresponding to players $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ one can combine the preimages to spend the coin at any player in $\mathsf{span}_{\mathcal{I}}(\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\})$. It holds that $P \in \mathsf{span}_{\mathcal{I}}(S)$ if $P \in S$ and since $\mathcal{I}$ is injective $\mathsf{span}_{\mathcal{I}}(\{P\}) = \{P\}$.

6.4 The Ideal Functionality

Introduction

In this section we define the ideal functionality and discuss why it captures the properties of an e-cash scheme.

We use a model for universal composability which is described in Section 2.2. The functionality described here has only one non-immediate function – the withdrawal protocol.

The adversary is allowed to choose an arbitrary number of players to corrupt at start-up. For further discussion on this, see Section 6.7. We do not allow the adversary to corrupt $\mathcal{B}$. It would be possible to give a functionality that allows the adversary to corrupt $\mathcal{B}$. In such a functionality even a corrupted $\mathcal{B}$ would not be able to "revoke" an issued coin. However, since this would the functionality more complex, we describe $\mathcal{F}_{\text{EC}}$ for a trusted bank.

Informal Description

The ideal functionality $\mathcal{F}_{\text{EC}}$ for an e-cash scheme accepts the following messages.

- `KeyGen` to set up keys.
- `Issue Coin` to issue a coin to the designated user.
- `Tick` to build a new hash tree.
- `Prepare Coin` to mark a coin for spending at a certain merchant.
- `Verify Coin` to verify whether or not a coin can be spent at a certain merchant.
- `Open Coin` to let the bank extract the identity of the user the coin was issued to.
- `Check Doublespent` to check whether a coin has been spent more than once.

There is no separate message for depositing a coin at the bank. To deposit the merchant hands the coin to the bank, who runs the `Verify Coin` algorithm to check that the coin is valid.

Table 6.1: The tables stored by the ideal functionality $\mathcal{F}_{\mathrm{EC}}$.

Name	Content
C_i	(c, P_i, k, h), where c is a bit-string, P_i the coin-owner, k the coin-secret and h the hash chain.
T_{prepared}	Coins which are about to be spent.
T_{signed}	The certified roots.

Definition of the Ideal Functionality

The ideal functionality $\mathcal{F}_{\mathrm{EC}}$ holds a counter t that is initialized to 0 and indexed sets C_i for coins that have been issued in period i. For convenience we let $C = \cup_i C_i$. For $e = (c, \cdot, k, \cdot) \in C$ we define $\mathsf{val}_H(e) = c||H(k_1)||H(k_2)|| \cdots ||H(k_{\kappa_2})$. The functionality holds a set of signed roots T_{signed} and a set of coins ready to be spent T_{prepared}. The sets $C_i, T_{\text{signed}}, T_{\text{prepared}}$ are initialized to $\emptyset$. The tables stored by the functionality are summarized in Table 6.1.

The functionality must be indistinguishable from the protocol, which implies that it must output data on the same format as the real protocol, and therefore in some way depend on the implementation of the real protocol. This can be achieved by querying the ideal adversary for any such output, or the functionality can produce the output itself. In the former case, the ideal adversary needs to be tailor-made for a certain implementation of the protocol, whereas in the latter case the functionality must be parameterized on the implementation. We choose the second approach in this paper.

Functionality 6.4.1 ($\mathcal{F}_{\mathrm{EC}}^{H,\mathcal{R},\mathcal{CS},\mathcal{SS}}$)**.** Until $(\mathcal{B}, \texttt{KeyGen})$ is received all messages except $(\mathcal{B}, \texttt{KeyGen})$ are ignored.

- Upon reception of $(P_i, \texttt{KeyGen})$ proceed as follows:
    1. If $P_i = \mathcal{B}$, set $sk \leftarrow \mathsf{Kg}(\kappa_1)$, $(pk, sk) \leftarrow \mathsf{SSKg}(\kappa_1)$, and return $(\mathcal{B}, \texttt{KeyGen}, pk)$.
    2. Else record P_i in the member list and draw U^i from the family $\mathcal{U}_{\kappa_1}$ and return $(P_i, \texttt{KeyGen})$.
- Upon reception of $(\mathcal{B}, \texttt{Issue Coin}, P_i)$, verify that P_i is in the member list. If not, return $(\mathcal{B}, \texttt{Not A Member})$ and quit. Set

  $$c \leftarrow E_{sk}(0),\ k_j \leftarrow (U^i(c||j))_{j=1}^{\kappa_2},\ z \leftarrow H(k)\ ,$$

  where $k = (k_1, k_2, \ldots, k_{\kappa_2})$. Add $(c, P_j, k, \emptyset)$ to C_t. Hand $(\mathcal{S}, \texttt{New Coin}, P_i)$ and $(P_i, \texttt{New Coin}, c, z)$ to $\mathcal{C_I}$.
- Upon reception of s message $(\mathcal{B}, \texttt{Tick})$, set $T \leftarrow \mathsf{buildtree}_H(\mathsf{val}_H(C_t))$ and modify each $e = (c, P_i, k, \emptyset) \in C_t$ into $(c, P_i, k, \mathsf{getchain}_T(\mathsf{val}_H(e)))$. Compute

$\sigma \leftarrow \mathsf{Sig}_{sk}(\mathsf{root}(T))$ and add $\mathsf{root}(T)$ to T_{signed}. Return $(\mathcal{B}, \texttt{Tick}, T, \sigma)$ to $\mathcal{C}_{\mathcal{I}}$. Set $t \leftarrow t + 1$.

- Upon reception of $(P_i, \texttt{Prepare Coin}, c, z, P_j)$, find k such that $(c, P_i, k, \cdot) \in C$. If no such k exists, then hand $\mathcal{C}_{\mathcal{I}}$ the message $(P_i, \texttt{Reject Prepare Coin})$ and quit. Otherwise set $\tilde{k} \leftarrow k_{\mathcal{I}(P_j)}$, return $(P_i, \texttt{Prepared Coin}, c, \tilde{k})$ to $\mathcal{C}_{\mathcal{I}}$ and store (c, P_j) in T_{prepared}.

- Upon reception of $(P_i, \texttt{Verify Coin}, c, z, \tilde{k}, P_j, h', \sigma, pk')$, find P_l, k, h such that $(c, P_l, k, h) \in C$. Return $(P_i, \texttt{Verify Coin}, c, P_j, \texttt{invalid})$ to $\mathcal{C}_{\mathcal{I}}$ at least one of the following holds:

  1. No such entry exists.
  2. $pk = pk'$ and $\mathsf{root}(h) \notin T_{\text{signed}}$.
  3. $\mathsf{Vf}_{pk'}(\mathsf{root}(h), \sigma) = 0$.
  4. $h' \neq h$.
  5. P_l is not corrupted, and
  $$(\tilde{k} \neq k_{\mathcal{I}(P_j)}) \vee (P_j \notin \mathsf{span}_{\mathcal{I}}(\{P \mid (c, P) \in T_{\text{prepared}}\})) \ .$$
  6. P_l is corrupted and $H(\tilde{k}) \neq z_{\mathcal{I}(P_j)}$.

  Otherwise return $(P_i, \texttt{Verify Coin}, c, P_j, \texttt{valid})$ to $\mathcal{C}_{\mathcal{I}}$.

- Upon reception of $(\mathcal{B}, \texttt{Open Coin}, c)$, find a value $(c, P, \cdot, \cdot)$ in C. If no such entry exists, then set $P \leftarrow D_{sk}(c)$. Return $(\mathcal{B}, \texttt{Open Coin}, c, P)$.

- Upon reception of $(P_l, \texttt{Check Doublespent}, c, z, \tilde{k}_1, \tilde{k}_2, h, \sigma, P_{j_1}, P_{j_2})$ from $\mathcal{C}_{\mathcal{I}}$, execute $(\texttt{Verify Coin}, c, z, \tilde{k}_i, h, \sigma, P_{j_i})$ for $i = 1, 2$.

  1. If at least one execution returns $(\texttt{Verify Coin}, c, P_{j_i}, \texttt{invalid})$, then return $(P_l, \texttt{Check Doublespent}, c, \texttt{invalid})$ to $\mathcal{C}_{\mathcal{I}}$.
  2. If $P_{j_1} = P_{j_2}$ then return $(P_l, \texttt{Check Doublespent}, c, \texttt{no})$, otherwise return $(P_l, \texttt{Check Doublespent}, c, \texttt{yes})$ to $\mathcal{C}_{\mathcal{I}}$.

The functionality is parameterized by a symmetric encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$, a signature scheme $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$, a family of pseudo-random functions $\mathcal{R}$ and a collision-free, one-way hash functions H drawn from a family of hash functions $\mathcal{H}_{\kappa_1}$. To simplify the description we assume that $\mathcal{SS}$ is correct. Instead of parameterizing the functionality it is possible to give a non-parameterized description, where the functionality is given (a description of) the function families from $\mathcal{S}$ at startup. The definition of $\mathcal{F}_{\text{EC}}$ is given in Functionality 6.4.1.

$\mathcal{F}_{\text{EC}}$ captures some specifics of the current scheme, such as the tree update function, the specific format of a coin, the weaker anonymity, a non-interactive payment protocol, and the possibility to transfer prepared coins to other users. Therefore a generic ideal functionality for electronic cash would differ from ours.

On the Ideal Functionality

In this section we discuss why $\mathcal{F}_{\mathrm{EC}}$ captures the security requirements for electronic cash. The five messages `KeyGen`, `Issue Coin`, `Tick`, `Prepare Coin`, and `Check Doublespent` are all straight-forward. They manipulate tables, and use $H, \mathcal{R}, \mathcal{CS}, \mathcal{SS}$ only to produce output that has the format of a coin. When answering the `Open Coin` query, the functionality decrypts c if it is not found in the table. This is so since the CCA2-security of $\mathcal{CS}$ does not prevent $\mathcal{Z}$ from producing a valid plaintext-ciphertext pair and use it to whether it interacts with the functionality or the real protocol.

Since the most involved message is `Verify Coin`, we discuss it in more detail. As noted in [31], a messages created by corrupted players or messages created by keys that do not originate from the protocol must be verified according to the real protocol rather than rejected. Otherwise the environment $\mathcal{Z}$ could distinguish between the ideal functionality and the real protocol by creating a new pair of signature keys and sign a root with this new key pair. The same holds for a corrupted $\mathcal{U}$ which might leak its secret to $\mathcal{Z}$ to let $\mathcal{Z}$ prepare coins internally without interacting with the protocol.

When a coin is verified, Condition 1 says that it should be considered invalid if it has not been issued by $\mathcal{B}$. Condition 2 say that if the coin is being verified with the correct key public key, then it is valid only if $\mathcal{B}$ actually signed the root, and Condition 3 ensures a correct answer when the coin is verified with a different public key. Because of the correctness of $\mathcal{SS}$, Condition 3 always holds for $pk = pk'$ if the coin has been signed. Condition 4 says that the correct path must be given. Condition 5 says that if the coin owner is not corrupted, the coin must have been prepared for the designated receiver P_j. (Recall that $\mathsf{span}_I(\{P\}) = \{P\}$). Alternatively if the coin has been prepared more than once, then P_j must be in the span of the set of receivers. Condition 6 says that for a corrupt coin owner, the coin is accepted if the given preimages actually hash to the correct values.

Anonymity

In the ideal protocol, c is an encryption of 0, and thus the coin does not contain any information about the owner. The only information that is disclosed to the merchant is to which tree the coin belongs. The amount of information this contains depends on the size of the tree. The larger the tree, i.e., the longer the interval between `Tick` messages, the smaller the amount of information released to the merchant.

Fairness

By fairness we mean that if a player (or coalition of players) prepares $l + 1$ coins that pass `Verify Coin` after withdrawing only l coins, at least one withdrawn coin will be detected as double-spent. Since the only coins that can be successfully spent are the coins in the database C, and the only way to have a coin being added to the database is to engage in the withdrawal protocol, by the pigeon hole principle at least one coin has been prepared twice in this case. The implementation of the double-spending detection in the ideal functionality guarantees that double-spender is revealed.

Non-Frameability

A coalition of players should not be able to spend coins withdrawn by someone outside of the coalition. Since the `Prepare Coin` algorithm checks that it is called by the coin owner, this requirement is fulfilled.

Detection of double-spenders

A user that spends a coin at two different merchants will by construction have to produce two different sets of k_i's and will always be detected by the bank.

Since double-spending at a single merchant will not be detected by the bank, it is the responsibility of the merchant to detect such actions, holding a list of the coins spent at that merchant. However, a simple modification of the scheme allows the merchant to remember only the coins spent the same day. To achieve this, we include the date in the computation of the index set. In the other words, rather than disclosing the list $k_{\mathcal{I}(P_i)}$, the list $k_{\mathcal{I}(P_i, date)}$ is disclosed.

Correctness

An e-cash scheme is correct if a coin withdrawn by an honest player always, or almost always, can be spent at an honest merchant and the merchant can deposit the coin at the bank. It is immediate from the construction that this property holds for the ideal functionality provided that the signature scheme $\mathcal{SS}$ is correct.

6.5 The Real Protocol

Definition of the Protocol

We give the definition of the protocol in the $\mathcal{F}_{\text{SIG}}$-hybrid model. The ideal signature functionality $\mathcal{F}_{\text{SIG}}$ [5, 31] accepts messages `KeyGen`, `Sign`, `Verify` to set up keys, sign a message, and verify a signature. We use the definition of $\mathcal{F}_{\text{SIG}}^{\mathcal{SS}}$ given in Figure 6.2, slightly modified from [31] in that the functionality is parameterized by the signature scheme, and $\mathcal{SS}$ is assumed to be correct.

We are now ready to define the protocol $\pi_{\text{EC}}^{H,\mathcal{R},\mathcal{CS}}$.

Protocol 6.5.1 ($\pi_{\text{EC}}^{H,\mathcal{R},\mathcal{CS}}$).

- The bank $\mathcal{B}$ acts as follows:
  - Upon reception of (`KeyGen`), $\mathcal{B}$ creates and stores a symmetric key $sk \leftarrow \mathsf{Kg}(\kappa_1)$, requests pk from $\mathcal{F}_{\text{SIG}}$, sets $C \leftarrow \emptyset$ and returns (`KeyGen`, pk).

Functionality 6.5.1 ($\mathcal{F}_{\text{SIG}}^{\mathcal{SS}}$ [31]).

- Upon reception of $(\mathcal{B}, \texttt{KeyGen})$, set $(pk, sk) \leftarrow \mathsf{SSKg}$. Hand $(\mathcal{B}, pk)$ to $\mathcal{C}_{\mathcal{I}}$.
- Upon reception of $(\mathcal{B}, \texttt{Sign}, m)$, set $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$, store m, and hand $(\mathcal{B}, \texttt{Signature}, m, \sigma)$ to $\mathcal{C}_{\mathcal{I}}$.
- Upon reception of $(P_i, \texttt{Verify}, m, \sigma, pk')$, set $f = 0$ if $\mathcal{B}$ is uncorrupted and m is not stored. Otherwise set $f = \mathsf{Vf}_{sk}(m, \sigma)$. Hand $(P_i, \texttt{Verify}, m, f)$ to $\mathcal{C}_{\mathcal{I}}$.

Figure 6.2: The definition of $\mathcal{F}_{\text{SIG}}^{\mathcal{SS}}$.

  - Upon reception of $(\texttt{Issue Coin}, P_i)$, $\mathcal{B}$ initiates the following protocol with P_i:
    1. $\mathcal{B}$ computes $c \leftarrow E_{sk}(P_i)$ and sends $(\texttt{Withdrawal Request}, c)$ to P_i.
    2. P_i sets $k_j \leftarrow R^i(c||j), z \leftarrow H(k)$. Then it outputs $(\texttt{New Coin}, c, z)$ and hands $(\texttt{Withdrawal Response}, c, z)$ to $\mathcal{B}$.
    3. $\mathcal{B}$ stores $(c||z_1||z_2||\dots||z_{\kappa_2})$ in C.
  - Upon reception of $(\texttt{Open Coin}, c)$, $\mathcal{B}$ returns $(\texttt{Open Coin}, c, D_{sk}(c))$.
  - Upon reception of $(\texttt{Tick})$, $\mathcal{B}$ computes a new hash tree T from all stored values, i.e., sets $T = \mathsf{buildtree}_H(C)$. It acquires a signature σ on $\mathsf{root}(T)$ from $\mathcal{F}_{\text{SIG}}$, sets $C = \emptyset$, and returns $(\texttt{Tick}, T, \sigma)$.

- A non-bank player P_i, i.e., $i > 0$, acts as below:
  - Upon reception of $(\texttt{KeyGen})$, P_i creates and stores a pseudo-random function $R^i \leftarrow_R \mathcal{R}_{\kappa_1}$ and returns $(\texttt{KeyGen})$.
  - Upon reception of $(\texttt{Prepare Coin}, c, z, P_j)$, P_i sets $k_l \leftarrow R^i(c||l)$ for $l = 1, \dots, \kappa_2$ and verifies that $z = H(k)$. If this does not hold, it outputs the message $(\texttt{Reject Prepare Coin})$ and quits. Otherwise it sets $\tilde{k} = k_{\mathcal{I}(P_j)}$ and outputs $(\texttt{Prepared Coin}, c, \tilde{k})$.
  - Upon reception of $(\texttt{Withdrawal Request})$, P_i acts as described above.

- In addition to the above, any player P_i, including the bank, acts as follows.
  - Upon reception of $(\texttt{Verify Coin}, c, z, \tilde{k}, P_j, h, \sigma, pk)$, P_i proceeds as follows:
    1. P_i sends $(\texttt{Verify}, \mathsf{root}(h), \sigma, pk)$ to $\mathcal{F}_{\text{SIG}}$. If $\mathcal{F}_{\text{SIG}}$ returns 0, then P_i returns $(\texttt{Verify Coin}, c, P_j, \texttt{invalid})$ and quits.

2. P_i verifies that $H(c,z) = \mathsf{leaf}(h)$ and that $\mathsf{isvalid}_H(h) = 1$. If this is not the case, then P_i returns $(\texttt{Verify Coin}, c, P_j, \texttt{invalid})$ and quits.
3. P_i verifies that $H(\tilde{k}) = z_{\mathcal{I}(P_j)}$. If this is not the case, then it returns $(\texttt{Verify Coin}, c, P_j, \texttt{invalid})$ and quits.

P_i returns $(\texttt{Verify Coin}, c, P_j, \texttt{valid})$.

– Upon reception of $(\texttt{Check Doublespent}, c, z, \tilde{k}_1, \tilde{k}_2, h, \sigma, P_{j_1}, P_{j_2})$, P_i executes $(\texttt{Verify Coin}, c, z, \tilde{k}_i, h, \sigma, P_{j_i})$ for $i = 1, 2$.

1. If at least one of the two executions above returns $(\texttt{Verify Coin}$, c, P_{j_i}, $\texttt{invalid})$, then P_i returns $(\texttt{Check Doublespent}, c, \texttt{invalid})$ and quits.
2. If $P_{j_1} = P_{j_2}$ then P_i returns $(\texttt{Check Doublespent}, c, \texttt{no})$, otherwise it returns $(\texttt{Check Doublespent}, c, \texttt{yes})$.

On the Real Protocol

The protocol relies on the existence of an ideal signature functionality. Such a functionality can be implemented with a CMA-secure signature scheme [5, 31]. It is possible that a merchant will verify several coins from the same tree. In these cases the merchant can save time by only verifying the signature once.

The scheme relies on the roots being constructed after a certain amount of time, and therefore coins may not be immediately usable. The scheme can also be used without this delay by constructing a tree of size one for each coin issued and returning the signature to the user immediately. This increases coin size since there is a separate signature for each coin, but does not increase the amount of computation the user has to perform. This modification also eliminates linkability issues when coins with same owner are placed in the same tree.

Security of the Real Protocol

Theorem 6.5.1. *The protocol $\pi_{\text{EC}}^{H,\mathcal{R},\mathcal{CS}}$ securely realizes $\mathcal{F}_{\text{EC}}^{H,\mathcal{R},\mathcal{CS},\mathcal{SS}}$ in the $\mathcal{F}_{\text{SIG}}$-hybrid model if H is drawn from a family $\mathcal{H}$ of one-way collision-free hash functions, $\mathcal{R}$ is a family of pseudo-random functions, and $\mathcal{CS}$ is a CCA2-secure encryption scheme.*

Proof. We divide the proof into subsections. First we define the simulator, then we define the hybrids used and finally we describe how to break one of the assumptions if an environment can distinguish between the ideal functionality and the protocol.

Description of the Simulator. The simulator works as follows: For each player P_i that the real-world adversary A corrupts, the ideal adversary $\mathcal{S}$ corrupts the corresponding dummy player $\tilde{P}_i$. When a corrupted dummy player $\tilde{P}_i$ receives a message m from $\mathcal{Z}$, the simulator $\mathcal{S}$ lets $\mathcal{Z}'$ send m to P_i. When a corrupted P_i

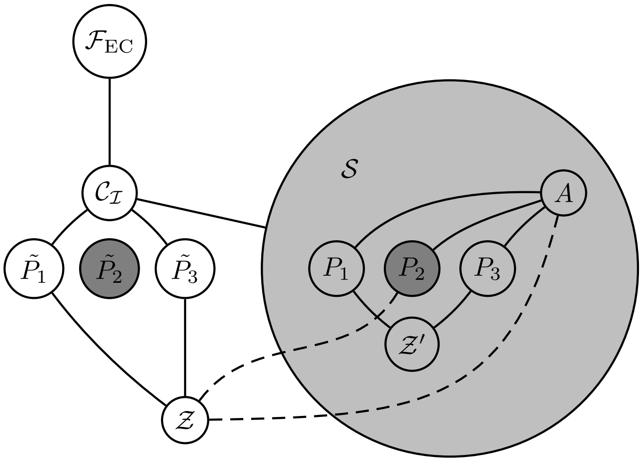


Figure 6.3: The simulator for a protocol with three players where P_2 is corrupted. The dashed edges represent simulated connections.

outputs a message m to $\mathcal{Z}'$, then $\mathcal{S}$ instructs the corrupted $\tilde{P}_i$ to output m to $\mathcal{Z}$. This corresponds to P_i being linked directly to $\mathcal{Z}$.

The simulated real-world adversary A is connected to $\mathcal{Z}$, i.e., when $\mathcal{Z}$ sends m to $\mathcal{S}$, $\mathcal{Z}'$ hands m to A, and when A outputs m to $\mathcal{Z}'$, $\mathcal{S}$ hands m to $\mathcal{Z}$. All non-corrupted players are simulated honestly. The corrupted players run according to their respective protocols.

If $\mathcal{S}$ receives the message $(\texttt{New Coin}, P_i)$ from $\mathcal{F}_{\mathrm{EC}}$, then it instructs $\mathcal{Z}'$ to send $(\texttt{Issue Coin}, P_i)$ to $\mathcal{B}$. All other functions are local and need not be simulated for A.

Building the Hybrids. Now assume there exists an environment $\mathcal{Z}$ that can distinguish between an execution of the ideal protocol and an execution of the real protocol for any ideal adversary $\mathcal{S}$. Then it can distinguish between the two for the simulator described above. We will create a chain of protocols $\pi_0, \dots, \pi_t$ such that π_0 is the ideal protocol and π_t the real protocol. We construct such a chain of polynomially many intermediate steps. If $\mathcal{Z}$ can distinguish between the ideal protocol and the real protocol, then there must exist an i such that $\mathcal{Z}$ can distinguish between π_i and π_{i+1}. We now build the chain and describe how $\mathcal{Z}$ can be turned into a machine that solves one of the problems assumed to be hard. To simplify the description we build the hybrid chains as several subchains called π^0, π^1 etc. We assume all chains have the same length m. Should a chain π^r as described below have length $m' < m$ we can always "pad" by letting $\pi^r_i = \pi^r_{m'}$ for

$i = m' + 1, \ldots, m$.

We let π_0^0 be the ideal protocol. We then let π_i^0 be the same protocol with the difference that up to the ith coin issued we set c to be an encryption of the identity, $c = E_{sk}(P_i)$, rather than $c = E_{sk}(0)$.

We define π_0^1 to be π_m^0. Define π_i^1 to be π_0^1 with the modification that for the first i calls to `Open Coin` actually decrypts c according to the real protocol instead of looking up the answer in the table.

Let $\pi_0^2 = \pi_m^1$. Define π_i^2 to be π_0^2 with the difference for player $P_1, \ldots, P_i$ a pseudo-random function is used to generate k_i instead of the random function U^i.

Let $\pi_0^3 = \pi_m^2$. Define π_i^3 to be π_0^3 with the modification that up to the ith time `Check Doublespent` is called `yes` is returned if more than $\kappa_2/2$ hash values have been opened rather than using the table.

The protocol π^4 is defined to be π^3 with the difference that the in the first i calls to `Verify Coin` it is checked whether the path p is valid rather than checks that the coin exists in the table.

Finally the hybrid π^5 is defined to be π^4 with the modification that for π_i^5 the i first times the `Verify Coin` algorithm is called it is checked that $\kappa_2/2$ values k_i hash to z_i rather then checking them against the tables C and S.

Breaking the Assumption. Assume $\mathcal{Z}$ can distinguish between π_i^0 and π_{i+1}^0 for some i. We show how to use $\mathcal{Z}$ to build an algorithm A that breaks the CCA2-security of $\mathcal{CS}$. Participating in Experiment 2.4.10 A has access to an encryption and a decryption oracle. No symmetric key is generated, but instead the encryption oracle is used to correctly form the first i coins, and the decryption oracle is used to open coins. When the $(i + 1)$st coin is about to be created with identity P_j, A asks the challenge oracle to encrypt either 0 or P_j. All subsequent coins are created according to the ideal functionality.

Note that if the challenge oracle encrypts 0, the protocol executed is π_i^0, and if it encrypts P_j, the protocol is π_{i+1}^0. Since $\mathcal{Z}$ is able to distinguish between π_i^0 and π_{i+1}^0 with non-negligible probability it will break the CCA2-security of $\mathcal{CS}$.

The protocols in hybrid chain π^1 answer the `Open Coin` query identically, and thus $\mathcal{Z}$ cannot distinguish between them.

Assume $\mathcal{Z}$ can distinguish between π_i^2 and π_{i+1}^2 for some i. We show how to use $\mathcal{Z}$ to build an algorithm A that is able to distinguish between a pseudo-random function R and a random function U, thus contradicting the assumption that R is pseudo-random. A is given oracle access to a function f. For players $P_1, \ldots, P_i$ a pseudo-random function is used as in the real protocol. When coins are created for player P_{i+1}, the oracle for f is used to generate k_j, and for players P_l, $l > i + 1$, the random function is used as in the ideal protocol.

If f is drawn from $\mathcal{U}_{\kappa_1}$, then the protocol described is π_i^2, and if f is drawn from $\mathcal{R}_{\kappa_1}$, then the protocol is π_{i+1}^2. Thus if $\mathcal{Z}$ is able to distinguish between π_i^2 and π_{i+1}^2 with non-negligible probability p, then A can distinguish between pseudo-random functions and random functions with the same probability p.

By construction instances of π^3 are indistinguishable, since no two $P_i \neq P_j$ have the same associated index set.

Assume $\mathcal{Z}$ distinguishes between π_i^4 and π_{i+1}^4. This means that $\mathcal{Z}$ with non-negligible probability has created a coin (c, z) and corresponding path h such that

- $\mathsf{root}(h)$ was signed by $\mathcal{B}$.
- h was not in the original tree created by $\mathcal{B}$.

From this we can construct an algorithm A that given a key for the hash function finds a collision, contradicting the assumption that $\mathcal{H}$ is collision-free.

Assume $\mathcal{Z}$ distinguishes between π_i^5 and π_{i+1}^5 with non-negligible probability p_1. In that case, $\mathcal{Z}$ has succeeded to provide `Verify Coin` with c, z and values k_i such that either

- no entry c, P_j exists in T_{prepared}, or
- the values k_i do not match the values in the database.

and $H(k_i) = z_i$.

In the first case, the corresponding `Prepare Coin` has not been executed, and thus the values of k_i have not been revealed. We can now produce an algorithm A that given y computes $x \in H^{-1}(y)$, contradicting the assumption that H is hard to invert. Let us assume that coin verified in the $(i+1)$st call to `Verify Coin` was issued to P in the jth call to `Issue Coin` with non-negligible probability p_2. A runs the protocol honestly except that $z_l = y$, where l is the smallest index in the index set $\mathcal{I}(P)$. Since $\mathcal{Z}$ is able to make `Verify Coin` accept, it must have provided x such that $H(x) = y$. A then outputs x. Under the assumptions A succeeds with probability at least $p_1 p_2$.

In the second case, we can construct an algorithm that finds a collision in the hash function.

We have now shown that if $\mathcal{Z}$ can distinguish between π_{EC} and $\mathcal{F}_{\text{EC}}$, it can be used either to break the CCA2-security of $\mathcal{CS}$, to break the CMA-security of $\mathcal{SS}$, to distinguish between $\mathcal{R}$ and random functions, to find a collision in H, or to compute $H^{-1}(x)$ for a random x. Since this is assumed to be hard, the proof is concluded. $\square$

6.6 Comparison to Group Signatures

Our scheme is in some ways similar to group signatures. A coin can be viewed as a signature on the identity of the merchant. Signatures by different users are indistinguishable by the merchant, but not by the bank. This corresponds to a group signatures scheme where the bank acts as group manager.

A user can only sign once for every coin she withdraws. For electronic cash this is a fundamental property, but it differs, of course, from ordinary group signatures. Also when used a group signatures scheme, there is no exculpability against the group manager. In other words the group manager can frame a group member.

Some group signature schemes offer revocation. When converting our scheme into a group-signature like scheme this can be achieved by publishing the coins issued to the revoked player. When verifying a signature, the verifier first checks the coin against the revocation list.

6.7 Additional Notes

Can We Do Better?

As seen above, the proposed scheme lacks some of the properties one could ask from an e-cash scheme. Also, when used as a group signature scheme, it does not have all the properties one could wish for. The reason for this is that we base the scheme on symmetric rather than asymmetric primitives. A natural question to ask is whether one could do better using only symmetric primitives.

It is known [24] that the existence of a group signature scheme implies existence of a CCA2-secure public-key encryption scheme. Impagliazzo and Rudich [53] show that it is unlikely that a public-key encryption scheme can be based only on the assumption of the existence of one-way functions where the function is used as a black box. Actually, the existence of such a construction would give a proof that $\mathbf{P} \neq \mathbf{NP}$. Therefore a construction of a group signature scheme from black-box access to symmetric primitives is likely to be extremely involved.

For electronic cash the situation is less clear. If double-spenders are detected only by the bank and not by anyone else (including the merchant), then it is possible to reduce CCA2-encryption to electronic cash in the same way as for group signatures. The same holds if there is a trusted third party that can identify coin owners. However, we are not aware of such a reduction from e-cash in general.

These restrictions only hold when there is only black-box access to symmetric primitives. When one is allowed access to the circuit computing a one-way functions, it is possible to, e.g., prove in zero-knowledge the knowledge of a preimage of a value under a hash function. Although polynomial, such proofs would most likely be highly inefficient.

On Adaptive Security

The security proof assumes that the adversary corrupts players in a non-adaptive way. An adversary that is allowed to corrupt adaptively would be able to distinguish between the real protocol and the ideal functionality. In the ideal functionality, the preimages a user $\mathcal{U}$ uses are random numbers, whereas in the real protocols they are pseudo-random numbers. When corrupting $\mathcal{U}$, the adversary expects to receive a key for the pseudo-random function that matches the preimages. In the ideal protocol the probability that such a key even exists is negligible.

We can modify the real protocol to solve the problem. If $\mathcal{U}$ instead of generating the preimages from a pseudo-random function generates random numbers, the above scenario does not apply. The drawback is that the amount of data that $\mathcal{U}$ needs to store increases.

Coin Size and External Databases

As the reader may have noted, the hash chains do not contain any sensitive information. Therefore they can be stored in public databases rather than by the user. This gives a way to reduce the size of the coins by storing only an index of the hash root together with the path in the tree as a $\{0, 1\}$ string. The merchant can then retrieve the hash values and (c, z) from a public database.

Since the databases do not need to be authenticated as long as the roots are signed, they could be provided by untrusted third parties, and not necessarily by the bank. By verifying the hash chain the merchant would detect if a database is corrupted. A large merchant could even have its own database.

Chapter 7

Conclusion of Part II

We have presented present a security definition for electronic cash which focuses on the user's security. While we have given a construction satisfying the requirements based on general methods, it is still an open problem to construct a scheme based on explicit number-theoretic assumptions.

For our second construction the goal is different. We have described a scheme where the goal is to make the scheme efficiently enough to be usable on a device with limited computing powers such as a smart-card, while keeping a reasonable security level. The scheme has unforgeable coins, and it is anonymous to merchants. It is, however, not anonymous to the bank.

Thus, the two schemes can be seen as two extremes with regards to efficiency and security. Obviously it would be very interesting to increase the efficiency of the former scheme, and to increase the security of the latter.

Part III

Hierarchical Group Signatures

Chapter 8

Introduction, Background, and Definitions

In this part we introduce the notion of hierarchical group signatures, which is a generalization of group signatures. Therefore, before we proceed we give an informal description of group signatures.

Group Signature Schemes

Recall group signatures from Section 1.12 and the generalization of hierarchical group signatures from Section 3.2.

The notion of group signatures was introduced by Chaum and van Heyst [36]. 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.

A group signature scheme is said to be static 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.

Related Work

The concept of group signatures was introduced by Chaum and van Heyst [36] in 1991. The original scheme in [36] and the group signature schemes that followed [37, 23] all have the property that the complexity of the scheme grows with the number of parties. In [29] 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 [4]. 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 [27]. This system was improved in [3]. The currently most efficient scheme that is secure under standard assumptions was given by Camenisch and Groth [24]. More efficient schemes do exist [17, 26], 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. [55], where signatures belonging to a member can be opened, or traced, in a distributed way without revealing the group secret.

Bellare et al. [9] 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 which is secure according to their definitions under general assumptions. Kiayias and Yung [56] define security for dynamic groups and prove that a modification of [3] is secure under these definitions. Independently, Bellare et al. [11] extend the definitions of [9] 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 [9].

In [4] the concepts of multi-group signatures and subgroup signatures are described, and in [58] 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.

As mentioned in Section 3.2, 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 [62] 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 [65]. 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.

Organization of this Part

In this chapter we introduce the notation and give security definitions for hierarchical group signatures. We also discuss the difficulties involved in constructing a hierarchical group signature scheme and propose a weaker security definition, to allow for more efficient realizations.

In Chapter 9 we describe a construction of hierarchical group signatures which is secure if based on a trapdoor permutation family. In Chapter 10 we give a construction that is secure and almost practical under standard computational assumptions. Finally in Chapter 11 we give a construction which is more efficient than the previous two schemes, but which only satisfies the weaker security definition.

Notation and Conventions

The main security parameter is denoted κ, 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. The additional security parameters are defined such that $2^{-\kappa_c}$ and $2^{-\kappa_r}$ are negligible in κ.

8.1 Introduction of the New Notion

In this section we introduce the notion of hierarchical group signatures. 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 encryption schemes that is used in the next section.

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_ω 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 3.1 in the introduction. We reproduce this figure below for convenience.

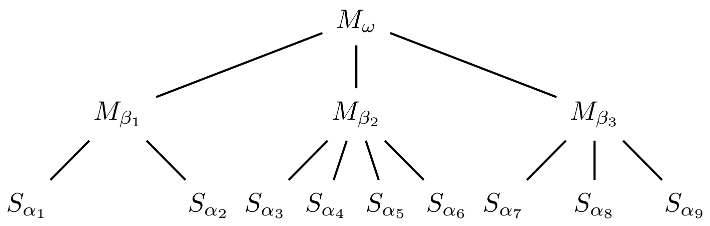


Figure 8.1: A tree of group managers and signers, where $\omega = \{\beta_1, \beta_2, \beta_3\}$, $\beta_1 = \{\alpha_1, \alpha_2\}$, $\beta_2 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6\}$, and $\beta_3 = \{\alpha_7, \alpha_8, \alpha_9\}$.

Note that standard group signatures correspond to having a single group manager $M_{[1,l]}$ that manages all signers $S_1, \ldots, S_l$.

8.2 The Definition of Security

Bellare et al. [9] 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 8.2.1 (Hierarchical Group Signature). *A hierarchical group signature scheme* $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$ *consists of four polynomial-time algorithms*

1. *The probabilistic* key generation algorithm HGKg *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 HGSig *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 HGVf *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 HGOpen *takes as input a tree T, a public map hpk, a private opening key hsk(β), a message m, and a candidate signature σ. It outputs an index $\alpha \in \beta$ or $\perp$.*

In the definition of HGSig 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 [9]. We assume that *hpk* and *hsk* map any input that is not a node of T to $\perp$ and that $\mathsf{HGOpen}(\cdot, \cdot, \perp, \cdot, \cdot) = \perp$.

We need to define what we mean by security for a hierarchical group signature scheme. We begin with anonymity. Consider Figure 8.2, 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.

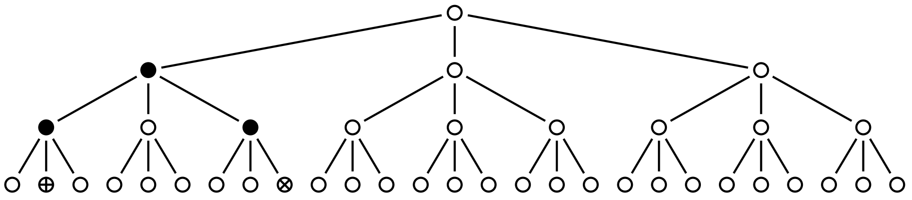


Figure 8.2: Nodes in black represent group managers able to distinguish between signatures by $S_{\alpha^{(0)}}$ and $S_{\alpha^{(1)}}$, the two leaves marked $\oplus$ and $\otimes$ respectively.

We define Experiment 8.2.1 to formalize these ideas. Throughout the experiment the adversary has access to an $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ 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 $\mathsf{HGSig}(\cdot, T, hpk, hsk(\alpha^{(b)}))$ oracle then computes the signature of m 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 $\mathsf{corrupt}$, choose and guess below allows the adversary to distinguish between the phases of the experiment.

Experiment 8.2.1 (Hierarchical Anonymity, $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$)**.**

```
$(hpk, hsk) \leftarrow \mathsf{HGKg}(1^\kappa, T)$
$state \leftarrow (hpk, hsk(\mathcal{L}(T)))$
$\mathcal{C} \leftarrow \emptyset$
$\alpha \leftarrow \emptyset$
repeat
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
    $(state, \alpha) \leftarrow A^{\mathsf{HGOpen}(T,hpk,hsk(\cdot),\cdot,\cdot)}(\mathsf{corrupt}, state, hsk(\alpha))$
until $\alpha \notin \mathcal{V}(T) \setminus \mathcal{C}$
$(state, \alpha^{(0)}, \alpha^{(1)}, m) \leftarrow A^{\mathsf{HGOpen}(T,hpk,hsk(\cdot),\cdot,\cdot)}(\mathsf{choose}, state)$
$\sigma \leftarrow \mathsf{HGSig}(m, T, hpk, hsk(\alpha^{(b)}))$
$d \leftarrow A^{\mathsf{HGOpen}(T,hpk,hsk(\cdot),\cdot,\cdot)}(\mathsf{guess}, state, \sigma)$
```

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 $\alpha_l^{(0)}$, $\alpha_l^{(1)}$, $l = t, \ldots, \delta - 1$, 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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle a query $(\alpha_l^{(0)}, m, \sigma)$ or $(\alpha_l^{(1)}, m, \sigma)$ return 0. Otherwise return d.

No generality is lost by having a $\mathsf{corrupt}$ 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 ω. In that case we may assume that $hsk(\omega)$ is never handed to the adversary, since the adversary fails in that case anyway. Similarly the $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle reduces to the GOpen oracle in [9]. Thus, our experiment reduces to the experiment for full anonymity given in [9] 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 [9] is similar to security against chosen message attacks as defined by Goldwasser, Micali and Rivest [52] for signatures. Their definition is given in Section 2.4.

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 HGVf 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 $\mathsf{HGSig}(\cdot, T, \mathit{hpk}, \mathit{hsk}(\cdot))$ 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 8.2.2 (Hierarchical Traceability, $\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$)**.**

```
$(\mathit{hpk}, \mathit{hsk}) \leftarrow \mathsf{HGKg}(1^\kappa, T)$
$\mathit{state} \leftarrow (\mathit{hpk}, \mathit{hsk}(\mathcal{V}(T) \setminus \mathcal{L}(T)))$
$\mathcal{C} \leftarrow \emptyset$
$\alpha \leftarrow \emptyset$
**repeat**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
    $(\mathit{state}, \alpha) \leftarrow A^{\mathsf{HGSig}(\cdot, T, \mathit{hpk}, \mathit{hsk}(\cdot))}(\mathsf{corrupt}, \mathit{state}, \mathit{hsk}(\alpha))$
**until** $\alpha \notin \mathcal{L}(T) \setminus \mathcal{C}$
$(m, \sigma) \leftarrow A^{\mathsf{HGSig}(\cdot, T, \mathit{hpk}, \mathit{hsk}(\cdot))}(\mathsf{guess}, \mathit{state})$
```

If $\mathsf{HGVf}(T, \mathit{hpk}, m, \sigma) = 0$ return 0. Define $\alpha_0 = \omega$ and define α_l for $l = 1, \ldots, \delta$ by $\alpha_l = \mathsf{HGOpen}(T, \mathit{hpk}, \mathit{hsk}(\alpha_{l-1}), m, \sigma)$. Return 1 if

1. $\alpha_l = \perp$ for some $0 < l \leq \delta$.

2. $\alpha_\delta \notin \mathcal{C}$ and the $\mathsf{HGSig}(\cdot, T, \mathit{hpk}, \mathit{hsk}(\cdot))$ oracle did not get a query (m, α_δ).

3. If there exists an index $\alpha \in \mathcal{V}(T)$ such that $\alpha \neq \alpha_l$ for $l = 1, \ldots, \delta$ and $\mathsf{HGOpen}(T, \mathit{hpk}, \mathit{hsk}(\alpha), m, \sigma) \neq \perp$.

Remark 8.2.2. The above definition differs from the one in [84] 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 $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$ oracle reduces to the GSig oracle in [9]. Thus, the definition reduces to the definition of full traceability in [9].

The advantages of the adversary in the experiments are defined by

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T) = |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},A}(\kappa, T) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},A}(\kappa, T) = 1]|$$

and

$$\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) = \mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) \ .$$

Definition 8.2.3 (Security of Hierarchical Group Signatures)**.** *A hierarchical group signature scheme* $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$ *is secure if for all trees* T *of polynomial size in* κ *with all leaves at the same depth, and all adversaries* $A \in \mathrm{PT}^*$ *the sum* $\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) + \mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T)$ *is negligible.*

An ordinary signature scheme $\mathcal{SS} = (\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf})$, with key generator SSKg, signature algorithm Sig, and verification algorithm Vf, can be viewed as a hierarchical group signature scheme $(\mathsf{SSKg}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{HGOpen})$ of depth 0. Definition 8.2.2 reduces to the definition of security against chosen message attacks as defined by Goldwasser, Micali, and Rivest [52].

Remark 8.2.4. 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.

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.

8.3 A Definition of an Optimistic Scheme

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 ciphertext is on this special form. The security of the encryption scheme used implies anonymity, since no adversary can distinguish ciphertexts 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 an encryption scheme. 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 $\omega = \alpha_0$ 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

$$(C_0, C_1, \ldots, C_{\delta-1}) = \left(E_{pk_{\alpha_0}}(pk_{\alpha_1}), E_{pk_{\alpha_1}}(pk_{\alpha_2}), \ldots, E_{pk_{\alpha_{\delta-1}}}(pk_{\alpha_\delta})\right) \ .$$

Note that each ciphertext C_l in the list encrypts the public key $pk_{\alpha_{l+1}}$ used to form the next ciphertext. 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 $pk \leftarrow D_{sk_\beta}(C_l)$.

If $\alpha_l = \beta$, then $pk \leftarrow pk_{\alpha_{l+1}}$. 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 ciphertext C_l about the encrypted plaintext, as long as the encryption scheme is semantically secure. There may be another way to deduce the content of C_l though. If the ciphertext 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 plaintext of C_l. This means that it can look at the chain of ciphertexts and extract information on the identity of the signer. We conclude that using the approach above, we need an encryption scheme which not only hides the plaintext, but also hides the public key used to form the ciphertext. An encryption scheme with this property is said to be *anonymous*. We give a definition in Section 8.4. The property of anonymity was discussed in [1] and studied extensively by Bellare et al. in [6].

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 $\omega = \alpha_0$ 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.

Optimistic Protocols

Many protocols may be designed to be more efficient if all participants are honest than if some participants are corrupt. Additional steps may be performed when dishonest participants are detected. Such protocols are called *optimistic*. Examples of such protocols include optimistic protocols for fair exchange [2], which uses a trusted third party that intervenes only in exceptional circumstances.

For real-world applications this is a realistic approach. In a typical execution all participants are honest, and the protocol is efficient. In the rare case of corrupt participants, the scheme can handle the abnormality and identify the misbehaving party. In other words, a corrupt player can cause extra work for other players, but it will itself be detected in the process. Especially when combined with secure hardware, optimistic protocols are an attractive option. If someone is ready to spend the necessary resources to break the hardware protection, all she achieves is to increase the work performed by other participants.

Optimistic Hierarchical Group Signatures

In a hierarchical group signature scheme any group manager can try to open a signature. There are two types of results of this. There could be no result at all, encoded by $\perp$. This means that the signer is not managed (not even indirectly) by the group manager who opened the signature. If on the other hand there is a result, it identifies the subtree below the group manager which contains the signer.

We now relax this and allow a signer to construct a signature which opens to $\perp$ by all group managers. However, we stress that the signer can not construct a signature that points out somebody else as the signer. If a signature opens to $\perp$, the group manager which tries to open the signature can ask a trusted party for help, and the trusted party can be implemented efficiently in a distributed way. In an actual application a group manager often knows (by asking its parent) whether it is supposed to be able to open a signature or not.

Participants

In addition to the participants of scheme for regular hierarchical group signatures, signers S_α and group managers denoted M_α, there is a trusted party $\mathfrak{T}$, which holds the trusted key that can open any signature.

Algorithms

In contrast to the opening algorithm found in a hierarchical group signature scheme the opening algorithm in an optimistic scheme comes in two flavors: the optimistic opening algorithm and the trusted opening algorithm. The former algorithm takes the same inputs as the opening algorithm in a hierarchical group signature scheme, and the latter takes a special trusted secret key as additional input.

Definition 8.3.1 (Scheme for Optimistic Hierarchical Group Signatures)**.** *An optimistic hierarchical group signature scheme $\mathcal{HGS}$ consists of the following five polynomial-time algorithms.*

1. *The probabilistic* key generation algorithm HGKg *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 $hpk, hsk : \mathcal{V}(T) \rightarrow \{0,1\}^*$ and a trusted opening key $sk_{\mathfrak{T}}$.*

2. *The probabilistic* signature algorithm HGSig *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 HGVf *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.

   We require that for every $m \in \{0,1\}^*$, every tree T of polynomial size in κ, every key triple output $(hpk, hsk, sk_{\mathfrak{T}})$ of $\mathsf{HGKg}(1^\kappa, T)$*, and every $\alpha \in \mathcal{L}(T)$,* $\mathsf{HGVf}(T, hpk, m, \mathsf{HGSig}(m, T, hpk, hsk(\alpha))) = 1$.

4. *The deterministic* optimistic opening algorithm HGOptOpen *takes as input a tree T, a public map hpk, a private opening key hsk(β), a message m, and a candidate signature σ. It outputs an index $\alpha \in \beta$ or $\perp$.*

5. *The deterministic* trusted opening algorithm HGTrustOpen *takes as input a tree T, public key map hpk, a trusted opening key $sk_{\mathfrak{T}}$, an index $\beta \in \mathcal{V}(T) \setminus \mathcal{L}(T)$, a message m and a candidate signature σ. It outputs an index $\alpha \in \beta$ or $\perp$.*

In the definition of HGSig 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 $\alpha = \alpha'$. We assume that *hpk* and *hsk* map any input that is not a node of T to $\perp$ and that $\mathsf{HGTrustOpen}(\cdot,\cdot,\cdot,\perp,\cdot,\cdot) = \perp$.

Definition of Security

As in the case of group signatures, we define two experiments: traceability and anonymity. As argued in [9] these two security properties cover any reasonable security property one could ask from a group signature scheme. When compared to signature schemes, traceability corresponds to unforgeability.

Traceability

The adversary is considered successful if it forges a signature that opens to $\perp$ or to an uncorrupted party. In the optimistic setting this is still the case for the trusted opening algorithm, but for the optimistic opening algorithm we require that the forged signature opens to an index. In other words, we explicitly allow the adversary to construct signatures which open to $\perp$ using the optimistic opening algorithm. The experiment below formalizes what we mean by a successful forgery.

Experiment 8.3.1 (Optimistic Hierarchical Traceability, $\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$).

```
$(hpk, hsk) \leftarrow \mathsf{HGKg}(1^\kappa, T)$, $state \leftarrow \big(hpk, hsk(\mathcal{V}(T) \setminus \mathcal{L}(T)), sk_{\mathfrak{T}}\big)$, $\mathcal{C} \leftarrow \emptyset$, $\alpha \leftarrow \emptyset$
repeat
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
  $(state, \alpha) \leftarrow A^{\mathsf{HGSig}(\cdot,T,hpk,hsk(\cdot))}(\mathsf{choose}, state, hsk(\alpha))$
until $\alpha \notin \mathcal{L}(T) \setminus \mathcal{C}$
$(m, \sigma) \leftarrow A^{\mathsf{HGSig}(\cdot,T,hpk,hsk(\cdot))}(\mathsf{guess}, state)$
```

If $\mathsf{HGVf}(T, hpk, m, \sigma) = 0$, then return 0. If not, then set $\alpha_0 \leftarrow \omega$ and define α_{i+1} for $i = 0, \ldots, \delta - 1$ by $\alpha_{i+1} \leftarrow \mathsf{HGTrustOpen}(T, hpk, sk_{\mathfrak{T}}, \alpha_i, m, \sigma)$. Return 1 if

1. $\exists \alpha \in \mathcal{V}(T) \setminus \mathcal{L}(T) \setminus \{\alpha_0, \ldots, \alpha_{\delta-1}\}$ such that $\mathsf{HGTrustOpen}(T,\ hpk,\ sk_{\mathfrak{T}},\ \alpha,\ m,\ \sigma) \neq \perp$,

2. $\alpha_\delta \notin \mathcal{C}$ and no query (m, α_δ) was given to the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-oracle.

3. $\exists \alpha \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ such that

$$\mathsf{HGOptOpen}(T, hpk, hsk(\alpha), m, \sigma) \notin \{\perp, \mathsf{HGTrustOpen}(T, hpk, sk_{\mathfrak{T}}, \alpha, m, \sigma)\} \ .$$

The advantage of an adversary A attacking the traceability of a group signature scheme $\mathcal{HGS}$ is defined as $\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) = \mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$.

Definition 8.3.2. *An optimistic group signature scheme $\mathcal{HGS}$ has* hierarchical traceability *if the advantage $\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$ is negligible as a function of κ for all trees T of polynomial size in κ and for all adversaries $A \in \mathrm{PT}^*$.*

Informally, these requirements ensure that: (1) the group managers for which the trusted opening algorithm output non-$\perp$ form a path from the root of the tree, (2) if the signature is a forgery, this path continues to a corrupted signer, and (3) the optimistic opening algorithm outputs $\perp$ or the same result as the trusted opening algorithm.

Anonymity

The definition of anonymity is identical to that for hierarchical group signatures except that the adversary has access to both a trusted opening oracle and an optimistic opening oracle.

Experiment 8.3.2 (Optimistic Hierarchical Anonymity, $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$).

```
$(hpk, hsk, sk_{\mathfrak{T}}) \leftarrow \mathsf{HGKg}(1^\kappa, T)$; $state \leftarrow (hpk, hsk(\mathcal{L}(T)))$; $\mathcal{C} \leftarrow \emptyset$; $\alpha \leftarrow \emptyset$
repeat
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\alpha\}$
  $(state, \alpha) \leftarrow$
     $A^{\mathsf{HGOptOpen}(T,hpk,hsk(\cdot),\cdot,\cdot),\mathsf{HGTrustOpen}(T,hpk,sk_{\mathfrak{T}},\cdot,\cdot,\cdot)}(\mathsf{corrupt}, state, hsk(\alpha))$
until $\alpha \notin \mathcal{V}(T) \setminus \mathcal{L}(T) \setminus \mathcal{C}$
$(state, \alpha^{(0)}, \alpha^{(1)}, m) \leftarrow$
   $A^{\mathsf{HGOptOpen}(T,hpk,hsk(\cdot),\cdot,\cdot),\mathsf{HGTrustOpen}(T,hpk,sk_{\mathfrak{T}},\cdot,\cdot,\cdot)}(\mathsf{choose}, state)$
$\sigma \leftarrow \mathsf{HGSig}(T, hpk, hsk(\alpha^{(b)}), m)$
$d \leftarrow A^{\mathsf{HGOptOpen}(T,hpk,hsk(\cdot),\cdot,\cdot),\mathsf{HGTrustOpen}(T,hpk,sk_{\mathfrak{T}},\cdot,\cdot,\cdot)}(\mathsf{guess}, state, \sigma)$
```

If $\alpha^{(0)} \notin \mathcal{L}(T)$ or $\alpha^{(1)} \notin \mathcal{L}(T)$ return 0. 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 $\alpha_l^{(0)}$, $\alpha_l^{(1)}$, $l = t, \ldots, \delta - 1$, 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 either its oracle for $\mathsf{HGOptOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ or its oracle for $\mathsf{HGTrustOpen}(T,\ hpk,\ sk_{\mathfrak{T}},\ \cdot,\ \cdot,\ \cdot)$ oracle a query $(\alpha_l^{(0)}, m, \sigma)$ or $(\alpha_l^{(1)}, m, \sigma)$, then return 0. Otherwise return d.

The advantage of an adversary A attacking the anonymity of an optimistic hierarchical group signature scheme is defined as

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T) = |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},A}(\kappa, T) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},A}(\kappa, T) = 1]| \ .$$

Definition 8.3.3. *An optimistic group signature scheme $\mathcal{HGS}$ has* optimistic hierarchical anonymity *if $\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T)$ is negligible as a function of κ for all trees of polynomial size in κ and for all adversaries $A \in \mathrm{PT}^*$.*

In the experiment above, B is the set of indices of the group managers that can distinguish signatures by $\alpha^{(0)}$ from signatures by $\alpha^{(1)}$. The definition captures the requirement that no other group manager or outsider can do this except with negligible probability.

Secure Optimistic Group Signature Scheme. Given these two definitions the definition of a secure optimistic hierarchical group signature scheme follows.

Definition 8.3.4. *An optimistic group signature scheme $\mathcal{HGS}$ is secure if it has hierarchical traceability and hierarchical anonymity.*

8.4 A Characterization of Anonymous Encryption Schemes

Anonymity

In some applications polynomial indistinguishability is not sufficient. The problem is that although the adversary can not learn anything about the encrypted plaintext, it may be able to tell which public key was used to compute the ciphertext. A encryption scheme 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 [6]. 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 8.4.1 (Anonymity, $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A}(\kappa)$).

```
$(pk_0, sk_0) \leftarrow \mathsf{Kg}(1^\kappa)$
$(pk_1, sk_1) \leftarrow \mathsf{Kg}(1^\kappa)$
$(m, state) \leftarrow A(pk_0, pk_1)$
$c \leftarrow E_{pk_b}(m)$
**return** $A(\mathsf{guess}, state, c)$
```

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 8.4.1 (Anonymity). *An encryption scheme $\mathcal{CS}$ is anonymous if the advantage*

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{CS},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A}(\kappa) = 1]|$$

is negligible in κ for all adversaries $A \in \mathrm{PT}^*$.

The property of anonymity is clearly useless if the encryption scheme 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 encryption scheme, then so is the encryption scheme where the encryption and decryption functions $c = E_{pk}(m)$ and $D_{sk}(c) = m$ are replaced by $(c, c') = E'_{pk}(m) = (E_{pk}(m), pk)$ and $D'_{sk}(c, c') = D_{sk}(c) = m$ respectively, and this is clearly not anonymous.

The following generalization follows by a similar argument as the generalization of polynomial indistinguishability. Denote by $\mathbf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-b}_{\mathcal{CS},A}(\kappa)$ 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 $c = (E_{pk_{i,b}}(m_{i,j}))_{i=1,j=1}^{\mu_1,\mu_2}$ instead of a single ciphertext. The lemma below follows by a straightforward hybrid argument.

Lemma 8.4.2. *If $\mathcal{CS}$ is polynomially indistinguishable, then for all adversaries $A \in \mathrm{PT}^*$ the absolute value*

$$|\Pr[\mathbf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mu_1-\mu_2-\mathsf{anon}-1}_{\mathcal{CS},A}(\kappa) = 1]|$$

is negligible in κ.

As explained above we need an anonymous encryption scheme to construct a hierarchical group signature scheme using our approach. The lemma below characterizes the set of encryption schemes which are both polynomially indistinguishable and anonymous.

Denote by $\mathbf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$ Experiment 2.4.8, but with the challenge ciphertext $E_{pk_b}(m)$ replaced by an element distributed according to a distribution D_κ, where $\mathcal{D}_{\mathrm{ind}} = \{D_\kappa\}$, and correspondingly for $\mathbf{Exp}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$. 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 8.4.2 ($\mathcal{D}_{\mathrm{ind}}$-Indistinguishability, $\mathbf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$).

```
$(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$
$(m_0, m_1, state) \leftarrow A(pk)$
$d \leftarrow A(T_{\mathcal{D}}(1^\kappa), state)$
return $d$
```

Experiment 8.4.3 ($\mathcal{D}_{\mathrm{ind}}$-Anonymity, $\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}(\kappa)$).

```
$(pk_0, sk_0) \leftarrow \mathsf{Kg}(1^\kappa)$
$(pk_1, sk_1) \leftarrow \mathsf{Kg}(1^\kappa)$
$(m, state) \leftarrow A(pk_0, pk_1)$
$d \leftarrow A(T_{\mathcal{D}}(1^\kappa), state)$
return $d$
```

Lemma 8.4.3. *Let $\mathcal{CS}$ be an encryption scheme which is both polynomially indistinguishable and anonymous. Then there exists an efficiently sampleable distribution $\mathcal{D}_{\mathrm{ind}}$ such that for all $A \in \mathrm{PT}^*$ the absolute value*

$$|\Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{ind}-b}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}(\kappa)}(\kappa) = 1]|$$

is negligible for $b \in \{0,1\}$. The reverse implication holds as well.

The intuition behind this lemma is that if an encryption scheme that is both polynomially indistinguishable and anonymous, then the ciphertexts 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 $A \in \mathrm{PT}^*$ such that

$$|\Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-0}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-1}(\kappa) = 1]|$$

is non-negligible which by the triangle inequality implies that

$$|\Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-b}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}(\kappa) = 1]|$$

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 2.4.8 defined as follows. On input pk it sets $pk_0 = pk$ generates $(pk_1, sk_1) = \mathsf{Kg}(1^\kappa)$ 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 $A(c, state)$. By construction $\mathbf{Exp}_{\mathcal{CS},A'}^{\mathsf{ind}-0}(\kappa)$ is identically distributed to $\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-0}(\kappa)$, and $\mathbf{Exp}_{\mathcal{CS},A'}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}(\kappa)$ is identically distributed to $\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{anon}-\mathcal{D}_{\mathrm{ind}}}(\kappa)$. This is a contradiction, since it implies that

$$|\Pr[\mathbf{Exp}_{\mathcal{CS},A'}^{\mathsf{ind}-0}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A'}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}(\kappa) = 1]|$$

is non-negligible.

Suppose next that $\mathcal{CS}$ is polynomially indistinguishable and anonymous. We define our candidate distribution $\mathcal{D}_{\mathrm{ind}}$ as follows. To generate a sample from $\mathcal{D}_{\mathrm{ind}}$, generate a key pair $(pk', sk') = \mathsf{Kg}(1^\kappa)$ and output an encryption $E_{pk'}(m')$, where m' is any fixed message. This implies that $\mathcal{D}_{\mathrm{ind}}$ is efficiently sampleable. Assume that

$$|\Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{ind}-b}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}(\kappa) = 1]|$$

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 8.4.1 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 $E_{pk_b}(m_0)$ for a randomly chosen $b \in \{0,1\}$ by the experiment. It hands $E_{pk_b}(m_0)$ 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 $\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa)$ and $\mathbf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{CS},A}(\kappa)$ are identically distributed to $\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa)$ and $\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_1}(\kappa)$ respectively. Thus

$$|\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_1}(\kappa) = 1]|$$

is non-negligible. From the anonymity of $\mathcal{CS}$ we have that

$$|\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_b}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{CS},A'_b}(\kappa) = 1]|$$

is negligible for $b \in \{0,1\}$. A hybrid argument then implies that

$$|\Pr[\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A'_0}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{CS},A'_1}(\kappa) = 1]|$$

is non-negligible for some $b \in \{0,1\}$. Without loss we assume $b = 0$. Denote by A'' the adversary in Experiment 2.4.8 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 $E_{pk}(m_0)$ or $E_{pk}(m')$, which it forwards to A. Finally, it returns the output of A. Since, $\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A''}(\kappa)$ is identically distributed to $\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_0}(\kappa)$ and $\mathbf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS},A''}(\kappa)$ is identically distributed to $\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{CS},A'_1}(\kappa)$, this contradicts the indistinguishability of $\mathcal{CS}$. □

Note that $\mathcal{D}_{\mathrm{ind}}$ depends on $\mathcal{CS}$ but is independent of all stochastic variables in the experiment. In the next section we prove that the probabilistic encryption scheme of Goldwasser and Micali [50] is anonymous.

Remark 8.4.4. Several standard probabilistic encryption schemes can be made anonymous by minor modifications, e.g., it is not hard to see that the ElGamal [42] encryption scheme is anonymous under the DDH-assumption if all parties employ the same group.

8.5 Proofs of Knowledge, Proofs, and Zero-Knowledge

Every relation $\mathcal{R}$ considered in this thesis corresponds to a language $L_{\mathcal{R}} \in \mathbf{NP}$ in the sense of Definition 2.4.16. Given two $\mathbf{NP}$-relations $\mathcal{R}_1$ and $\mathcal{R}_2$ we denote by $\mathcal{R}_1 \vee \mathcal{R}_2$ the relation defined by $((x_1, x_2), w) \in \mathcal{R}_1 \vee \mathcal{R}_2$ if and only if $(x_1, w) \in \mathcal{R}_1$ or $(x_2, w) \in \mathcal{R}_2$. Similarly we denote by $\mathcal{R}_1 \wedge \mathcal{R}_2$ the relation defined by $((x_1, x_2), (w_1, w_2)) \in \mathcal{R}_1 \vee \mathcal{R}_2$ if $(x_1, w_1) \in \mathcal{R}_1$ and $(x_2, w_2) \in \mathcal{R}_2$.

Computationally Convincing Proofs of Knowledge

The standard definition of a proof of knowledge given by Bellare and Goldreich [7] is too strict for our setting. The standard definition states that there must exist an algorithm, called the knowledge extractor, which for every $x \in \mathcal{R}$ and every prover P^* that convinces an honest verifier with non-negligible probability outputs a witness w such that $(x, w) \in \mathcal{R}$ 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 [41] 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 $\mathcal{R}$ 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 $\mathcal{R}$ 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 $(x, w) \in \mathcal{R}$ 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 $\mathrm{view}_{P^*}^{V}(\kappa, h, r_{\mathrm{p}}, r_{\mathrm{v}})$ 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 $\delta_{P^*}^{V}(\kappa, h, r_{\mathrm{p}}) = \Pr_{r_{\mathrm{v}}}[\mathrm{Acc}_V(\mathrm{view}_{P^*}^{V}(\kappa, h, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1]$. To simplify the exposition we sometimes omit the security parameter from our notation.

Definition 8.5.1 (Computationally Convincing Proof of Knowledge). *A protocol (P, V) is a computationally convincing proof of knowledge for an* **NP***-relation $\mathcal{R}$ with regards to the distribution of $h \in H$ if there exists a probabilistic oracle al-*

gorithm $\mathcal{X}^{(\cdot)}$ called the knowledge extractor, and a polynomial $p(\kappa)$ such that for all $P^* \in \mathrm{PT}^*$ 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*

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

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

Computationally Convincing Proofs

The standard definition of a proof introduced by Goldwasser, Micali, and Rackoff [51] 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 8.5.3 (Computationally Convincing Proof)**.** *A protocol (P, V) is a computationally convincing proof for an* **NP***-relation $\mathcal{R}$ with regards to the distribution of $h \in H$ if for all provers $P^* \in \mathrm{PT}^*$ the probability*

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

is negligible in κ.

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

Definition 8.5.4 (Proof)**.** *A protocol (P, V) is a proof for an* **NP***-relation $\mathcal{R}$ 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 8.5.5 (Soundness). *If (P, V) is a computationally convincing proof of knowledge for an* **NP***-language $L_{\mathcal{R}}$ 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

$$\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_{\mathcal{R}} \wedge \delta_{P^*}^{V}(h, r_{\mathrm{p}}, r_{\mathrm{v}}) \geq \kappa^{-c}] < \kappa^{-c} \ . \tag{8.1}$$

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

$$\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_{\mathcal{R}} \wedge \delta_{P^*}^{V}(h, r_{\mathrm{p}}) \geq \kappa^{-c}] \geq \kappa^{-c} \ ,$$

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

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

is overwhelming. The union bound implies that

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

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

Next we prove the statement in the proposition using Equation (8.1). 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

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

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

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

From this we conclude that $\Pr[I_{P^*}(h, r_{\mathrm{p}}) \notin L_{\mathcal{R}} \wedge \delta_{P^*}^{V}(h, r_{\mathrm{p}}) \geq \frac{1}{2\kappa^{c}}] \geq \frac{1}{2\kappa^{c}}$. This contradicts Equation (8.1), and the proposition holds. □

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 [51]. 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 $\mathrm{hview}_P^V(h, x, w)$ 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 $\mathrm{hview}_P^V(h, x, w, c)$.

Definition 8.5.6 (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 $x \in L_{\mathcal{R}}$, the distributions of* $\mathrm{hview}_P^V(h, x, w)$ *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.*

Completeness

Let $\mathcal{R}$ 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 $(x, w) \in \mathcal{R}$.

Definition 8.5.7 (Completeness). *A computationally convincing proof of knowledge (P, V) has completeness p if for all special parameters $h \in H$ and all $(x, w) \in \mathcal{R}$ 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.*

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 8.5.8 (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 $(x, w) \in \mathcal{R}$ and $c \in C$, the distributions of $S(\kappa, x, c)$ and $\mathrm{hview}_P^V(\kappa, x, w, c)$ 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 8.5.9 (Special Soundness)**.** *Let $C = C_1 \times \cdots \times C_k$. A C-three-move protocol (P, V) for a relation $\mathcal{R}$ 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 [38].

Definition 8.5.10 (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 8.5.1. *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 8.5.2. *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 8.5.11. *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

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

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 ${r_\mathrm{v}}'$ 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 8.5.2.

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 8.5.2 is satisfied. □

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 [45].

This idea can be explained as follows. Let $\mathcal{R}$ be an **NP**-relation and suppose that one party holds a witness w of some joint input x such that $(x, w) \in \mathcal{R}$. Let (P, V) be a C-Σ-protocol for the language $L_\mathcal{R}$. 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 $c \leftarrow H(x, \alpha)$. 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 $c \leftarrow H(x, \alpha)$. 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 $(\alpha, c, d) \leftarrow P^{\mathsf{O}(m,\cdot)}$, i.e., it includes the message to be signed as a prefix to its random oracle. The signature is verified by checking that $V^{\mathsf{O}(m,\cdot)}(\alpha, c, d) \leftarrow 1$. Thus, the triple (α, c, d) may be viewed as a signature of m computed by the party holding a witness w such that $(x, w) \in \mathcal{R}$. For this to make any sense it must of course be infeasible to find a witness w such that $(x, w) \in \mathcal{R}$ given only x.

In Chapters 10 and 11 we analyze two hierarchical group signature schemes in the random oracle model, and use the above notation.

Chapter 9

A Constructions under General Assumptions

9.1 About the Construction

In this section 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 9.1.1. *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 $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$.and prove its security.

Building Blocks

Our construction is based on three primitives: the group signature scheme of Bellare et al. [9], the public key encryption scheme of Goldwasser and Micali [50], and a non-interactive zero-knowledge proof as defined in Section 2.4. Of these we use the first and last in a blackbox way. Bellare et al. [9] prove the following theorem.

Theorem 9.1.2. *If there exists a family of trapdoor permutations, then there exists a secure group signature scheme* $\mathcal{GS} = (\mathsf{GKg}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{GOpen})$.

As explained in Section 8.2 every group signature scheme is also a hierarchical group signature scheme, and our definition of security reduces to the definition of security given in [9] for group signatures. The definition of a trapdoor permutation family is given in Section 2.4.

Recall from Section 2.4 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. [9] 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 [9], i.e., single statement simulation soundness suffices. A precise definition of the type of NIZK we use is given in Section 2.4.

De Santis et al. [79] extend the results in [43] and [77] and prove the following theorem.

Theorem 9.1.3. *If there exists a family of trapdoor permutations, then there exists a NIZK for any language in* **NP**.

The probabilistic encryption scheme of Goldwasser and Micali [50] is defined in Section 9.1.4. Goldwasser and Micali prove that their encryption scheme is polynomially indistinguishable, i.e., it satisfies Definition 2.4.12, but as explained in the previous section we need an anonymous encryption scheme. Let us now review the Goldwasser-Micali encryption scheme.

The Goldwasser-Micali Encryption Scheme

Goldwasser and Micali [50] construct a public key encryption scheme 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 encryption scheme $\mathcal{GM} = (\mathsf{Kg}^{\mathsf{gm}}, E^{\mathsf{gm}}, D^{\mathsf{gm}})$ of Goldwasser and Micali [50] using the family of trapdoor permutations $\mathcal{T} = \mathcal{F} \times \mathcal{F}^{-1}$ and hardcore bit $\mathcal{B}$ can be defined as follows. The key generator $\mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$ simply draws (f, f^{-1}) from $\mathcal{T}$ and sets $(pk, sk) \leftarrow (f, f^{-1})$. To compute a ciphertext $E^{\mathsf{gm}}_{pk}(m)$ of a bit $m \in \{0,1\}$, a sample $r \leftarrow_R \mathrm{Dom}(f)$ is drawn and $(f(r), \mathcal{B}(r) \oplus m)$ is the ciphertext. To decrypt a ciphertext (c, c'), we compute $D^{\mathsf{gm}}_{sk}(c, c') = \mathcal{B}(f^{-1}(c)) \oplus c'$. Goldwasser and Micali essentially show the following theorem.

Theorem 9.1.4. *If $\mathcal{F}$ is a trapdoor permutation family with hard-core bit $\mathcal{B}$, then $\mathcal{GM}$ is polynomially indistinguishable.*

Goldreich and Levin [48] show how to construct a family of trapdoor permutations $\mathcal{F}$ 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 10.2.8 we know that this is as secure as the original scheme.

Lemma 9.1.5. *If $\mathcal{F}$ is a trapdoor permutation family with hard-core bit $\mathcal{B}$, then $\mathcal{GM}$ is anonymous.*

Proof. Suppose that $\mathcal{GM}$ is not anonymous. Let $U_{\kappa+1}$ be the uniform and independent distribution over $\{0,1\}^{\kappa+1}$. Then for some adversary $A \in \mathrm{PT}^*$,

$$|\Pr[\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{GM},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-U_{\kappa+1}}_{\mathcal{GM},A}(\kappa) = 1]|$$

is non-negligible for a fixed $b \in \{0,1\}$. Without loss we assume $b = 0$. Since $\mathcal{GM}$ is a bitwise encryption scheme, we may without loss assume that $m_0 = 0$ and $m_1 = 1$. Let $m \in \{0,1\}$ be randomly chosen. Then a ciphertext $E_{pk}(m) = (f(r), \mathcal{B}(r) \oplus m)$ is distributed according to $U_{\kappa+1}$, since the function f is a permutation and $\mathcal{B}(r) \oplus m$ is uniformly and independently distributed. A trivial averaging argument then implies that $|\Pr[\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{GM},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-1}_{\mathcal{GM},A}(\kappa) = 1]|$ is non-negligible which is a contradiction. □

We remind the reader at this point that we define a trapdoor permutation family to have domain $\{0,1\}^\kappa$.

Notation

We let $\mathcal{F}$ denote a family of trapdoor permutations with a hard-core bit $\mathcal{B}$, and assume that a Goldwasser-Micali encryption scheme $\mathcal{GM}$ has been constructed from this. We denote by $\mathcal{GS} = (\mathsf{GKg}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{GOpen})$ the group signature scheme of Bellare et al. also constructed from $\mathcal{F}$. 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 $\mathcal{F}$ to construct a NIZK for a language L_{HGS} defined below.

The Basic Idea of the Construction

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 $gsk(\omega)$. Instead, each group manager is given a key pair (pk_β, sk_β) of the $\mathcal{GM}$ encryption scheme. 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., $\alpha_0 = \omega$ and $\alpha_\delta = \alpha$. Then the signer forms a chain of ciphertexts $C \leftarrow (E_{pk_{\alpha_0}}(pk_{\alpha_1}), \ldots, E_{pk_{\alpha_{\delta-1}}}(pk_{\alpha_\delta}))$. Finally, it forms a NIZK π that the chain of ciphertexts 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 ciphertext. 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 $C' \pm gets(E_{pk}(pk_{\alpha_0}), \ldots, E_{pk}(pk_{\alpha_{\delta-1}}))$.

9.2 The Algorithms of the Scheme

Algorithm 9.2.1 (Key Generation, $\mathsf{HGKg}(1^\kappa, T)$)**.** The key generation algorithm is defined as follows.

1. Generate a random string $\xi \in \{0,1\}^*$ sufficiently long for a NIZK based on $\mathcal{F}$ for the language L_{HGS} defined below. Generate $(pk, sk) \leftarrow \mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$.
2. For each node α in $\mathcal{V}(T)$, compute $(pk_\alpha, sk_\alpha) \leftarrow \mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$.
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 $\alpha_0 = \omega$ and $\alpha_\delta \in \mathcal{L}(T)$. Define I to map anything else to $\perp$. Denote by $T_{\mathcal{GS}}$ the tree with root ω and leaves $\mathcal{L}(T)$.
4. Run $(gpk, gsk) \leftarrow \mathsf{GKg}(1^\kappa, T_{\mathcal{GS}})$ to generate keys for a group signature scheme, and set $(hpk(\alpha), hsk(\alpha)) \leftarrow ((pk_\alpha, gpk(\alpha)), gsk(\alpha))$ for $\alpha \in \mathcal{L}(T)$.
5. Define the keys of the root ω by $(hpk(\omega), hsk(\omega)) \leftarrow ((\xi, pk, pk_\omega, gpk(\omega)), sk_\omega)$ and set $(hpk(\beta), hsk(\beta)) \leftarrow (pk_\beta, sk_\beta)$ for $\beta \in \mathcal{V}(T) \setminus \mathcal{L}(T)$, $\beta \neq \omega$. Note that $hsk(\omega)$ does not contain $gsk(\omega)$.
6. Output (hpk, hsk).

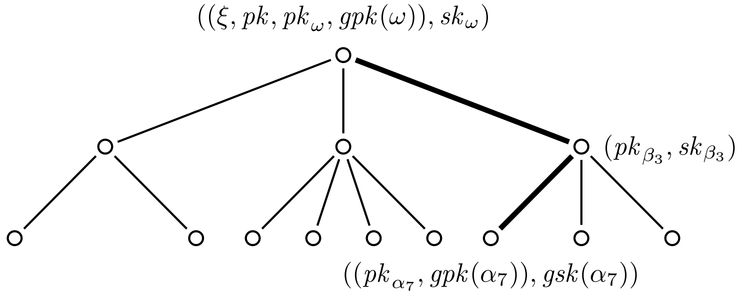

$((\xi, pk, pk_\omega, gpk(\omega)), sk_\omega)$
$(pk_{\beta_3}, sk_{\beta_3})$
$((pk_{\alpha_7}, gpk(\alpha_7)), gsk(\alpha_7))$

Figure 9.1: The figure illustrates the public and private keys along a path in the tree of keys corresponding to Figure 3.1. The edges along the path have thick edges. Each node contains a pair of public and private keys.

The result of running the above algorithm is illustrated in Figure 9.1. 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 $pk_0, \ldots, pk_\delta, gsk(\alpha)$ and random strings $r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta$ such that

$$\alpha_0 = \omega \ ,$$
$$(C_l, C'_l) = (E_{pk_{\alpha_l}}((pk_{\alpha_{l+1}}, r'_l), r_l), E_{pk}(pk_{\alpha_l}, r'_l)) \text{ for } l = 0, \ldots, \delta - 1 \ ,$$
$$\alpha = I(pk_{\alpha_0}, \ldots, pk_{\alpha_{\delta-1}}) \ , \quad \text{and } \sigma = \mathsf{GSig}_{r_\delta}(m, T_{\mathcal{GS}}, gpk, gsk(\alpha)) \ .$$

The NIZK now enables us to give a succinct description of the signature algorithm.

Algorithm 9.2.2 (Signing, $\mathsf{HGSig}(m, T, hpk, hsk(\alpha))$)**.** Let $\alpha_0, \ldots, \alpha_\delta$ be the path to the signer S_α, i.e., $\omega = \alpha_0$ and $\alpha_\delta = \alpha$. Choose r_i and r'_i randomly and compute

$$\begin{aligned}
\sigma &= \mathsf{GSig}_{r_\delta}(m, T_{\mathcal{GS}}, gpk, gsk(\alpha)) \\
(C_l, C'_l) &= (E^{\mathsf{gm}}_{pk_{\alpha_l}}((pk_{\alpha_{l+1}}, r'_l), r_l), E^{\mathsf{gm}}_{pk}(pk_{\alpha_l}, r'_l)) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
\pi &= P_{\mathrm{hgs}}((T, hpk, m, \sigma, C, C'), \\
&\qquad (pk_{\alpha_0}, \ldots, pk_{\alpha_\delta}, gsk(\alpha), r_0, r'_0, \ldots, r_{\delta-1}, r'_{\delta-1}, r_\delta), \xi) \ .
\end{aligned}$$

Then output (σ, C, C', π).

Algorithm 9.2.3 (Verification, $\mathsf{HGVf}(T, hpk, m, (\sigma, C, C', \pi))$)**.** On input a candidate signature (σ, C, C', π) output $V_{\mathrm{hgs}}((T, hpk, m, \sigma, C, C'), \pi, \xi)$.

Algorithm 9.2.4 (Opening, $\mathsf{HGOpen}(T, gpk, gsk(\beta), m, (\sigma, C, C', \pi))$)**.** Let $l \leftarrow \mathsf{level}_T(\beta)$. If $\mathsf{HGVf}(T,\ hpk,\ m,\ (\sigma, C, C', \pi)) = 0$, then return $\perp$. Otherwise, compute $(pk_\alpha,\ r') = D^{\mathsf{gm}}_{sk_\beta}(C_l)$ and verify that $C'_l = E^{\mathsf{gm}}_{pk}(pk_\beta, r')$ and $\alpha \in \beta$. Return α if this is the case and return $\perp$ otherwise.

Remark 9.2.1. The scheme described above differs from the scheme in [84]. As explained in Remark 8.2.2 the original definition in [84] 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 ciphertexts 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 9.2.2. In Section 9.4 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 9.2.3. 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 [8] 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 $f(r) < 2^\kappa$. This implies that $f(r) < N$ for all κ-bit moduli N and that the first part of a Goldwasser-Micali ciphertext 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 $f(r) < 2^\kappa$ with probability at least $1/4$. To change the construction in this way we do need to modify the definition of a public key encryption scheme such that it allows the encryption algorithm to fail with negligible probability.

9.3 Proof of Security

We prove the following lemma on the security of our construction, from which Theorem 9.1.1 follows immediately.

Lemma 9.3.1. *If $\mathcal{F}$ 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 $\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T) \geq 1/\kappa^c$ 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 8.2.1, with $\mathcal{HGS}$ to A. It also plays the role of adversary in Experiment 8.2.1 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 $(\xi, \mathsf{simstate})$. The string ξ is used instead of a random string. Recall that $T_{\mathcal{GS}}$ denotes the very simple tree having ω, 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 $gsk(\omega)$ 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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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

$$\mathsf{HGVf}(T, hpk, m, (\sigma, C, C', \pi)) = 1 \ .$$

If not it returns $\perp$. If so it asks its $\mathsf{GOpen}(T_{\mathcal{GS}}, gpk, gsk(\cdot), \cdot, \cdot)$-oracle the question (ω, 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., $\alpha_0 = \omega$ and $\alpha_\delta = \alpha$. Let β be on depth l. Then A' instructs the $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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 ciphertexts $C_0, C_0', \ldots, C_{t-1}, C_{t-1}'$ honestly. It chooses random samples $C_t, C_t', \ldots, C_{\delta-1}, C_{\delta-1}'$ according to the distribution $\mathcal{D}_{\mathrm{ind}}$ guaranteed to exist by Lemma 8.4.3. Here we in fact need to apply Lemma 10.2.8 and Lemma 8.4.2 to increase the length of messages that can be encrypted, and then apply Lemma 8.4.3, but we abuse notation below. Then it invokes S_{hgs} of the NIZK on $((T, hpk, m, \sigma, C, C'), \xi, \mathsf{simstate})$ 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 $H^b_{\mathrm{c,o,p}}$ the machine that on input κ simply simulates Experiment 8.2.1 with $\mathcal{HGS}$ to A and outputs the result. Denote by $H^b_{\mathrm{c,o}}$ the machine which is identical to $H^b_{\mathrm{c,o,p}}$ 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, $H^b_{\mathrm{c,o}}$ simulates Experiment 8.2.1 with $\mathcal{HGS}$ to A. We also define H^b_{c} to be identical to $H^b_{\mathrm{c,o}}$ except that it simulates the $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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 $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{GS},A'}(\kappa)$. This gives us a chain of distributions $H^b_{\mathrm{c,o,p}}, H^b_{\mathrm{c,o}}, H^b_{\mathrm{c}}, H^b$ starting with $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa)$ and ending with $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{GS},A'}(\kappa)$. In the following claims we show that the distance between each pair of distributions is negligible.

Claim 1. There exists a negligible function $\varepsilon_1(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c,o,p}}(\kappa) = 1] - \Pr[H^b_{\mathrm{c,o}}(\kappa) = 1]| < \varepsilon_1(\kappa) \ .$$

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 2.4.11 or 2.4.12. 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 2.4.11 or 2.4.12. More precisely, it hands $(T, hpk, m, \sigma, C, C')$ 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 $\mathbf{Exp}^{\mathsf{ad-ind}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)$ and $\mathbf{Exp}^{\mathsf{ad-ind}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)$ respectively. The adaptive zero-knowledge property of the NIZK implies that there exists a negligible function $\varepsilon_1(\kappa)$ such that

$$|\mathbf{Exp}^{\mathsf{ad-ind}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa) - \mathbf{Exp}^{\mathsf{ad-ind}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)| < \varepsilon_1(\kappa) \ ,$$

and the claim follows. □

Claim 2. There exists a negligible function $\varepsilon_2(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c,o}}(\kappa) = 1] - \Pr[H^b_{\mathrm{c}}(\kappa) = 1]| < \varepsilon_2(\kappa) \ .$$

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 $\mathsf{GOpen}(T_{\mathcal{GS}}, gpk, gsk(\cdot), \cdot, \cdot)$-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 2.4.13 (or the soundness property of Definition 2.4.18), 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 encryption schemes 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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle. Define $\alpha_0 = \omega$ and define α_l for $l = 1, \ldots, \delta$ by

$$\alpha_l = \mathsf{HGOpen}(T, hpk, hsk(\alpha_{l-1}), m, (\sigma, C, C', \pi)) \ .$$

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 $\beta' = \mathsf{HGOpen}(T, hpk, hsk(\beta), m, (\sigma, C, C', \pi))$ with $\beta' \in \beta$. This is one of the two places in the proof where we use the ciphertexts 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 encryption scheme it would be easy to compute such a signature. Thus, we must argue that the ciphertexts in the list C' prohibits the construction of such signatures.

By the definition of the open algorithm the above anomaly can only happen if $(pk_{\beta'}, r') = D^{\mathsf{gm}}_{sk_\beta}(C_l)$ and $E_{pk}(pk_\beta, r') = C'_l$. This is a contradiction and we conclude that the claim is true, since we know that $E^{\mathsf{gm}}_{sk}(C'_l) = pk_{\alpha_l}$. □

Claim 3. There exists a negligible function $\varepsilon_3(\kappa)$ such that

$$|\Pr[H^b_{\mathrm{c}}(\kappa) = 1] - \Pr[H^b(\kappa) = 1]| < \varepsilon_3(\kappa) \ .$$

Proof. This follows from the polynomial indistinguishability and the anonymity of the Goldwasser-Micali encryption scheme $\mathcal{GM}$ using Theorem 9.1.4, Lemma 9.1.5, and Lemma 8.4.3 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_{c}^b 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 8.4.3. 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_{c}^b. By construction, $A_{\mathrm{ind},t-1}(\kappa)$ and $A_{\mathrm{ind},\delta-1}(\kappa)$ are identically distributed to $H_{\mathrm{c}}^b(\kappa)$ 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 section, Experiment 8.4.2, run with $\mathcal{GM}$. It chooses $\beta_\delta^{(b)}$ randomly from $\mathcal{L}(T)$. Let $\beta_0, \ldots, \beta_\delta$ be the corresponding path, i.e., $\omega = \beta_0$ and $\beta_\delta = \beta_\delta^{(b)}$. Then it simulates $A^{j-1}_{\mathrm{ind},l-1}$ 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

$$\begin{aligned}
&|\Pr[\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{GM},A_{\mathrm{ind}}}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-\mathcal{D}_{\mathrm{ind}}}_{\mathcal{GM},A_{\mathrm{ind}}}(\kappa) = 1]| \\
&= |\Pr[A^0_{\mathrm{ind},l-1} = 1 \wedge \alpha^{(b)} = \beta^{(b)}] - \Pr[A_{\mathrm{ind}^1,l-1} = 1 \wedge \alpha^{(b)} = \beta^{(b)}]| \\
&= (1/|\mathcal{L}(T)|)|\Pr[A^0_{\mathrm{ind},l-1} = 1] - \Pr[A^1_{\mathrm{ind},l-1} = 1]| \geq 1/(|\mathcal{L}(T)|\delta\kappa^c) \enspace .
\end{aligned}$$

The first equality follows by construction. The second equality follows by independence. In view of Theorem 9.1.4, Lemma 9.1.5, and Lemma 8.4.3 this contradicts either the indistinguishability or the anonymity of $\mathcal{GM}$. 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

$$|\Pr[H^b_{\mathrm{c,o,p}}(\kappa) = 1] - \Pr[H^b(\kappa) = 1]| < \varepsilon_1(\kappa) + \varepsilon_2(\kappa) + \varepsilon_3(\kappa) \enspace ,$$

which gives

$$\begin{aligned}
&|\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{GS},A'}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{GS},A'}(\kappa) = 1]| \\
&\geq |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},A}(\kappa) = 1]| \\
&\quad -2(\varepsilon_1(\kappa) + \varepsilon_2(\kappa) + \varepsilon_3(\kappa)) \enspace .
\end{aligned}$$

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 $\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) \geq 1/\kappa^c$ 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 8.2.2, with $\mathcal{HGS}$ to A. It also plays the role of the adversary in Experiment 8.2.2 with $\mathcal{GS}$.

The key generation is simulated as follows. First the NIZK simulator is invoked to compute a reference string with a trapdoor $(\xi, \mathsf{simstate})$. The string ξ is used instead of a random string. Recall that $T_{\mathcal{GS}}$ denotes the tree having ω, the root of T, as root, and children $\mathcal{L}(T)$. The adversary A' waits until it receives the keys $(gpk(\omega), gsk(\omega))$ 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 $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-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 HGSig. Finally, it uses the NIZK simulator S_{hgs} on input $((T, hpk, m, \sigma, C, C'), \xi, \mathsf{simstate})$ 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 $H_{\pi,\mathrm{p}}$ the machine that simulates Experiment 8.2.2 with $\mathcal{HGS}$ to A, and outputs 1 if the experiment outputs 1 and the output $(m, (\sigma, C, C', \pi))$ of A satisfies $(T, hpk, m, \sigma, C, C') \in L_{\mathrm{HGS}}$. Consider such an execution and define $\alpha_0 \leftarrow \omega$ and α_l for $l = 1, \ldots, \delta$ by

$$\alpha_l = \mathsf{HGOpen}(T, hpk, hsk(\alpha_{l-1}), m, (\sigma, C, C', \pi)) \ .$$

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 $\beta' = \mathsf{HGOpen}(T, hpk, hsk(\beta), m, (\sigma, C, C', \pi))$ with $\beta' \in \beta$. If it did, then we would have $(pk_{\beta'}, r') = D^{\mathsf{gm}}_{sk_\beta}(C_l)$ and $E^{\mathsf{gm}}_{pk}(pk_\beta, r') = C'_l$, but this contradicts the fact that $D^{\mathsf{gm}}_{sk}(C'_l) = pk_{\alpha_l}$. 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 $H_{\pi,\mathrm{p}}$ except that it simulates the answers from the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-oracle to A precisely as A' does.

Claim 5. There exists a negligible function $\varepsilon_1(\kappa)$ such that

$$\Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1] \leq \Pr[H_{\pi,\mathrm{p}}(\kappa) = 1] + \varepsilon_1(\kappa) \ .$$

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 $(T, hpk, m, \sigma, C, C') \in L_{\mathrm{HGS}}$. From the soundness of the NIZK follows that $\Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1 \wedge \overline{E_{\pi,\mathrm{p}}}]$ is negligible. By definition we have that $\Pr[H_{\pi,\mathrm{p}}(\kappa) = 1] = \Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa) = 1 \wedge E_{\pi,\mathrm{p}}]$. The claim follows. □

Claim 6. There exists a negligible function $\varepsilon_2(\kappa)$ such that

$$|\Pr[H_\pi(\kappa) = 1] - \Pr[H_{\pi,\mathrm{p}}(\kappa) = 1]| \quad < \quad \varepsilon_2(\kappa) \ .$$

Proof. The claim follows from the adaptive zero-knowledge of the NIZK. We construct an adversary A_{adzk} against the adaptive zero-knowledge, Experiments 2.4.11 and 2.4.12, 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 $((T, hpk, m, \sigma, C, C'), \xi, \mathsf{simstate})$ 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

$$\begin{aligned}&|\Pr[H_{\pi,\mathrm{p}}(\kappa)=1]-\Pr[H_{\pi}(\kappa)=1]|\\&=|\Pr[\mathbf{Exp}^{\mathsf{adzk}-0}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)=1]-\Pr[\mathbf{Exp}^{\mathsf{adzk}-1}_{\pi_{\mathrm{hgs}},A_{\mathrm{adzk}}}(\kappa)=1]|<\varepsilon_2(\kappa)\ ,\end{aligned}$$

for some negligible function $\varepsilon_2(\kappa)$. □

Claim 7. $\Pr[H_\pi(\kappa)=1] \leq \Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{GS},A'}(\kappa)=1]$.

Proof. All inputs to A in the simulation of H_π are identically distributed to the corresponding inputs in Experiment 8.2.2. 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

$$\begin{aligned}&\Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa)=1]\leq\Pr[H_{\pi,\mathrm{p}}(\kappa)=1]+\varepsilon_1(\kappa)\\&\leq Pr[H_\pi(\kappa)=1]+\varepsilon_1(\kappa)+\varepsilon_2(\kappa)\leq\Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{GS},A'}(\kappa)=1]+\varepsilon_1(\kappa)+\varepsilon_2(\kappa)\ .\end{aligned}$$

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. □

9.4 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 $\mathcal{SS}=(\mathsf{SSKg},\mathsf{Sig},\mathsf{Vf})$ be a signature scheme. For each group manager M_α and signer S_α, $(spk_\alpha,ssk_\alpha)\leftarrow\mathsf{SSKg}(1^\kappa)$, and keys $(pk_\alpha,sk_\alpha)\leftarrow\mathsf{Kg}^{\mathsf{gm}}(1^\kappa)$ to a Goldwasser-Micali encryption scheme are generated. Then for each child α of $\beta\in\mathcal{V}(T)$, $\sigma_\beta(\alpha)\leftarrow\mathsf{Sig}_{ssk_\beta}(spk_\alpha,pk_\alpha)$ is computed. For each $\alpha\in\mathcal{V}(T)\setminus\mathcal{L}(T)\setminus\{\omega\}$ we set $hpk(\alpha)\leftarrow(spk_\alpha,pk_\alpha,\sigma_\beta(\alpha))$, where $\alpha\in\beta$, and $hsk(\alpha)\leftarrow sk_\alpha$. For each $\alpha\in\mathcal{L}(T)$ set $hpk(\alpha)\leftarrow(spk_\alpha,pk_\alpha,\sigma_\beta(\alpha))$, where $\alpha\in\beta$, and $hsk(\alpha)\leftarrow(ssk_\alpha,sk_\alpha)$. For the root ω we set $hpk(\omega)\leftarrow(spk_\omega,pk_\omega)$ and $hsk(\omega)\leftarrow sk_\omega$.

Consider a signer S_α corresponding to a path $\alpha_0, \ldots, \alpha_\delta$, where $\alpha_0 = \omega$ and $\alpha_\delta = \alpha$. To sign a message m the signer computes

$$\begin{aligned} C_l &\leftarrow E^{\mathsf{gm}}_{pk_{\alpha_l}}((\sigma_{\alpha_l}(\alpha_{l+1}), r'_l), r_l) \text{ ,for } l = 0, \ldots, \delta - 1 \text{ ,} \\ C'_l &\leftarrow E^{\mathsf{gm}}_{pk}(pk_{\alpha_l}, r'_l) \text{ ,for } l = 0, \ldots, \delta - 1 \text{ ,} \mathit{and} \\ C_\delta &\leftarrow E^{\mathsf{gm}}_{pk_{\alpha_\delta}}(\mathsf{Sig}_{ssk_\alpha}(m))) \end{aligned}$$

and provides a NIZK π that (C, C') is formed as above with $pk_{\alpha_0} = pk_\omega$. 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 $(\sigma, r') \leftarrow D^{\mathsf{gm}}_{sk_\beta}(C_l)$. If σ is equal to $\sigma_\beta(\alpha)$ for some $\alpha \in \beta$ and $E^{\mathsf{gm}}_{pk}(pk_\beta, 1) = C'_l$, then it returns α and otherwise $\perp$.

This construction is a strict generalization of the construction in [9] except that we require that the encryption scheme 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 encryption scheme of Sahai [77], and the group signature scheme of Bellare et al. [9], 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.

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 10

A Construction under Standard Assumptions

10.1 About the Construction

In this chapter we construct an almost practical scheme for hierarchical group signatures. 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 ElGamal encryption scheme, the Cramer-Shoup encryption scheme, the Cramer-Shoup signature scheme, the Chaum-van Heijst-Pfitzmann hash function, and the Shamir hash function.

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 10.2 the definition and basic properties of the ElGamal encryption scheme [42]. It is well known that the ElGamal encryption scheme 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 ElGamal encryption scheme.

Recall from Section 10.2 the definition of the Cramer-Shoup signature scheme [40]. 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 $\omega = \alpha_0$ to the leaf $\alpha = \alpha_\delta$.

To form a signature of a message m the signer first computes a chain of ciphertexts on the form

$$\begin{aligned}&((u_0, v_0, u'_0, v'_0), \ldots, (u_{\delta-1}, v_{\delta-1}, u'_{\delta-1}, v'_{\delta-1}))\\ &= (E_{y_{\alpha_0}}(y_{\alpha_1}), E_{y_{\alpha_0}}(1), \ldots, E_{y_{\alpha_{\delta-1}}}(y_{\alpha_\delta}), E_{y_{\alpha_{\delta-1}}}(1)) \enspace .\end{aligned}$$

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 ciphertexts 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 ciphertexts. 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

$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C(\sigma_\alpha), \pi(m)) \enspace .$$

Recall from Section 2.2 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 ciphertext (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.

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 10.2 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

$$(y_3^{t'''} g_3^{s'''}, y_3^{s'''} y_\alpha) \enspace , \quad (y_2^{t''} g_2^{s''}, y_2^{s''} g_2^{y_\alpha}) \enspace , \quad \text{and} \quad (y_1^{t'} g_1^{s'}, y_1^{s'} g_1^{g_2^{y_\alpha}}) \enspace ,$$

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 [82] we can give a reasonably practical cut-and-choose proof that the elements hidden in two such commitments are identical.

Recall from Section 10.2 that the collision-free Chaum-Heijst-Pfitzmann hash function [35] 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 ElGamal 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

$$((\mu_0, \nu_0), \ldots, (\mu_{\delta-1}, \nu_{\delta-1})) = ((y_2^{t''_0} g_2^{s''_0}, y_2^{s''_0} h_1^{y_{\alpha_1}}), \ldots, (y_2^{t''_{\delta-1}} g_2^{s''_{\delta-1}}, y_2^{s''_{\delta-1}} h_\delta^{y_{\alpha_\delta}})) \enspace .$$

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.,

$$\left(\prod_{i=0}^{\delta-1} \mu_i, \prod_{i=0}^{\delta-1} \nu_i \right) = \left(y_2^{t''} g_2^{s''}, y_2^{s''} \prod_{i=1}^{\delta} h_i^{y_{\alpha_i}} \right) \enspace , \tag{10.1}$$

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 ciphertexts. 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 ciphertexts. The Cramer-Shoup signature scheme uses an RSA-modulus $\mathbf{N}$ and elements from the subgroup $\mathrm{SQ}_{\mathbf{N}}$ of squares in $\mathbb{Z}_{\mathbf{N}}^*$, and it is parameterized by two collision-free hash functions. We refer the reader to Section 10.2 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 $\mathrm{SQ}_{\mathbf{N}}$. To move the hash value up in the exponent and to change group from G_{q_1} to $\mathrm{SQ}_{\mathbf{N}}$, the signer forms two commitments

$$\left(y_1^{t'} g_1^{s'}, y_1^{s'} g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})} \right) \quad \text{and} \quad \mathbf{y}^t \mathbf{g}^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})} \enspace .$$

Then it gives a cut-and-choose proof that the exponent in the left commitment equals the value committed to in the product (10.1). 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 $\mathrm{SQ}_{\mathbf{N}}$ of the hash value of the public keys it used to form the chain of ciphertexts.

The second hash function used in the Cramer-Shoup signature scheme is applied to a single element in $\mathrm{SQ}_{\mathbf{N}}$. Since H^{CHP} is not collision-free on such inputs, we use the Shamir hash function defined by $H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})} : \mathbb{Z} \to \mathrm{SQ}_{\mathbf{N}}$, $x \mapsto \mathbf{g}^x \bmod \mathbf{N}$ instead. A more detailed account of this function is given in Section 10.2. 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 ciphertexts.

10.2 Building Blocks

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

A Variation of the Cramer-Shoup Signature Scheme

Cramer and Shoup [40] introduce a signature scheme based on the strong RSA-assumption. We describe a slightly modified scheme. The Cramer-Shoup signature scheme $\mathcal{SS}^{\mathsf{cs}}_{\mathcal{F}_1,\mathcal{F}_2} = (\mathsf{SSKg}^{\mathsf{cs}}_{\mathcal{F}_1,\mathcal{F}_2}, \mathsf{Sig}^{\mathsf{cs}}, \mathsf{Vf}^{\mathsf{cs}})$ is defined as follows, where $\mathcal{F}_1, \mathcal{F}_2$ are collision-free families of functions.

On input 1^κ the key generation algorithm $\mathsf{SSKg}^{\mathsf{cs}}_{\mathcal{F}_1,\mathcal{F}_1}$ 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 $\mathbf{N} \leftarrow \mathbf{pq}$, $\mathbf{p}' \leftarrow (\mathbf{p}-1)/2$, and $\mathbf{q}' \leftarrow (\mathbf{q}-1)/2$. Then it chooses $\mathbf{h}, \mathbf{z} \in \mathrm{SQ}_{\mathbf{N}}$ and a $(\kappa+1)$-bit prime e' such that $e' = 1 \bmod 4$ randomly. Finally, it draws $f_1 \leftarrow_R F^{(1)}_\kappa$, $f_2 \leftarrow_R F^{(2)}_\kappa$, where $\mathcal{F}_1 = (F^{(1)}_i)^\infty_{i=1}$, $\mathcal{F}_2 = (F^{(2)}_i)^\infty_{i=1}$. Then it outputs $((f_1, f_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e'), (f_1, f_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}'))$. We also assume that it on input $(1^\kappa, f_1)$ uses f_1 instead of generating it.

The signature algorithm $\mathsf{Sig}^{\mathsf{cs}}$ takes as input a message m and a private key $(f_1, f_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}')$ 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 $\boldsymbol{\sigma}' \in \mathrm{SQ}_{\mathbf{N}}$ and computes

$$\mathbf{z}' = (\boldsymbol{\sigma}')^{e'} \mathbf{h}^{-f_1(m)} \ , \ \text{and} \quad \boldsymbol{\sigma} = \left(\mathbf{z}\mathbf{h}^{f_2(\mathbf{z}')}\right)^{1/e} \quad .$$

The verification algorithm $\mathsf{Vf}^{\mathsf{cs}}$ takes as input a message m, candidate signature $(e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')$, and $(f_1, f_2, \mathbf{N}, \mathbf{h}, \mathbf{z}, e')$ 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 $\mathbf{z}' \leftarrow \boldsymbol{\sigma}')^{e'} \mathbf{h}^{-f_1(m)}$ and verifies that $\mathbf{z} = \boldsymbol{\sigma}^e \mathbf{h}^{-f_2(\mathbf{z}')}$. 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 $\mathcal{F}_1$ and $\mathcal{F}_2$ 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 10.2.1. *The signature scheme $\mathcal{SS}^{\mathsf{cs}}_{\mathcal{F}_1,\mathcal{F}_2}$ is CMA-secure under the strong RSA-assumption and assuming that $\mathcal{F}_1$ and $\mathcal{F}_2$ are collision free families of functions.*

Proof. We assume familiarity with [40]. When the length of the exponent is between $\kappa + 1$ bits and $\frac{3}{2}\kappa - 4$ bits, the proof of [40] 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 $\mathbf{g} \in \mathrm{SQ}_{\mathbf{N}}$ 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 [40] 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 [40] that

$$\boldsymbol{\sigma}^e = \mathbf{z}\mathbf{h}^{f_2(\mathbf{z}')} \ . \tag{10.2}$$

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 $(\boldsymbol{\sigma}, e, 1, f_2(\mathbf{z}'))$ that satisfies Equation (10.2). We conclude from Lemma 10.2.11 that there exists no Type III adversary. □

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 10.2.2 (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

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

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. [74], [75]). Young and Yung [88] 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 10.2.3 (Assumption on the Distribution of Primes)**.** *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.

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.

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 [71].

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 [13, 59].

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 10.2.4 (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 $A \in \mathrm{PT}^*$ 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 $A \in \mathrm{PT}^*$ 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 $\mathcal{R}_{\mathrm{DL}}$ 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 10.2.5 (Non-Trivial Representation)**.** *Suppose that the DL-assumption holds, and let q_i be defined as in Definition 10.2.4. Choose $g_{i,1}, \ldots, g_{i,N} \in G_{q_i}$ randomly. Then for all $A \in \mathrm{PT}^*$, $\Pr[A(q_i, g_{i,1}, \ldots, g_{i,N}) = (\eta_1, \ldots, \eta_N) \neq 0 \wedge \prod_{j=1}^N g_{i,j}^{\eta_j} = 1]$ 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 $g_j \leftarrow h$, and chooses $e_l \in \mathbb{Z}_q$ randomly and sets $g_l \leftarrow g^{e_l}$ for $l \neq j$. Then it computes $(\eta_l)_{l=1}^N \leftarrow A(G_q, g, (g_l)_{l=1}^N)$ 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. □

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 [35]. 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 [35], but it is also an immediate consequence of Lemma 10.2.5 above.

Proposition 10.2.6. *Let k be a constant and define $\mathsf{CHPg}_{k,i}$ to be the algorithm which takes $(1^\kappa, \delta)$ as input, computes $(q_0, \ldots, q_{k-1}) \leftarrow \mathsf{CunnGen}_k(1^\kappa)$, 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 $(h_1, \ldots, h_\delta)$.

The Decision Diffie-Hellman Assumption

The Decision Diffie-Hellman (DDH) assumption introduced in Section 2.3 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. The DDH-assumption is equivalent to the security of the ElGamal encryption scheme, 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. We also use a variant of the DDH-problem captured below.

Lemma 10.2.7 (Variant DDH-Assumption)**.** *Suppose that the DDH-assumption is true, and let q_i be defined as in Definition 2.3.1. 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 $A \in \mathrm{PT}^*$ 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 $A \in \mathrm{PT}^*$, a constant $c > 0$, and an infinite index set $\mathcal{N}$ such that

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

This implies that one of the following inequalities hold

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

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. □

Generalized Semantic Security

Recall semantic security for asymmetric encryption schemes from Section 2.4 The following generalization is standard. Denote by $\mathbf{Exp}_{\mathcal{CS},A}^{\mu_1-\mu_2-\mathsf{ind}-b}(\kappa)$ 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 $c \leftarrow (E_{pk_i}(m_{b,i,j}))_{i=1,j=1}^{\mu_1,\mu_2}$ instead of a single ciphertext $E_{pk}(m_b)$. The following lemma follows by a straightforward hybrid argument.

Lemma 10.2.8. *If $\mathcal{CS}$ is polynomially indistinguishable, then for all adversaries $A \in \mathrm{PT}^*$ the absolute value $|\Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mu_1-\mu_2-\mathsf{ind}-0}(\kappa) = 1] - \Pr[\mathbf{Exp}_{\mathcal{CS},A}^{\mu_1-\mu_2-\mathsf{ind}-1}(\kappa) = 1]|$ is negligible in κ.*

The ElGamal Encryption Scheme

The ElGamal [42] public key encryption scheme 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 $\mathsf{Kg}^{\mathsf{elg}}$ 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 $E^{\mathsf{elg}}_{(g,y)}(m, r) = (g^r, y^r m)$, or $E^{\mathsf{elg}}_y(m, r) = (g^r, y^r m)$ when g is fixed. Sometimes we also write $E^{\mathsf{elg}}_{(g,y)}(m)$, when we do not care about the random input. To decrypt a ciphertext (u, v) using the private key x the decryption algorithm outputs vu^{-x}. We denote this $D^{\mathsf{elg}}_x(u, v) = vu^{-x}$.

Note that the plaintext 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 [63]. 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 ElGamal encryption scheme 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 ElGamal encryption scheme is that it is homomorphic. This means that if $(u_0, v_0) = E^{\mathsf{elg}}_{(g,y)}(m_0, r_0)$ and $(u_1, v_1) = E^{\mathsf{elg}}_{(g,y)}(m_1, r_1)$ then $(u_0 u_1, v_0 v_1) = E^{\mathsf{elg}}_{(g,y)}(m_0 m_1, r_0 + r_1)$. This is a straightforward consequence of the definition, but it implies that a ciphertext (u, v) can be re-encrypted by computing (ug^r, vy^r). All ElGamal 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 [85].

Proposition 10.2.9. *Let k be a constant and define $\mathsf{Kg}^{\mathsf{elg}}_{k,i}$ 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 $\mathsf{Kg}^{\mathsf{elg}}(q_i, g_i)$. Then the ElGamal encryption scheme $(\mathsf{Kg}^{\mathsf{elg}}_{k,i}, E^{\mathsf{elg}}, D^{\mathsf{elg}})$ is polynomially indistinguishable under the DDH-assumption.*

The Cramer-Shoup Encryption Scheme

Cramer and Shoup [39] introduced the first practical encryption scheme which is CCA2-secure under standard assumptions. The encryption scheme $\mathcal{CCA}_H =$

($\mathsf{Kg}^{\mathsf{CCA}}$, E^{CCA}, D^{CCA}), is defined as follows.

Let $\mathcal{F} = (F_i)_{i=1}^{\infty}$ be a collision-free family of functions. Assume that $D_i \supset G_q \times G_q$ and that $f_i(D_i) \subset \mathbb{Z}_q$ for all $i \in \{0,1\}^{\log_2 q}$. An example of this is the Chaum-van Heyst-Pfitzmann function in Section 10.2 with suitable modified security parameter.

The key generation algorithm $\mathsf{Kg}_{\mathcal{F}}^{\mathsf{CCA}}$ 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 $f \leftarrow_R F_\kappa$ and outputs $((f, q, g_1, g_2, c, d, h), (f, q, x_1, x_2, y_1, y_2, z))$. Encryption of a message $m \in G_q$ using the public key $Y = (f, q, g_1, g_2, c, d, h)$ and randomness $r \in \mathbb{Z}_q$ is given by

$$E_Y^{\mathsf{CCA}}(m, r) = (u, \mu, v, \nu) = (g_1^r, g_2^r, h^r m, c^r d^{r f(u,\mu,v)}) \enspace .$$

Note that (u, v) is an ElGamal ciphertext of the message m using the ElGamal public key (g_1, h), so decryption of a ciphertext (u, μ, v, ν) using the private key $X = (i, q, x_1, x_2, y_1, y_2, z)$ is given by $D_X^{\mathsf{CCA}}(u, \mu, v, \nu) = D_z^{\mathsf{elg}}(u, v) = m$ for valid ciphertexts. A ciphertext is considered valid if the predicate

$$T_X^{\mathsf{CCA}}(u, \mu, v, \nu) = (u^{x_1 + x_2 f(u,\mu,v)} \mu^{y_1 + y_2 f(u,\mu,v)} = \nu)$$

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

Theorem 10.2.10. *Let k be a constant and define $\mathsf{Kg}_{k,i,\mathcal{CF}}^{\mathsf{CCA}}$ to be the algorithm that on input 1^κ computes $(q_0, \ldots, q_{k-1}) \leftarrow \mathsf{CunnGen}_k(1^\kappa)$ and outputs $\mathsf{Kg}_{\mathcal{CF}}^{\mathsf{CCA}}(q_i)$. The Cramer-Shoup encryption scheme $(\mathsf{Kg}_{k,i,\mathcal{CF}}^{\mathsf{CCA}}, E^{\mathsf{CCA}}, D^{\mathsf{CCA}})$ is CCA2-secure under the DDH-assumption if $\mathcal{CF}$ is collision-free.*

A proof is given in [39]. Note that Proposition 10.2.6, and the obvious fact that the first statement of the DL-assumption is true if the DDH-assumption is true, imply that the encryption scheme is secure under the DDH-assumption if we instantiate the collision-free hash function $\mathcal{CF}$ with the Chaum-van Heijst-Pfitzmann hash function $\mathcal{CHP}$.

The Strong RSA-Assumption

To simplify some of the proofs in the thesis we prove a useful lemma that gives alternative ways to view the strong RSA-assumption. The proof of the lemma below follows the proof in Damgård and Fujisaki [41], 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 10.2.11 (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* $\mathbf{g}, \mathbf{h} \in \mathrm{SQ}_{\mathbf{N}}$ *be random. Then for all adversaries* $A \in \mathrm{PT}^*$ *the probabilities*

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

are negligible in κ.

Before we prove the lemma we introduce a relation. We define $\mathcal{R}_{\mathrm{SRSA}}$ 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 10.2.11. 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 $A \in \mathrm{PT}^*$, a constant c, and an infinite index set $\mathcal{N}$ such that

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

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 $(\mathbf{b}, \eta_0, \eta_1, \eta_2) \leftarrow A(\mathbf{N}, \mathbf{g}, \mathbf{h})$ and $(f, a, b) \leftarrow \mathrm{extgcd}(\eta_0, \eta_1 + e\eta_2)$. 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 $\mathbf{p}' \leftarrow (\mathbf{p} - 1)/2$, $\mathbf{q}' \leftarrow (\mathbf{q} - 1)/2$, and $t \leftarrow \mathbf{p}'\mathbf{q}'$. 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 $t \leftarrow \mathbf{p}'(\mathbf{N}/(2\mathbf{p}' + 1) - 1)/2$ 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

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

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 $A \in \mathrm{PT}^*$, 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 $(\eta_1, \eta_2) \leftarrow A(\mathbf{N}, \mathbf{g}', \mathbf{h}')$. If $\eta_1 = \pm\eta_2$ it outputs $(\mathbf{g}, \eta_1 + 1)$. Otherwise it computes $(f, a, b) \leftarrow \mathrm{extgcd}(\eta_1, \eta_2)$ 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 $\mathrm{SQ}_{\mathbf{N}}$ 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. □

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.

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 $\mathrm{SQ}_\mathbf{N}$, 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 10.2.12. *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 $A \in \mathrm{PT}^*$ 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 10.2.13. 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.

10.3 The Algorithms of The Scheme

We are now ready to describe the details of our construction following the informal description above. We denote our scheme by $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$, and define algorithms HGKg, HGSig, HGVf, and HGOpen 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.

Key Generation

The key generation phase proceeds as follows. Each group manager is given an ElGamal 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 10.3.1 (Key Generation, $\mathsf{HGKg}(1^\kappa, T)$)**.**

1. Run $(q_0, \ldots, q_3) \leftarrow \mathsf{CunnGen}_4(1^\kappa)$ 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
$$H^{\mathsf{CHP}} = (h_1, \ldots, h_\delta) \leftarrow \mathsf{CHPg}(q_2, \delta)$$
to generate a collision-free Chaum-van Heijst-Pfitzmann hash function.
3. Run $(X, Y) \leftarrow \mathsf{Kg}^{\mathsf{CCA}}(q_3)$ to generate keys for a Cramer-Shoup encryption scheme over G_{q_3}.
4. Run $((H^{\mathsf{CHP}}, \mathbf{g}, \mathbf{N}, \mathbf{h}, \mathbf{z}, e'), (H^{\mathsf{CHP}}, \mathbf{g}, \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{p}', \mathbf{q}')) \leftarrow \mathsf{SSKg}^{\mathsf{cs}}_{\mathsf{CHPg},\mathsf{Shg}}(1^\kappa)$ and choose $\mathbf{y} \in \mathrm{SQ}_{\mathbf{N}}$ 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 $(\mathit{spk}, \mathit{ssk})$ 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 10.2 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
$$(\mathit{hpk}(\beta), \mathit{hsk}(\beta)) \leftarrow (y_\beta, x_\beta) = \mathsf{Kg}^{\mathsf{elg}}(q_3, g_3)$$
for an ElGamal encryption scheme 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 $\alpha_0 = \omega$ and $\alpha_\delta = \alpha$, and compute
$$(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha) \leftarrow \mathsf{Sig}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, \mathit{ssk}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta}) \enspace .$$
Then redefine $\mathit{hsk}(\alpha) \leftarrow (e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$.
8. Let ω be the root of T. Redefine the public key $\mathit{hpk}(\omega)$ of the root ω to be
$$(\mathit{hpk}(\omega), \mathbf{N}, \mathbf{h}, \mathbf{z}, e', \mathbf{g}, \mathbf{y}, q_0, g_1, y_1, g_2, y_2, g_3, y_3, H^{\mathsf{CHP}}, Y, g_{\mathbf{N}}, y_{\mathbf{N}}, \kappa_c, \kappa_r)$$
and output $(\mathit{hpk}, \mathit{hsk})$.

Remark 10.3.1. 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.

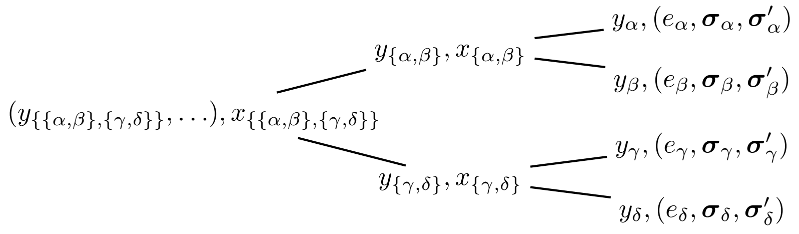

$(y_{\{\{\alpha,\beta\},\{\gamma,\delta\}\}},\ldots), x_{\{\{\alpha,\beta\},\{\gamma,\delta\}\}}$
$y_{\{\alpha,\beta\}}, x_{\{\alpha,\beta\}}$
$y_{\{\gamma,\delta\}}, x_{\{\gamma,\delta\}}$
$y_\alpha, (e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$
$y_\beta, (e_\beta, \boldsymbol{\sigma}_\beta, \boldsymbol{\sigma}'_\beta)$
$y_\gamma, (e_\gamma, \boldsymbol{\sigma}_\gamma, \boldsymbol{\sigma}'_\gamma)$
$y_\delta, (e_\delta, \boldsymbol{\sigma}_\delta, \boldsymbol{\sigma}'_\delta)$

Figure 10.1: An illustration of the output of HGKg for a three-level tree. The common group parameters, i.e., key size, generators etc., are not explicit.

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 $L_{\mathcal{R}_{\mathrm{HGS}}}$ 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 $G_{q_3}^{4\delta} \times G_{q_3}^{4} \times \mathrm{SQ}_{\mathbf{N}}^5$ 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

$$\begin{aligned}
&\gamma_0 = y_{\alpha_0} \ , \\
&(u_l, v_l, u'_l, v'_l) = (E^{\mathsf{elg}}_{(\gamma_l, g)}(\gamma_{l+1}, \tau_l), E^{\mathsf{elg}}_{(\gamma_l, g)}(1, \tau'_l)) \text{ for } l = 0, \ldots, \delta-1 \ , \\
&C_\delta = E^{\mathsf{CCA}}_{Y}(\gamma_\delta, \tau_\delta) \ , \\
&\mathbf{u} = \mathbf{y}^{\boldsymbol{\zeta}}\mathbf{g}^{\boldsymbol{\tau}} \ , \quad \mathbf{u}' = \mathbf{y}^{\boldsymbol{\zeta}'}\mathbf{g}^{\boldsymbol{\tau}'} \ , \quad \mathbf{C} = \mathbf{y}^{\boldsymbol{\psi}}\mathbf{g}^{\varepsilon} \ , \quad \text{and} \\
&\mathsf{Vf}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}, spk}((\gamma_1, \ldots, \gamma_\delta), (\varepsilon, \mathbf{v}/\mathbf{y}^{\boldsymbol{\tau}}, \mathbf{v}'/\mathbf{y}^{\boldsymbol{\tau}'})) = 1 \ .
\end{aligned}$$

In Section 10.5 we construct a zero-knowledge proof of knowledge denoted by $\pi_{\mathrm{hgs}} = (P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$ for this relation.

Algorithm 10.3.2 (Signing, $\mathsf{HGSig}(m, T, hpk, hsk(\alpha))$)**.** Let $\alpha_0, \ldots, \alpha_\delta$ with $\omega = \alpha_0$ 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 $(u_l,\ v_l,\ u'_l,\ v'_l) \leftarrow (E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l),\ E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(1, r'_l))$, for $l = 0, \ldots, \delta - 1$, and $C_\delta = E^{\mathsf{CCA}}_Y(y_{\alpha_\delta}, r_\delta)$. This is the list of ciphertexts.
2. Choose $r, s, r', s', t \leftarrow_R [0, 2^{\kappa_r}\mathbf{N}-1]$ and set $(\mathbf{u}, \mathbf{v}) \leftarrow (\mathbf{y}^s\mathbf{g}^r, \mathbf{y}^r\boldsymbol{\sigma}_\alpha)$, $(\mathbf{u}', \mathbf{v}') \leftarrow (\mathbf{y}^{s'}\mathbf{g}^{r'}, \mathbf{y}^{r'}\boldsymbol{\sigma}'_\alpha)$, and $\mathbf{C} \leftarrow \mathbf{y}^t\mathbf{g}^{e_\alpha}$. This is a commitment of the signature $(e_\alpha, \boldsymbol{\sigma}_\alpha, \boldsymbol{\sigma}'_\alpha)$.
3. Compute a non-interactive proof
$$\pi \leftarrow P^{\mathsf{O}(m,\cdot)}_{\text{hgs}}\Big(\big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}\big), \big((\tau_0, \ldots, \tau_\delta), (\boldsymbol{\tau}, \boldsymbol{\zeta}, \boldsymbol{\tau}', \boldsymbol{\zeta}', \boldsymbol{\psi}, \varepsilon)\big)\Big)$$
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 10.3.2. Note that we switch the order of the components in the ElGamal encryption scheme in order to simplify the construction of the proof of knowledge. For example, $D^{\mathsf{elg}}_{1/x_\omega}(u_0, v_0) = y_{\alpha_1}$.

The construction of the proof of knowledge π_{hgs} is involved and postponed until Section 10.5. The verification algorithm consists simply of verifying the proof of knowledge contained in a signature.

Algorithm 10.3.3 (Verification, $\mathsf{HGVf}(T, hpk, m, \sigma)$)**.** 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 $V^{\mathsf{O}(m,\cdot)}_{\text{hgs}}(c, \pi)$.

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 10.3.4 (Open, $\mathsf{HGOpen}(T, hpk, hsk(\beta), m, \sigma)$)**.** 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 $\mathsf{HGVf}(T, hpk, m, \sigma) = \perp$ or if $u'_l \neq (v'_l)^{x_\beta}$ and $\beta \neq \omega$, then return $\perp$. Otherwise compute $y_\alpha \leftarrow D^{\mathsf{elg}}_{1/x_\beta}(u_l, v_l)$ and return α.

Remark 10.3.3. To ensure that the protocol satisfies the new and stronger definition instead of the one presented in [84], 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.

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 10.6.

10.4 Proof of Security

We analyze the security of the scheme and prove the following theorem.

Theorem 10.4.1. *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 [40] 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 $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$ using A as a blackbox.

Denote by H^b_{o} the machine that is identical to H^b except that the open oracle $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ is simulated as follows. Consider a query on the form (α, m, σ), where α is on depth l. If $\mathsf{HGVf}(T, hpk, m, \sigma) = 0$, 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_{o}^b computes $D_X^{\mathsf{CCA}}(C_\delta)$ and if the result does not equal y_{α_δ} for some $\alpha_\delta \in \mathcal{L}(T)$, the $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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 $|\Pr[H^b = 1] - \Pr[H_{\mathrm{o}}^b = 1]|$ 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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ is answered incorrectly.

Let $p(\kappa)$ denote the running time of A. Then it follows that A asks the simulated $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$ oracle at most $p(\kappa)$ queries. Denote by T_l the machine that simulates H_{o}^b until $l-1$ queries have been answered by the simulated $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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

$$\sigma = ((u_l, v_l, u_l', v_l')_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, (\gamma, c, e)) \enspace .$$

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 $L_{\mathcal{R}_{\mathrm{HGS}}}$, 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 $(E_{y_{\alpha_0}}^{\mathsf{elg}}(y_{\alpha_1}), E_{y_{\alpha_0}}^{\mathsf{elg}}(1), \ldots, E_{y_{\alpha_{\delta-1}}}^{\mathsf{elg}}(y_{\alpha_\delta}), E_{y_{\alpha_{\delta-1}}}^{\mathsf{elg}}(1)), E_Y^{\mathsf{CCA}}(y_{\alpha_\delta})$ for any path $\alpha_0, \ldots, \alpha_\delta$ from the root $\omega = \alpha_0$ in T to a leaf $\alpha = \alpha_\delta \in \mathcal{L}(T)$.

Note that if the tuple above does belong to $L_{\mathcal{R}_{\mathrm{HGS}}}$, then $(u_l', v_l') = E_{(y_{\alpha_l}, g)}^{\mathsf{elg}}(1, r_l')$ 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 $\mathbf{k} \in \mathrm{SQ}_{\mathbf{N}}$ 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 [40] 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

$$\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^*_{\mathrm{hgs}}}^{V_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^*_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}) \notin L_{\mathcal{R}_{\mathrm{HGS}}}]$$

is non-negligible. This contradicts the soundness of the protocol π_{hgs}. Formally, the contradiction follows from combining Proposition 10.5.21 with Proposition 8.5.5.

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 10.5 of the next section, 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 10.5 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 $P^{\emptyset}_{\mathrm{hgs}}$ 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 $\mathbf{k} \in \mathrm{SQ}_{\mathbf{N}}$ 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 $\kappa \in \mathcal{N}$

$$\begin{aligned}
\kappa^{-c_1} &\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1] \\
&\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \mid \delta_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\qquad \cdot \Pr[\delta_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\quad + \frac{1}{2}\kappa^{-c_1} \quad .
\end{aligned}$$

Thus, $\Pr[\delta_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq \frac{1}{2}\kappa^{-c_1}$ and from Proposition 10.5.22 we conclude that there exists an extractor $\mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}$ and a polynomial $p(\kappa)$ such that

$$\Pr[(I_{P^{\emptyset}_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}), \mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}})) \in \mathcal{R}'_{\mathrm{HGS}} \mid \delta_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq 1 - \varepsilon(\kappa)$$

for some negligible function $\varepsilon(\kappa)$, and such that the expected running time of $\mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}$ on inputs $(\Lambda', r_{\mathrm{p}})$ such that $\delta_{P^{\emptyset}_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}$ 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 $r_{\mathrm{p}} \in \{0,1\}^*$ randomly and simulates $\mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}$ on these inputs except that $P^{\emptyset}_{\mathrm{hgs}}$ uses the value of $\mathbf{k}$ instead of generating it. Furthermore, $\mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}$ 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 $I_{P^{\emptyset}_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}})$ as

$$\big((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}\big) \in G_{q_3}^{4\delta} \times G_{q_3}^{4} \times \mathrm{SQ}_{\mathbf{N}}^{5}$$

and it interprets the output of $\mathcal{X}^{P^{\emptyset}_{\mathrm{hgs}}}$ 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 $(\varepsilon, \mathbf{v}/\mathbf{y}^{\tau}, \mathbf{v}'/\mathbf{y}^{\tau'})$. 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

$$\frac{1}{2\kappa^{c_1}}\left(\frac{1}{2\kappa^{c_1}} - \frac{1}{4\kappa^{c_1}} - \varepsilon(\kappa)\right)$$

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 [40] that this contradicts the strong RSA-assumption, or the collision-freeness of $\mathcal{SH}$ or $\mathcal{CHP}$. From Proposition 10.2.12 we know that finding a collision in $\mathcal{SH}$ contradicts the strong RSA-assumption, and from Proposition 10.2.6 we know that finding a collision in $\mathcal{CHP}$ contradicts the DL-assumption. □

Denote by $H^b_{\mathrm{o,g}}$ 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 $H^b_{\mathrm{o,g}}$ 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. $\Pr[H^b_{\mathrm{o}} = 1] = \Pr[H^b_{\mathrm{o,g}} = 1]/|\mathcal{L}(\kappa)|^2$.

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 $y_{\beta^{(b)}_l}$, 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

$$y_{\beta^{(b)}_l} = D_{1,l} \enspace .$$

Note that the simulated hierarchical group signature of m is only computed if $(\alpha^{(0)}, \alpha^{(1)}) = (\beta^{(0)}_\delta, \beta^{(1)}_\delta)$. The single query m to the $\mathsf{HGSig}(T, hpk, hsk(\alpha^{(b)}), \cdot)$ oracle is simulated as follows. To simplify the exposition we write α_l instead of $\alpha^{(b)}_l$ as in Experiment 8.2.1. 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

$$\begin{aligned}
&(u_l, v_l, u'_l, v'_l) = (D_{2,l}, y_{\alpha_{l+1}} D_{3,l}, D'_{2,l}, D'_{3,l}), \quad \text{for } l = 0, \ldots, \delta - 1 \enspace , \\
&C_\delta = E^{\mathsf{CCA}}_Y(y_{\alpha_\delta}, r) \enspace , \quad \text{and} \\
&(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^{\boldsymbol{\zeta}}, \mathbf{g}^{\boldsymbol{\tau}}), \quad (\mathbf{u}', \mathbf{v}') = (\mathbf{g}^{\boldsymbol{\zeta}'}, \mathbf{g}^{\boldsymbol{\tau}'}), \quad \mathbf{C} = \mathbf{g}^{\boldsymbol{\psi}} \enspace .
\end{aligned}$$

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 10.5.20. 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 10.2. 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 10.5.20, 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 10.2.7 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 10.2.7, 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 encryption scheme is broken by the adversary A' taking part in Experiment 2.4.9 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 encryption scheme. Instead of computing $D^{\mathsf{CCA}}_X(C_\delta)$ to simulate the answer to a query to the $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-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 ciphertext C_δ. It then uses this challenge ciphertext to construct the simulated hierarchical group signature.

It follows that $\mathbf{Exp}^{\mathsf{cca2}-b}_{\mathcal{CCA}_H,A'}(\kappa)$ is identically distributed to $H^b_{\mathrm{o,g,nddh}}$, and the CCA2-security of the Cramer-Shoup encryption scheme is broken. This contradicts Proposition 10.2.10 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 $\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$. Denote by H_{p} the machine that is identical to H except that it simulates the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-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 10.5.20 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 10.5.20 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 $\mathbf{k} \in \mathrm{SQ}_{\mathbf{N}}$ 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 $\omega = \alpha_0$ 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

$$V_{\mathrm{hgs}}((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}, \gamma, c, d) = 1 \text{ and still}$$
$$((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \notin L_{\mathcal{R}_{\mathrm{HGS}}}$$

is non-negligible. This implies that

$$\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}^{V_{\mathrm{hgs}}}_{P^{\beta}_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \wedge I_{P^{\beta}_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \notin L_{\mathcal{R}_{\mathrm{HGS}}}]$$

is non-negligible. This contradicts the soundness of the protocol π_{hgs}. Formally, the contradiction follows from combining Proposition 10.5.21 with Proposition 8.5.5.

Consider a σ such that $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \in L_{\mathcal{R}_{\mathrm{HGS}}}$. Then we have $(u_l, v_l) = E^{\mathsf{elg}}_{(\gamma_l, g)}(\gamma_{l+1}, r_l)$ and $(u'_l, v'_l) = E^{\mathsf{elg}}_{(\gamma_l, g)}(1, r'_l)$ 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 $((u_l, v_l, u'_l, v'_l)_{l=0}^{\delta-1}, C_\delta, (\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}'), \mathbf{C}) \in L_{\mathcal{R}_{\mathrm{HGS}}}$ and

$\mathsf{HGVf}(T, hpk, m, \sigma) = 1$ 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 $\Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1]$ 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 $\kappa \in \mathcal{N}$

$$\begin{aligned}
\kappa^{-c_1} &\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1] \\
&\leq \Pr[\mathrm{Acc}_{V_{\mathrm{hgs}}}(\mathrm{view}_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}, r_{\mathrm{v}})) = 1 \mid \delta_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\qquad \cdot \Pr[\delta_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \\
&\quad + \frac{1}{2}\kappa^{-c_1} \ .
\end{aligned}$$

Thus, $\Pr[\delta_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda, r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq \frac{1}{2}\kappa^{-c_1}$ and from Proposition 10.5.22 we conclude that there exists an extractor $\mathcal{X}^{P^\beta_{\mathrm{hgs}}}$ and a polynomial $t(\kappa)$ such that

$$\Pr[(I_{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}), \mathcal{X}^{P^\beta_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}})) \in \mathcal{R}'_{\mathrm{HGS}} \mid \delta_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}] \geq 1 - \varepsilon(\kappa)$$

for some negligible function $\varepsilon(\kappa)$, and such that the expected running time of $\mathcal{X}^{P^\beta_{\mathrm{hgs}}}$ on inputs $(\Lambda', r_{\mathrm{p}})$ such that $\delta_{P^\beta_{\mathrm{hgs}}}^{V'_{\mathrm{hgs}}}(\Lambda', r_{\mathrm{p}}) \geq \frac{1}{2}\kappa^{-c_1}$ 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

$$\frac{1}{2\kappa^{c_1}}\left(\frac{1}{2\kappa^{c_1}} - \frac{1}{4\kappa^{c_1}} - \varepsilon(\kappa)\right)$$

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 10.2 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 10.2.12 we know that the first event contradicts the strong RSA-assumption, and from Proposition 10.2.6 we know that the second event contradicts the DL-assumption. Thus, the claim holds. $\square$

Conclusion of Proof. To conclude the proof it suffices to note that we have proved that both $\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T)$ and $\mathbf{Adv}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$ are negligible. $\square$

Remark 10.4.2. It is shown in [9] that it is necessary to use a CCA2-secure encryption scheme to form a group signature scheme. Still we only use a CCA2-secure encryption scheme 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 $\mathsf{HGOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle for intermediate levels of the tree can be answered using a single query to the decryption oracle for the CCA2-secure Cramer-Shoup encryption scheme used to encrypt leaves.

Remark 10.4.3. The exposition here differs from the exposition in [84]. 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 [84] 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.

10.5 Construction of the Proof of Knowledge

In this section we describe the proof of knowledge needed in Chapter 10. 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.

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 10.1.

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 $\mathrm{SQ}_{\mathbf{N}}$, then the random exponent should be chosen in $[0, 2^{\kappa+\kappa_r} - 1]$ to achieve a statistically hiding commitment 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 $\mathbf{z} \in \mathrm{SQ}_{\mathbf{N}}$ 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 10.5.1. The exposition here differs from the exposition in the preliminary full version of [84] in one important aspect. In [84] 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.

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 $\mathrm{SQ}_{\mathbf{N}}$ or $G_{\mathbf{N}}$ it is given $\mathbf{N}$ as input. We do not state this explicitly to avoid cluttering the exposition.

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 $\mathcal{R}_{\mathrm{HGS}}$ 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 ciphertexts is chained properly.

Protocol 10.5.1 (Chained Ciphertexts)**.**
COMMON INPUT: $y_0, g, y \in G_q$ and $\left((u_l, v_l, u_l', v_l'), (\mu_l, \nu_l)\right)_{l=0}^{\delta-1} \in G_q^{6\delta}$
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 $(u_l, v_l, u_l', v_l') = (E^{\mathsf{elg}}_{(y_l,g)}(y_{l+1}, r_l), E^{\mathsf{elg}}_{(y_l,g)}(1, r_l')) = (y_l^{r_l}, g^{r_l}y_{l+1}, y_l^{r_l'}, g^{r_l'})$ 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
$$A_{1,l} \leftarrow y^{a_l'}g^{a_l}\mu_l^{r_{l+1}} \ , \ A_{2,l} \leftarrow y^{a_l}\nu_l^{r_{l+1}} \ , \ \text{and} \quad A_{3,l} \leftarrow y^{a_l''}g^{r_{l+1}}$$
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 $B_0 \leftarrow y_0^{e_0}$,
$$B_{1,l} \leftarrow g^{e_l}y^{i_l} \quad \text{and} \quad B_{2,l} \leftarrow g^{i_l}y^{j_l}$$

for $l = 0, \ldots, \delta - 1$, and

$$\begin{aligned} B_{3,l} &\leftarrow y^{b'_l} g^{b_l} \mu_l^{e_{l+1}} \ , & B_{4,l} &\leftarrow y^{b_l} \nu_l^{e_{l+1}} \ , \\ B_{5,l} &\leftarrow y^{h_l} g^{f_l} \ , & B_{6,l} &\leftarrow y^{f_l} \ , \\ B_{7,l} &\leftarrow y^{b''_l} g^{e_{l+1}} \ , & B_{8,l} &\leftarrow (u'_{l+1})^{k_l} \ , \\ B_{9,l} &\leftarrow y^{w'_l} (v'_{l+1})^{k_l} \ , & B_{10,l} &\leftarrow g^{w_l} \end{aligned}$$

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

$$d_{1,l} \leftarrow cr_l + e_l \ , \quad d_{2,l} \leftarrow -cs_l + i_l \ , \text{ and } \quad d_{3,l} \leftarrow -ct_l + j_l \tag{10.3}$$

for $l = 0, \ldots, \delta - 1$ and

$$d_{4,l} \leftarrow ca_l + b_l \ , \qquad d_{5,l} \leftarrow ca'_l + b'_l \ , \tag{10.4}$$
$$d_{6,l} \leftarrow c\,(a_l + s_l r_{l+1}) + f_l \ , \qquad d_{7,l} \leftarrow ct_l r_{l+1} + h_l \ , \tag{10.5}$$
$$d_{8,l} \leftarrow ca''_l + b''_l \ , \qquad d_{9,l} \leftarrow ca''_l + w'_l \ , \tag{10.6}$$
$$d_{10,l} \leftarrow c(r_{l+1}/r'_l) + k_l \ , \qquad d_{11,l} \leftarrow cr'_l + w_l \tag{10.7}$$

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

$$u_0^c B_0 = y_0^{d_{1,0}} \tag{10.8}$$

and

$$(v_l/\nu_l)^c B_{1,l} = g^{d_{1,l}} y^{d_{2,l}} \text{ and } \qquad B_{2,l} = \mu_l^c y^{d_{3,l}} g^{d_{2,l}} \ , \tag{10.9}$$
$$\tag{10.10}$$

for $l = 0, \ldots, \delta - 1$ and

$$A_{1,l}^c B_{3,l} = y^{d_{5,l}} g^{d_{4,l}} \mu_l^{d_{1,l+1}} \ , \qquad A_{2,l}^c B_{4,l} = y^{d_{4,l}} \nu_l^{d_{1,l+1}} \ , \tag{10.11}$$
$$A_{1,l}^c B_{5,l} = g^{d_{6,l}} y^{d_{7,l}} \ , \qquad (A_{2,l}/u_{l+1})^c B_{6,l} = y^{d_{6,l}} \ , \tag{10.12}$$
$$A_{3,l}^c B_{7,l} = y^{d_{8,l}} g^{d_{1,l+1}} \ , \qquad u_l^c B_{8,l} = (u'_{l+1})^{d_{10,l}} \ , \tag{10.13}$$
$$A_{3,l}^c B_{9,l} = y^{d_{9,l}} (v'_{l+1})^{d_{10,l}} \ , \qquad (v'_{l+1})^c B_{10,l} = g^{d_{11,l}} \tag{10.14}$$

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 10.2.

$$\begin{tikzcd}
(u_0, v_0, u'_0, v'_0) \arrow[d, leftrightarrow] & (u_1, v_1, u'_1, v'_1) \arrow[d, leftrightarrow] & \cdots & (u_{\delta-1}, v_{\delta-1}, u'_{\delta-1}, v'_{\delta-1}) \arrow[d, leftrightarrow] \\
(\mu_0, \nu_0) \arrow[ur, leftrightarrow] & (\mu_1, \nu_1) \arrow[ur, leftrightarrow] & \cdots \arrow[ur, leftrightarrow] & (\mu_{\delta-1}, \nu_{\delta-1})
\end{tikzcd}$$

Figure 10.2: The protocol for a chain of ciphertexts. 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 10.5.2. *Protocol 10.5.1 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 (10.8)–(10.14), 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 (10.3)–(10.7) to extract ρ_l, ζ_l, and τ_l for $l = 0, \ldots, \delta - 1$ such that

$$\begin{aligned}
&u_0 = y_0^{\rho_0} \ , \\
&v_l/\nu_l = g^{\rho_l} y^{-\zeta_l} \quad \text{and} \quad \mu_l = y^{\tau_l} g^{\zeta_l}
\end{aligned}$$

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

$$\begin{aligned}
A_{1,l} &= y^{\alpha'_l} g^{\alpha_l} \mu_l^{\rho_{l+1}} \ , & A_{2,l} &= y^{\alpha_l} \nu_l^{\rho_{l+1}} \ , \\
A_{1,l} &= y^{\lambda_l} g^{\omega_l} \ , & A_{2,l}/u_{l+1} &= y^{\omega_l} \ , \\
A_{3,l} &= y^{\alpha''_l} g^{\rho_{l+1}} \ , & u_l &= (u'_{l+1})^{\rho_l^{\times}} \ , \\
A_{3,l} &= y^{\omega_l^{\times}} (v'_{l+1})^{\rho_l^{\times}} \ , & v'_{l+1} &= g^{\rho_l^{+}}
\end{aligned}$$

From this we can compute $\zeta_l^* \leftarrow (\omega_l - \alpha_l)/\rho_{l+1}$ and $\tau_l^* \leftarrow (\lambda_l - \alpha_l')/\rho_{l+1}$ 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., $(u_l, v_l) = E^{\mathsf{elg}}_{(\gamma_l, g)}(\gamma_{l+1}^*, \rho_l)$, 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 10.5 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 ρ_0, …, $\rho_{\delta-1}$, τ_0, …, $\tau_{\delta-1}$, ζ_0, …, $\zeta_{\delta-1}$, and γ_1, …, γ_δ 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 10.5 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 (10.8)–(10.14). 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 [82] studied a simpler problem, namely, given a ciphertext $E_{(g,y)}^{\mathsf{elg}}(m)$ with $g, y \in G_{q_3}$ and g_2^m, prove that an exponential relation holds between the plaintext 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 10.5.2 (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 10.5.3 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 10.5.3. *Protocol 10.5.2 is a* $\{0,1\}^{\kappa_c}$*-*Σ*-protocol.*

Proof. This follows directly from Lemma 10.5.4 below. □

We now give the double-decker exponentiation protocol called from within the protocol above.

Protocol 10.5.3 (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 $F_{1,l} \leftarrow y_2^{e_l'} g_2^{f_l'}$, $F_{2,l} \leftarrow y_2^{f_l'} h^{y_3^{f_l}}$, and $A_l \leftarrow y_3^{e_l} g_3^{f_l}$. 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 $d_{1,l} \leftarrow e_l - b_l t$, $d_{2,l} \leftarrow f_l - b_l s$, $d_{3,l} \leftarrow f_l' - b_l y_3^{d_{2,l}} s'$, and $d_{4,l} \leftarrow e_l' - b_l y_3^{d_{2,l}} t'$, 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

$$\theta^{b_l y_3^{d_{2,l}}} y_2^{d_{4,l}} g_2^{d_{3,l}} = F_{1,l} \ , \qquad y_2^{d_{3,l}} (\omega^{b_l} h^{(1-b_l)})^{y_3^{d_{2,l}}} = F_{2,l} \ , \text{ and} \tag{10.15}$$

$$\phi^{b_l} y_3^{d_{1,l}} g_3^{d_{2,l}} = A_l \ . \tag{10.16}$$

Lemma 10.5.4. *Protocol 10.5.3 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 (10.15)–(10.16). 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

$$\begin{aligned} F_{2,l} &= y_2^{d_{3,l}} \omega^{y_3^{d_{2,l}}} = y_2^{\psi' - y_3^{d_{2,l}} \zeta'} \omega^{y_3^{d_{2,l}}} = y_2^{\psi' - y_3^{\psi - \zeta} \zeta'} \omega^{y_3^{\psi - \zeta}} \quad \text{and} \\ F_{2,l} &= y_2^{d'_{3,l}} h^{y_3^{d'_{2,l}}} = y_2^{\psi'} h^{y_3^{d'_{2,l}}} = y_2^{\psi'} h^{y_3^{\psi}} \ . \end{aligned}$$

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

$$\begin{aligned} F_{1,l} &= \theta^{y_3^{d_{2,l}}} y_2^{d_{4,l}} g_2^{d_{3,l}} = \theta^{y_3^{d_{2,l}}} y_2^{\varepsilon' - y_3^{d_{2,l}} \tau'} g_2^{\psi' - y_3^{d_{2,l}} \zeta'} \quad \text{and} \\ F_{1,l} &= y_2^{d'_{4,l}} g_2^{d'_{3,l}} = y_2^{\varepsilon'} g_2^{\psi'} \ . \end{aligned}$$

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 (10.15)–(10.16). We conclude that the protocol is special honest verifier perfect zero-knowledge. □

Our next protocol shows that the plaintext of an ElGamal encryption is the value hidden in a commitment. Since the protocol is used in conjunction with Cramer-Shoup ciphertexts, we use a notation that is consistent with the notation we use for the Cramer-Shoup encryption scheme in the main protocol.

Protocol 10.5.4 (Equality of Committed and Encrypted Plaintexts).
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 $A_1 \leftarrow y_3^a g_3^e$, $A_2 \leftarrow y_3^e \bar{h}^f$, $A_3 \leftarrow \bar{g}_1^f$, 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 $d_1 \leftarrow ct + a$, $d_2 \leftarrow cs + e$, $d_3 \leftarrow -cr + f$ and hands (d_1, d_2, d_3) to the verifier.
4. The verifier checks that

$$\mu^c A_1 = y_3^{d_1} g_3^{d_2}, \quad (\nu/\bar{v})^c A_2 = y_3^{d_2} \bar{h}^{d_3}, \quad A_3/\bar{u}^c = \bar{g}_1^{d_3} \ . \tag{10.17}$$

Lemma 10.5.5. *Protocol 10.5.4 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 (10.17) 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 ciphertext 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 (10.17). 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 ciphertext is valid. Here H denotes the representation of a collision-free hash function.

Protocol 10.5.5 (Validity of Cramer-Shoup Ciphertext).
COMMON INPUT: $H : G_{q_3}^3 \rightarrow \mathbb{Z}_{q_3}$, $\bar{g}_1, \bar{g}_2, \bar{c}, \bar{d} \in G_{q_3}$, 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 $B_1 \leftarrow \bar{g}_1^a$, $B_2 \leftarrow \bar{g}_2^a$, $B_3 \leftarrow (\bar{c}\bar{d}^{H(\bar{u},\bar{\mu},\bar{v})})^a$ 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 $d \leftarrow cr + a$ 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 10.5.6. *Protocol 10.5.5 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 $\rho \leftarrow (d - d')/(c - c')$ 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 10.3.2 involving only elements from G_{q_1}, G_{q_2}, and G_{q_3}.

Protocol 10.5.6 (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

$$\begin{aligned}(u_l, v_l) &=\leftarrow E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\ (u'_l, v'_l) &= E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(1, r'_l) \text{ for } l = 0, \ldots, \delta - 1 \ , \\ C_\delta &= E^{\mathsf{CCA}}_Y(y_{\alpha_\delta}, r_\delta) \ , \quad \text{and} \\ (\mu'', \nu'') &= (y_1^{t''} g_1^{s''}, y_1^{s''} g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})}) \ .\end{aligned}$$

1. The prover chooses $s_l, t_l \in \mathbb{Z}_{q_2}$ randomly, computes commitments
$$(\mu_l, \nu_l) \leftarrow \left(y_3^{t_l} g_3^{s_l}, y_3^{s_l} y_{\alpha_{l+1}}\right)$$
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 $(\mu'_l, \nu'_l) \leftarrow (y_2^{t'_l} g_2^{s'_l}, y_2^{s'_l} h_{l+1}^{y_{\alpha_{l+1}}})$ for $l = 0, \ldots, \delta - 1$, and hands $(\mu'_l, \nu'_l)_{l=1}^{\delta}$ to the verifier.

3. The prover and verifier computes $(\mu', \nu') \leftarrow \left(\prod_{l=0}^{\delta-1} \mu'_l, \prod_{l=0}^{\delta-1} \nu'_l\right)$. The prover computes $s' \leftarrow \sum_{l=0}^{\delta-1} s'_l$ and $t' \leftarrow \sum_{l=0}^{\delta-1} t'_l$.

4. Invoke the following protocols in parallel:

   a) Protocol 10.5.1 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 10.5.2 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 10.5.2 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 10.5.4 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 10.5.5 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 10.5.7. *Protocol 10.5.6 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 8.5.1 and 8.5.2 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 10.5.2 we can find $\tau_l, \tau_l', \zeta_l, \psi_l \in \mathbb{Z}_{q_3}$, $\gamma_l \in G_{q_3}$ such that

$$\begin{aligned}(u_l, v_l) &= E^{\mathsf{elg}}_{(\gamma_l, g_3)}(\gamma_{l+1}, \tau_l) = (\gamma_l^{\tau_l}, g_3^{\tau_l}\gamma_{l+1}) \ ,\\ (u_l', v_l') &= E^{\mathsf{elg}}_{(\gamma_l, g_3)}(1, \tau_l') = (\gamma_l^{\tau_l'}, g_3^{\tau_l'}) \ , \ \text{and}\\ (\mu_l, \nu_l) &= (y_3^{\psi_l} g_3^{\zeta_l}, y_3^{\zeta_l}\gamma_{l+1})\end{aligned}$$

for $l = 0, \ldots, \delta - 1$. Using Lemma 10.5.3 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 10.5 is satisfied. Thus, we assume that $\gamma_\delta^* = \gamma_\delta$, $\psi_\delta^* = \psi_\delta$, and $\zeta_\delta^* = \zeta_\delta$.

Using Lemma 10.5.3 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 10.5 is satisfied. Thus, we assume that $\prod_{l=1}^{\delta} h_l^{\gamma_l} \leftarrow \Gamma$, $\psi' = \sum_{l=0}^{\delta-1} \psi_l'$, and $\zeta' = \sum_{l=0}^{\delta-1} \zeta_l'$.

Using Lemma 10.5.5 we can find $\psi_{\delta-1}^{\times}, \zeta_{\delta-1}^{\times}, \tau \in \mathbb{Z}_{q_3}$ and $\gamma_\delta^{\times} \in G_{q_3}$ such that

$$(\mu_{\delta-1}, \nu_{\delta-1}) = (y_3^{\psi_{\delta-1}^{\times}} g_3^{\zeta_{\delta-1}^{\times}}, y_3^{\zeta_{\delta-1}^{\times}}\gamma_\delta^{\times}) \quad \text{and} \quad (\bar{u}, \bar{v}) = (\bar{g}_1^{\tau}, \bar{h}^{\tau}\gamma_\delta^{\times}) \ .$$

If $\gamma_\delta^{\times} \neq \gamma_\delta$, then either $\psi_{\delta-1}^{\times} \neq \psi_{\delta-1}$ or $\zeta_{\delta-1}^{\times} \neq \zeta_{\delta-1}$. Then we define $\eta = (\psi_{\delta-1}^{\times} - \psi_{\delta-1})/(\zeta_{\delta-1} - \zeta_{\delta-1}^{\times})$ and conclude that $y_3 = g_3^{\eta}$. In other words Case 3 in Section 10.5 is satisfied. Thus, we assume that $\gamma_\delta^{\times} = \gamma_\delta$, $\psi_{\delta-1}^{\times} = \psi_{\delta-1}$ and $\zeta_{\delta-1}^{\times} = \zeta_{\delta-1}$.

Using Lemma 10.5.6 we can find $\tau \in \mathbb{Z}_{q_3}$ such that $(\bar{u}, \bar{\mu}, \bar{v}, \bar{\nu}) = E_Y^{\mathsf{CCA}}(\gamma_\delta, \tau)$, where Y is the public key $Y = (H, \bar{g}_1, \bar{g}_2, \bar{c}, \bar{d}, \bar{h})$ to the Cramer-Shoup encryption scheme 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. □

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 $\mathrm{SQ}_{\mathbf{N}}$ 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 [19] as well as by Camenisch and Michels [28]. We use Boudot's protocol [18] 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 [18] for details.

Protocol Head 10.5.7 (A Committed Number Lies in an Interval)**.**
Common Input: $\mathbf{g}, \mathbf{y}, \mathbf{C} \in \mathrm{SQ}_{\mathbf{N}}$ 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 10.5.8. *Protocol 10.5.7 is a* $\{0,1\}^{\kappa_c}$*-*Σ*-protocol.*

Proof. Boudot [18] essentially shows that either $e \in [a, b]$ or Case 5 in Section 10.5 is satisfied. □

We now give the proof of equality of exponents over distinct groups using the protocol above.

Protocol 10.5.8 (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

$$\mathbf{A} \leftarrow \mathbf{y}^b \mathbf{g}^a \quad \text{and} \quad A = y^{b'} g^a$$

   and hands $(\mathbf{A}, A)$ to the verifier.

2. Protocol 10.5.7 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

$$\begin{aligned} d_1 &\leftarrow ce + a \bmod 2^{\kappa_c+\kappa_r} n \ , & (10.18)\\ d_2 &\leftarrow cs + b \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \ \text{and} & (10.19)\\ d_3 &\leftarrow cs' + b' \bmod n \ , & (10.20) \end{aligned}$$

   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 10.5.9. *Protocol 10.5.8 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

$$\mathbf{y}^{d_2-d_2'}\mathbf{g}^{d_1-d_1'} = \mathbf{C}^{c-c'} \quad \text{and} \quad y^{d_3-d_3'}g^{d_1-d_1'} = C^{c-c'} \quad .$$

If $c-c'$ does not divide both $d_1 - d_1'$ and $d_2 - d_2'$ we define $\eta_0 \leftarrow c-c'$, $\eta_1 \leftarrow d_1 - d_1'$, $\eta_2 \leftarrow d_2 - d_2'$, and $\mathbf{b} \leftarrow \mathbf{C}$ and conclude that Case 5 in Section 10.5 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 10.5 is satisfied.

Thus, we assume that $c - c'$ divides both $d_1 - d_1'$ and $d_2 - d_2'$ and define $\varepsilon \leftarrow (d_1-d_1')/(c-c')$ and $\zeta \leftarrow (d_2-d_2')/(c-c')$ over the integers and $\zeta' \leftarrow (d_3-d_3')/(c-c')$ over $\mathbb{Z}_n$. This gives

$$\mathbf{C} = \mathbf{y}^{\zeta}\mathbf{g}^{\varepsilon} \quad \text{and} \quad C = y^{\zeta'}g^{\varepsilon} \quad .$$

Finally, using Lemma 10.5.8 we can find $\varepsilon_* \in [0, n-1]$ and ζ_* such that

$$\mathbf{C} = \mathbf{y}^{\zeta_*}\mathbf{g}^{\varepsilon_*} \quad .$$

We may assume that $\varepsilon_* = \varepsilon$, since otherwise we can define $\eta_0 \leftarrow \varepsilon - \varepsilon_*$, $\eta_1 \leftarrow \zeta - \zeta_*$, and $\mathbf{b} \leftarrow \mathbf{C}$ and conclude that Case 6 in Section 10.5 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. □

Protocols in the Squares Modulo An RSA-modulus

Zero-knowledge proofs of knowledge of logarithms of elements in $\mathrm{SQ}_{\mathbf{N}}$ have been studied by Fujisaki and Okamoto [47] and Damgård and Fujisaki [41]. We use similar techniques. More precisely we the consider Pedersen commitments $\mathbf{y}^s\mathbf{g}^e$ over $\mathrm{SQ}_{\mathbf{N}}$ and the problem of proving relations between the committed values in such commitments.

Protocol 10.5.9 (Knowledge of Committed Value).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{SQ}_{\mathbf{N}}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_{\mathbf{N}}^*$.
PRIVATE INPUT: $s, t \in [0, 2^{\kappa_r}\mathbf{N} - 1]$, $\mathbf{r} \in \mathrm{SQ}_{\mathbf{N}}$ 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 $\boldsymbol{\mu} \leftarrow \mathbf{y}^a \mathbf{g}^b$, 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
$$d_1 \leftarrow cs + a \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \quad \text{and} \quad d_2 \leftarrow ct + b \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N}$$
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 10.5.10. *Protocol 10.5.9 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
$$\mathbf{u}^{c-c'} = \mathbf{y}^{d_1 - d_1'}\mathbf{g}^{d_2 - d_2'} \quad .$$
If $(c-c')$ does not divide $(d_1 - d_1')$ and $(d_2 - d_2')$ we define $\eta_0 \leftarrow c - c'$, $\eta_1 \leftarrow d_1 - d_1'$, $\eta_2 \leftarrow d_2 - d_2'$, and $\mathbf{b} \leftarrow \mathbf{u}$ and conclude that Case 5 in Section 10.5 is satisfied.

Thus, we assume that $(c - c')$ divides $(d_1 - d_1')$ and $(d_2 - d_2')$ and define $\zeta \leftarrow (d_1 - d_1')/(c - c')$ and $\tau \leftarrow (d_2 - d_2')/(c - c')$. This gives
$$\mathbf{u} = \mathbf{y}^{\zeta}\mathbf{g}^{\tau} \quad .$$

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 10.5.10 (Equality of Committed Values).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \in \mathrm{SQ}_{\mathbf{N}}$ 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 $\mathbf{r} \in \mathrm{SQ}_{\mathbf{N}}$ 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
$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \leftarrow (\mathbf{y}^a\mathbf{g}^b, \mathbf{y}^b\mathbf{y}^{-b'}, \mathbf{y}^{a'}\mathbf{g}^{b'})$$
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

$$\begin{aligned} d_1 &\leftarrow cs + a \bmod 2^{\kappa_c + 2\kappa_r} z \ , \\ d_2 &\leftarrow ct + b \bmod 2^{\kappa_c + 2\kappa_r} z \ , \\ d_3 &\leftarrow cs' + a' \bmod 2^{\kappa_c + 2\kappa_r} z \ , \quad \text{and} \\ d_4 &\leftarrow ct' + b' \bmod 2^{\kappa_c + 2\kappa_r} z \ , \end{aligned}$$

   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 10.5.11. *Protocol 10.5.10 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 $\eta_0 \leftarrow c - c'$, $\eta_1 \leftarrow d_1 - d_1'$, $\eta_2 \leftarrow d_2 - d_2'$, and $\mathbf{b} = \mathbf{u}$ and conclude that Case 5 in Section 10.5 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 10.5.11 (Specific Committed Value)**.**
Common Input: $\mathbf{g}, \mathbf{y} \in \mathrm{SQ}_{\mathbf{N}}$ 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 10.5.10 on common input $\mathbf{g}, \mathbf{y}, (\mathbf{u}, \mathbf{v}), (\mathbf{1}, \mathbf{w})$ and private exponents $s, t, 0, 0$.

Lemma 10.5.12. *Protocol 10.5.11 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. This follows directly from Lemma 10.5.11. □

In Protocol 10.5.2 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 10.5.3. 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 10.5.12 (Basic Double-Decker Exponentiation)**.**
Common Input: $\mathbf{g}, \mathbf{y}, \boldsymbol{\phi} \in \mathrm{SQ}_{\mathbf{N}}$ 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
$$(F_{1,l}, F_{2,l}, \mathbf{A}_l) \leftarrow (y_{\mathbf{N}}^{e_l'} g_{\mathbf{N}}^{f_l'}, y_{\mathbf{N}}^{f_l'} g_{\mathbf{N}}^{\mathbf{y}^{f_l}}, \mathbf{y}^{e_l} \mathbf{g}^{f_l})$$
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
$$\begin{aligned} d_{1,l} &\leftarrow e_l - b_l t \bmod 2^{2\kappa_r}\mathbf{N} \ , \\ d_{2,l} &\leftarrow f_l - b_l s \bmod 2^{2\kappa_r}\mathbf{N} \ , \\ d_{3,l} &\leftarrow f_l' - b_l \mathbf{y}^{d_{2,l}} s' \bmod \mathbf{N} \ , \quad \text{and} \\ d_{4,l} &\leftarrow e_l' - b_l \mathbf{y}^{d_{2,l}} t' \bmod \mathbf{N} \ , \end{aligned}$$
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 10.5.13. *Protocol 10.5.12 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 10.5.3, 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

$$\left\{ \begin{array}{l} d_{4,l} = e'_l - b_l \mathbf{y}^{d_{2,l}} t' \\ d'_{4,l} = e'_l - b'_l \mathbf{y}^{d'_{2,l}} t' \end{array} \right\} \ .$$

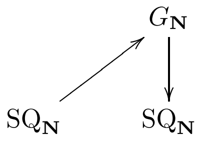
Then we have

$$\begin{aligned} F_{1,l} &= \theta^{\mathbf{y}^{d_{2,l}}} y_{\mathbf{N}}^{d_{4,l}} g_{\mathbf{N}}^{d_{3,l}} = \theta^{\mathbf{y}^{d_{2,l}}} y_{\mathbf{N}}^{\varepsilon' - \mathbf{y}^{d_{2,l}} \tau'} g_{\mathbf{N}}^{\psi' - \mathbf{y}^{d_{2,l}} \zeta'} \quad \text{and} \\ F_{1,l} &= y_{\mathbf{N}}^{d'_{4,l}} g_{\mathbf{N}}^{d'_{3,l}} = y_{\mathbf{N}}^{\varepsilon'} g_{\mathbf{N}}^{\psi'} \ . \end{aligned}$$

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. $\square$

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 $\mathrm{SQ}_{\mathbf{N}}$. To achieve this we combine the above protocol with a protocol for proving equivalence of exponents over distinct groups. This is illustrated in Figure 10.3.



$$\begin{tikzcd} & G_{\mathbf{N}} \arrow[d] \\ \mathrm{SQ}_{\mathbf{N}} \arrow[ur] & \mathrm{SQ}_{\mathbf{N}} \end{tikzcd}$$

Figure 10.3: Double-decker exponentiation proof over an RSA-modulus.

Protocol 10.5.13 (Double-Decker Exponentiation).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{h} \in \mathrm{SQ}_{\mathbf{N}}$, $(\mathbf{u}, \mathbf{v}), (\mathbf{u}', \mathbf{v}') \in (\mathbb{Z}_{\mathbf{N}}^*)^2$, and $g_{\mathbf{N}}, y_{\mathbf{N}} \in G_{\mathbf{N}}$.
PRIVATE INPUT: $\mathbf{r} \in \mathrm{SQ}_{\mathbf{N}}$, $s, t, s', t' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ 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 10.5.12 on common input $\mathbf{g}, \mathbf{y}, \boldsymbol{\phi} \in \mathrm{SQ}_{\mathbf{N}}$ 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 10.5.8 on common input $\mathbf{y}, \mathbf{h}, \mathbf{v}' \in \mathrm{SQ}_{\mathbf{N}}$, $g_{\mathbf{N}}, y_{\mathbf{N}}, v \in G_{\mathbf{N}}$ and private input $\mathbf{r}, t', t''$.

Lemma 10.5.14. *Protocol 10.5.13 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 10.5.13 we can find $\zeta'', \tau'', \zeta, \tau$ such that

$$(\theta, \omega, \boldsymbol{\phi}) = (y_{\mathbf{N}}^{\tau''} g_{\mathbf{N}}^{\zeta''}, y_{\mathbf{N}}^{\zeta''} g_{\mathbf{N}}^{\mathbf{y}^{\zeta}}, \mathbf{y}^{\tau} \mathbf{g}^{\zeta}) \enspace .$$

Thus, we can compute $\boldsymbol{\rho}$ such that

$$(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^{\zeta} \mathbf{y}^{\tau}, \mathbf{g}^{\tau} \boldsymbol{\rho}) \quad \text{and} \quad (u, v) = (g_{\mathbf{N}}^{\zeta''} y_{\mathbf{N}}^{\tau''}, g_{\mathbf{N}}^{\tau''} y_{\mathbf{N}}^{\boldsymbol{\rho}}) \enspace .$$

Using Lemma 10.5.9 we can find ζ', τ''_* and $\rho \in [0, \mathbf{N} - 1]$ such that

$$\mathbf{v}' = \mathbf{g}^{\zeta'} \mathbf{h}^{\rho} \quad \text{and} \quad v = y_{\mathbf{N}}^{\tau''_*} g_{\mathbf{N}}^{\rho} \enspace .$$

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 10.5 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 10.5.13 and Lemma 10.5.9 that the protocol is special honest verifier perfect zero-knowledge. $\square$

Protocol 10.5.14 (Knowledge of a Root of a Committed Value)**.**
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{SQ}_{\mathbf{N}}$ 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 $\mathbf{r} \in \mathrm{SQ}_{\mathbf{N}}$ 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

$$(\mathbf{A}_1, \mathbf{A}_2) \leftarrow (\mathbf{y}^a \mathbf{g}^b \mathbf{u}^e, \mathbf{y}^b \mathbf{v}^e) \ , \tag{10.21}$$
$$(\mathbf{B}_1, \mathbf{B}_2) \leftarrow (\mathbf{y}^f \mathbf{g}^h \mathbf{u}^i, \mathbf{y}^h \mathbf{v}^i) \ , \quad \text{and} \tag{10.22}$$
$$\mathbf{B}_3 \leftarrow \mathbf{y}^j \mathbf{g}^i \ . \tag{10.23}$$

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 10.5.10 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 10.5.9 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

$$d_1 \leftarrow ca + f \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N} \ , \tag{10.24}$$
$$d_2 \leftarrow cb + h \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N} \ , \tag{10.25}$$
$$d_3 \leftarrow ce + i \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N} \ , \quad \text{and} \tag{10.26}$$
$$d_4 \leftarrow cs'' + j \bmod 2^{\kappa_c + 2\kappa_r}\mathbf{N} \ . \tag{10.27}$$

4. The verifier checks that

$$\mathbf{A}_1^c \mathbf{B}_1, \mathbf{A}_2^c \mathbf{B}_2 = (\mathbf{y}^{d_1} \mathbf{g}^{d_2} \mathbf{u}^{d_3}, \mathbf{y}^{d_2} \mathbf{v}^{d_3}) \ , \quad \text{and} \tag{10.28}$$
$$\mathbf{C}^c \mathbf{B}_3 = \mathbf{y}^{d_4} \mathbf{g}^{d_3} \ . \tag{10.29}$$

Lemma 10.5.15. *Protocol 10.5.14 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

$$\begin{aligned} \mathbf{A}_1^{c-c'} &= \mathbf{y}^{d_1-d_1'}\mathbf{g}^{d_2-d_2'}\mathbf{u}^{d_3-d_3'} \ , \\ \mathbf{A}_2^{c-c'} &= \mathbf{y}^{d_2-d_2'}\mathbf{v}^{d_3-d_3'} \ , \quad \text{and} \\ \mathbf{C}^{c-c'} &= \mathbf{y}^{d_4-d_4'}\mathbf{g}^{d_3-d_3'} \ . \end{aligned}$$

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 10.5 is satisfied.

Thus, we assume that $c - c'$ divides $d_1 - d_1'$, $d_2 - d_2'$, $d_3 - d_3'$, and define $\alpha \leftarrow (d_1 - d_1')/(c - c')$, $\beta \leftarrow (d_2 - d_2')/(c - c')$, $\varepsilon \leftarrow (d_3 - d_3')/(c - c')$, and $\zeta'' \leftarrow (d_4 - d_4')/(c - c')$. 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 10.5.10 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 10.5.11 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 10.5 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 $\mathbf{A}_1, \mathbf{A}_2 \in \mathrm{SQ}_{\mathbf{N}}$ 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 10.5.15 (Equality of Exponents of Committed Values).
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{u}, \mathbf{v}, \mathbf{C} \in \mathrm{SQ}_{\mathbf{N}}$
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 $(\boldsymbol{\mu}, \boldsymbol{\nu}) \leftarrow (\mathbf{y}^a\mathbf{g}^b, \mathbf{y}^b\mathbf{h}^e)$ and $\mathbf{B} \leftarrow \mathbf{g}^e\mathbf{y}^f$ 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

$$\begin{aligned} d_1 &\leftarrow cr + a \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \\ d_2 &\leftarrow cs + b \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \\ d_3 &\leftarrow ct + e \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \quad \text{and} \\ d_4 &\leftarrow cw + f \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \end{aligned}$$

   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 10.5.16. *Protocol 10.5.15 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 10.5 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 $\rho \leftarrow (d_1 - d_1')/(c - c')$, $\zeta \leftarrow (d_2 - d_2')/(c - c')$, $\tau \leftarrow (d_3 - d_3')/(c - c')$, $\omega \leftarrow (d_4 - d_4')/(c - c')$. 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 10.5.16 (A Committed Value Can Be Written as $ka + l$).
COMMON INPUT: $\mathbf{g}, \mathbf{y} \in \mathrm{SQ}_{\mathbf{N}}$ 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
$$\begin{aligned} \mathbf{A} &\leftarrow \mathbf{y}^e \mathbf{g}^a \ , && (10.30)\\ \mathbf{B}_1 &\leftarrow \mathbf{y}^h \mathbf{g}^f \ , \quad \text{and} && (10.31)\\ \mathbf{B}_2 &\leftarrow \mathbf{y}^i \ , && (10.32) \end{aligned}$$
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
$$\begin{aligned} d_1 &\leftarrow ca + f \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , && (10.33)\\ d_2 &\leftarrow ce + h \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \ , \quad \text{and} && (10.34)\\ d_3 &\leftarrow c(ek - t) + i \bmod 2^{\kappa_c+2\kappa_r}k\mathbf{N} \ , && (10.35) \end{aligned}$$
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 10.5.17. *Protocol 10.5.16 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
$$\begin{aligned} \mathbf{A}^{c-c'} &= \mathbf{y}^{d_2-d_2'}\mathbf{g}^{d_1-d_1'} \text{ and}\\ (\mathbf{g}^l\mathbf{A}^k/\mathbf{C})^{c-c'} &= \mathbf{y}^{d_3-d_3'} \ . \end{aligned}$$
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 10.5 is satisfied.

Thus, we assume that $c - c'$ divides $d_1 - d_1'$, $d_2 - d_2'$, and $d_3 - d_3'$ and define $\alpha \leftarrow (d_1 - d_1')/(c - c')$, $\varepsilon \leftarrow (d_2 - d_2')/(c - c')$, and $\zeta \leftarrow (d_3 - d_3')/(c - c')$. 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 $\mathbf{A} \in \mathrm{SQ}_{\mathbf{N}}$, $d_1, d_2 \in [0, 2^{\kappa_c+2\kappa_r}\mathbf{N}-1]$, $d_3 \in [0, 2^{\kappa_c+2\kappa_r}k\mathbf{N}-1]$ 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 10.5.17 (Validity of Committed Signature from Hash)**.**
COMMON INPUT: $\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{z} \in \mathrm{SQ}_{\mathbf{N}}$, $\mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}', \mathbf{C}, \mathbf{C}' \in \mathbb{Z}_{\mathbf{N}}^*$, $e' \in [2^\kappa, 2^{\kappa+1}-1]$.
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

$$\begin{aligned}(\mathbf{u}, \mathbf{v}) &= (\mathbf{y}^s\mathbf{g}^r, \mathbf{y}^r\boldsymbol{\sigma}) \ , \\ (\mathbf{u}', \mathbf{v}') &= (\mathbf{y}^{s'}\mathbf{g}^{r'}, \mathbf{y}^{r'}\boldsymbol{\sigma}') \ , \\ \mathbf{C} &= \mathbf{y}^t\mathbf{g}^e \ , \\ \mathbf{C}' &= \mathbf{y}^{t'}\mathbf{g}^{w_\alpha} \ , \quad \text{and} \\ \mathsf{Vf}^{\mathsf{cs}}_{\mathrm{id}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, (\mathbf{N},\mathbf{h},\mathbf{z},e')}(w_\alpha, (e, \boldsymbol{\sigma}, \boldsymbol{\sigma}')) &= 1 \ .\end{aligned}$$

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

$$\begin{aligned}(\boldsymbol{\mu}, \boldsymbol{\nu}) &\leftarrow (\mathbf{y}^\zeta\mathbf{g}^\tau, \mathbf{y}^\tau \boldsymbol{h}^{-w_\alpha}) \ , \\ (\boldsymbol{\mu}', \boldsymbol{\nu}') &\leftarrow (\mathbf{y}^{\zeta'}\mathbf{g}^{\tau'}, \mathbf{y}^{\tau'}\mathbf{z}') \ , \\ (\boldsymbol{\mu}'', \boldsymbol{\nu}'') &\leftarrow (\mathbf{y}^{\zeta''}\mathbf{g}^{\tau''}, \mathbf{y}^{\tau''}\boldsymbol{\sigma}^e) \ , \\ (\boldsymbol{\mu}''', \boldsymbol{\nu}''') &\leftarrow (\mathbf{y}^{\zeta'''}\mathbf{g}^{\tau'''}, \mathbf{y}^{\zeta'''}H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{z}')) \ , \quad \text{and} \\ (\boldsymbol{\mu}'''', \boldsymbol{\nu}'''') &\leftarrow (\mathbf{y}^{\zeta''''}\mathbf{g}^{\tau''''}, \mathbf{y}^{\zeta''''}\mathbf{h}^{-H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{z}')}) \ .\end{aligned}$$

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 10.5.9 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 10.5.9 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 10.5.15 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 10.5.10 with $z \leftarrow \mathbf{N} + \mathbf{N}2^{\kappa+1}$ 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 10.5.14 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 10.5.13 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 10.5.13 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 10.5.11 with $z \leftarrow 2\mathbf{N}$ 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 10.5.16 with $k \leftarrow 4$ and $l \leftarrow 3$ 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 10.5.7 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 10.5.18. *Protocol 10.5.17 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 10.5.10 we can find $\zeta, \rho, \mathbf{r}$ and $\zeta', \rho', \mathbf{r}'$ such that

$$(\mathbf{u}, \mathbf{v}) = (\mathbf{y}^\zeta \mathbf{g}^\rho, \mathbf{y}^\rho \mathbf{r}) \text{ and} \tag{10.36}$$

$$(\mathbf{u}', \mathbf{v}') = (\mathbf{y}^{\zeta'} \mathbf{g}^{\rho'}, \mathbf{y}^{\rho'} \mathbf{r}') \ . \tag{10.37}$$

Using Lemma 10.5.16 we can find $\zeta_1, \rho_1, \omega, \tau$ such that

$$(\boldsymbol{\mu}, \boldsymbol{\nu}) = (\mathbf{y}^{\zeta_1} \mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1} \mathbf{h}^{-\omega}) \text{ and} \tag{10.38}$$

$$\mathbf{C}' = \mathbf{y}^{-\tau} \mathbf{g}^\omega \ . \tag{10.39}$$

Using Lemma 10.5.11 we can find $\zeta_1, \tau_1, \mathbf{r}_1, \zeta_e, \tau_e, \mathbf{r}_e$ such that

$$(\boldsymbol{\mu}', \boldsymbol{\nu}') = (\mathbf{y}^{\zeta_1} \mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1} \mathbf{r}_1) \text{ and} \tag{10.40}$$

$$(\boldsymbol{\mu}(\mathbf{u}')^{e'}, \boldsymbol{\nu}(\mathbf{v}')^{e'}) = (\mathbf{y}^{\zeta_e} \mathbf{g}^{\rho_e}, \mathbf{y}^{\rho_e} \mathbf{r}_1) \ . \tag{10.41}$$

If we combine Equations (10.36), (10.38), and (10.41) we get

$$(\mathbf{y}^{\zeta_1+\zeta' e'}\mathbf{g}^{\rho_1+\rho' e'}, \mathbf{y}^{\rho_1+\rho' e'}\mathbf{h}^{-\omega}\mathbf{r}'^{e'}) \quad = \quad (\mathbf{y}^{\zeta_e}\mathbf{g}^{\rho_e}, \mathbf{y}^{\rho_e}\mathbf{r}_1) \ .$$

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 10.5 is satisfied. Thus, we have

$$(\boldsymbol{\mu}', \boldsymbol{\nu}') \quad = \quad (\mathbf{y}^{\zeta_1}\mathbf{g}^{\rho_1}, \mathbf{y}^{\rho_1}\mathbf{h}^{-\omega}\mathbf{r}'^{e'}) \ .$$

Using Lemma 10.5.15 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 10.5 is satisfied.

Using Lemma 10.5.14 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 10.5 is satisfied.

Using Lemma 10.5.14 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 10.5 is satisfied.

Using Lemma 10.5.12 we can find ζ_5, τ_5 and $\mathbf{z}$ such that

$$(\boldsymbol{\mu}''\boldsymbol{\mu}'''', \boldsymbol{\nu}''\boldsymbol{\nu}'''') \quad = \quad (\mathbf{y}^{\zeta_5}\mathbf{g}^{\rho_5}, \mathbf{y}^{\rho_5}\mathbf{z}) \ .$$

We may assume that $(\zeta_5, \rho_5) = (\zeta_2' + \zeta_4', \rho_2' + \rho_3')$ and thus we have $\mathbf{z} = \mathbf{r}^{\varepsilon}\mathbf{h}^{\mathbf{h}^{\mathbf{r}_3}} = \mathbf{r}^{\varepsilon}\mathbf{h}^{H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}(\mathbf{h}^{-\omega}\mathbf{r}'^{e'})}$, since otherwise Case 6 in Section 10.5 is satisfied.

Using Lemma 10.5.17 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 10.5 is satisfied.

Using Lemma 10.5.8 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 10.5 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 $\boldsymbol{\mu}', \boldsymbol{\mu}'', \boldsymbol{\mu}''', \boldsymbol{\mu}'''', \boldsymbol{\nu}', \boldsymbol{\nu}'', \boldsymbol{\nu}''', \boldsymbol{\nu}'''' \in \mathrm{SQ}_{\mathbf{N}}$ 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. □

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 10.3.2. The common input consists of a chain of ciphertexts and commitments of a $\mathcal{SS}^{\mathsf{cs}}$ signature of the public keys corresponding to the path of the signer in the tree.

Protocol 10.5.18 (Valid $\mathcal{HGS}$ Signature).
COMMON INPUT:

$$\begin{array}{ll}
H^{\mathsf{CHP}} = (h_1, \ldots, h_\delta) \in G_{q_2}^{\delta} & (u_l, v_l, u_l', v_l')_{l=0}^{\delta-1} \in G_{q_3}^{4\delta} \\
g_1, y_1 \in G_{q_1}, g_2, y_2 \in G_{q_2}, g_3, y_3 \in G_{q_3} & C_\delta \in G_{q_3}^{4} \\
y_{\alpha_0} \in G_{q_3}, Y \in G_{q_3}^{5} & \mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}', \mathbf{C} \in \mathrm{SQ}_{\mathbf{N}} \\
e' \in [2^\kappa, 2^{\kappa+1}-1] & \\
\mathbf{g}, \mathbf{y}, \mathbf{h}, \mathbf{z} \in \mathrm{SQ}_{\mathbf{N}} &
\end{array}$$

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

$$\begin{aligned}
(u_l, v_l) E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(y_{\alpha_{l+1}}, r_l) &\text{ for } l = 0, \ldots, \delta - 1 \ , \\
(u_l', v_l') = E^{\mathsf{elg}}_{(y_{\alpha_l}, g_3)}(1, r_l') &\text{ for } l = 0, \ldots, \delta - 1 \ , \\
C_\delta = E^{\mathsf{CCA}}_Y(y_{\alpha_\delta}, r_\delta) &\ , \\
\mathbf{u} = \mathbf{y}^s \mathbf{g}^r &\ , \\
\mathbf{u}' = \mathbf{y}^{s'} \mathbf{g}^{r'} &\ , \\
\mathbf{C} = \mathbf{y}^t \mathbf{g}^e &\ , \quad \text{and} \\
\mathsf{Vf}^{\mathsf{cs}}_{H^{\mathsf{CHP}}, H^{\mathsf{Sh}}_{(\mathbf{N},\mathbf{g})}, (\mathbf{N},\mathbf{h},\mathbf{z},e')}((y_{\alpha_1}, \ldots, y_{\alpha_\delta}), (e, \mathbf{v}/\mathbf{y}^r, \mathbf{v}'/\mathbf{y}^{r'})) = 1 &\ .
\end{aligned}$$

1. The prover chooses $s, t \in \mathbb{Z}_{q_2}$ and $t' \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ randomly and computes $(\mu, \nu) \leftarrow (y_1^s g_1^t, y_1^t g_1^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})})$, $\mathbf{C}' \leftarrow \mathbf{y}^{t'} \mathbf{g}^{H^{\mathsf{CHP}}(y_{\alpha_1}, \ldots, y_{\alpha_\delta})}$. The prover hands (μ, ν) and $\mathbf{C}'$ to the verifier.

2. The following protocols are executed in parallel

   a) Protocol 10.5.6 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 10.5.8 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 10.5.17 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 10.5.19. *Protocol 10.5.18 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 10.5.7 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

$$\begin{aligned}
(u_l, v_l, u_l', v_l') &= (E^{\mathsf{elg}}_{(\gamma_l, g_3)}(\gamma_{l+1}, \rho_l), E^{\mathsf{elg}}_{(\gamma_l, g_3)}(1, \rho_l')) \text{ for } l = 0, \ldots, \delta - 1 \ , \\
C_\delta &= E^{\mathsf{CCA}}_Y(\gamma_\delta, \rho_\delta) \ , \quad \text{and} \\
(\mu, \nu) &= (y_1^{\zeta''} g_1^{\tau''}, y_1^{\tau''} g_1^{H^{\mathsf{CHP}}(\gamma_1, \ldots, \gamma_\delta)}) \ .
\end{aligned}$$

Using Lemma 10.5.9 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 10.5 is satisfied. Using Lemma 10.5.18 we can find $\rho, \zeta, \rho', \zeta', \tau, \tau', \varepsilon, \omega^*$ and $\mathbf{r}, \mathbf{r}' \in \mathrm{SQ}_\mathbf{N}$

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 10.5 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 $\mathbf{C}' \in \mathrm{SQ}_{\mathbf{N}}$ 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. □

A Computationally Convincing Proof of Knowledge

We are finally ready to prove the main results of this section.

Proposition 10.5.20. *Protocol 10.5.18 is honest verifier statistical zero-knowledge.*

Proof. This is an immediate consequence of 10.5.19. □

Proposition 10.5.21. *Protocol 10.5.18 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 10.5.19 that the protocol is a Σ-protocol for the relation $\mathcal{R} = \mathcal{R}_{\mathrm{HGS}} \vee \mathcal{R}_{\mathrm{DL}} \vee \mathcal{R}_{\mathrm{SRSA}}$. From Lemma 8.5.11 follows that the protocol is a proof of knowledge for the relation $\mathcal{R}_{\mathrm{HGS}} \vee \mathcal{R}_{\mathrm{DL}} \vee \mathcal{R}_{\mathrm{SRSA}}$. 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

$$\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}), r_{\mathrm{p}}), \mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}), r_{\mathrm{p}})) \in \mathcal{R} \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}), r_{\mathrm{p}}) \geq \kappa^{-c}] \qquad (10.42)$$

is overwhelming.

We argue that $\mathcal{X}$ is an extractor for a computationally convincing proof of knowledge for the relation $\mathcal{R}_{\mathrm{HGS}}$. The requirement on the running time of $\mathcal{X}$ in Definition 8.5.1 follows immediately. Suppose that the requirement on the output of $\mathcal{X}$ in Definition 8.5.1 does not hold. Then there exists an adversary P^*, a constant

c, and an infinite index set $\mathcal{N}$ such that

$$\begin{aligned}\Pr_{(\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}}[\delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] &\geq \kappa^{-c} \\ \Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \notin \mathcal{R}_{\mathrm{HGS}} \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}] &\geq \kappa^{-c} \ .\end{aligned}$$

The union bound and the fact that the probability in Equation (10.42) is overwhelming implies that

$$\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \in \mathcal{R}_{\mathrm{DL}} \vee \mathcal{R}_{\mathrm{SRSA}} \mid \delta^V_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}) \geq \kappa^{-c}]$$

is at least $\frac{1}{2\kappa^c}$ and we conclude that $\Pr[(I_{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}}),\mathcal{X}^{P^*}((\mathbf{\Gamma},\overline{g}),r_{\mathrm{p}})) \in \mathcal{R}_{\mathrm{DL}} \vee \mathcal{R}_{\mathrm{SRSA}}] \geq \frac{1}{2\kappa^{2c}}$.

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

$$\Pr[A(\mathbf{\Gamma},\overline{g}) \in \mathcal{R}_{\mathrm{DL}} \vee \mathcal{R}_{\mathrm{SRSA}}] \geq \frac{1}{4\kappa^{2c}} \ .$$

It follows from Lemma 10.2.11 and Lemma 10.2.5 that this contradicts either the DL-assumption, Definition 10.2.4, or the strong RSA-assumption, Definition 2.3.2, 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

$$(D^{\mathsf{elg}}_{1/sk(\alpha_0)}(u_0,v_0),\ldots,D^{\mathsf{elg}}_{1/sk(\alpha_{\delta-1})}(u_{\delta-1},v_{\delta-1})) = (pk(\alpha_1),\ldots,pk(\alpha_\delta)) \ .$$

Denote the resulting protocol by π'_{hgs}. The following result follows similarly to the proposition above.

Proposition 10.5.22. *Let T be a tree with all leaves at the same depth. Then Protocol 10.5.18 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.*

Prot.	Prover	Verifier	Setup
10.5.1	$25\delta - 2$	$30\delta - 3$	
10.5.2	$2 + (10.5.3)$	$2 + (10.5.3)$	
10.5.3	$7.5\kappa_c$	$7.5\kappa_c$	2 bases
10.5.4	5	8	
10.5.5	3	6	
10.5.6	$7\delta + 2 + (10.5.1)+$ $(\delta + 1) \cdot (10.5.2)$ $(10.5.4) + (10.5.5)$	$(10.5.1)+$ $(\delta + 1) \cdot (10.5.2)+$ $(10.5.4) + (10.5.5)$	
10.5.8	10	18	
10.5.9	2	3	
10.5.10	2	3	
10.5.11	$(10.5.10)$	$(10.5.10)$	
10.5.13	$6 + (10.5.8) + (10.5.12)$	$(10.5.8) + (10.5.12)$	
10.5.12	$7.5\kappa_c$	$7.5\kappa_c$	2 bases
10.5.14	$12 + 2 \cdot (10.5.9) + (10.5.10)$	$8 + 2 \cdot (10.5.9) + (10.5.10)$	
10.5.15	4	9	
10.5.16	7	7	
10.5.7	6	12	
10.5.17	$20 + 2 \cdot (10.5.9)+$ $(10.5.10) + (10.5.11)+$ $2 \cdot (10.5.13) + (10.5.14)+$ $(10.5.15) + (10.5.16)+$ $(10.5.7)$	$2 \cdot (10.5.9)+$ $(10.5.10) + (10.5.11)+$ $2 \cdot (10.5.13) + (10.5.14)+$ $(10.5.15) + (10.5.16)+$ $(10.5.7)$	
10.5.18	$6 + \delta + (10.5.6)+$ $(10.5.8) + (10.5.17)$	$\delta + (10.5.6)+$ $(10.5.8) + (10.5.17)$	

Table 10.1: Number of exponentiations in each subprotocol

10.6 Complexity Analysis

We now analyze the number of exponentiations needed for some typical parameters. Table 10.1 shows the number of exponentiations necessary for each protocol.

Since Protocol 10.5.18 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 [22] 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 10.5.3 or 10.5.12 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 10.1 and adding a setup phase for 10 bases, we 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., [67]. 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 11

An Optimistic Construction

11.1 About the Construction

The two constructions of proper hierarchical group signature of this thesis are both more or less impractical, although the second one can be implemented and run on a modern workstation.

In this chapter we provide an optimistic construction for hierarchical group signatures $\mathcal{HGS}$ that is more efficient and satisfies Definition 8.3.4 under the strong RSA assumption and the.decisional Diffie-Hellman assumption in the random oracle model. For an honest signer S, the group managers on the path to S can open the signature. A dishonest signer can produce a signature that cannot be opened by some or all group managers on the path, but such a signature can be opened by a trusted party. Put differently, a corrupt signer can force the group managers to perform extra work, but it will still be identified in the process. This is the optimistic property of the scheme. We stress that also in the optimistic scheme no group manager not on the path to S can open the signature. Hence no dishonest signer can frame another signer even temporarily.

The following proposition captures the security of our construction.

Proposition 11.1.1. *The optimistic hierarchical group signature scheme $\mathcal{HGS}$ is secure under the strong RSA assumption and the DDH assumption in the random oracle model.*

Although our construction bears some resemblance to the constructions given earlier in this part, there are important differences. Thus, before we give details we explain the key ideas.

The Basic Idea of the Construction

Each group manager M_β holds a public key y_β and a corresponding secret key x_β of an ElGamal cryptosystem over G_Q. Each party M_β (or S_β) is assigned an element

h^{e_β}, where the integers e_β are chosen such that no parties on the same level have identical elements.

Let us first consider the problem of hiding the identity of the signer such that no group manager can extract more information from a signature than it should be able to. Let $\alpha_0, \ldots, \alpha_\delta$ be a path from the root $\omega = \alpha_0$ to a leaf $\alpha_\delta = \alpha$. The idea is that the signer S_α encrypts the chain $h^{e_{\alpha_1}}, \ldots, h^{e_{\alpha_\delta}}$ using the keys of the group managers along this chain, i.e., it computes

$$((u_0, v_0), \ldots, (u_{\delta-1}, v_{\delta-1})) = (E^{\mathsf{elg}}_{y_{\alpha_0}}(h^{e_{\alpha_1}}), \ldots, E^{\mathsf{elg}}_{y_{\alpha_{\delta-1}}}(h^{e_{\alpha_\delta}})) \enspace ,$$

and includes this as part of the signature. This ensures that no group manager can extract more information than it should be able to from a signature.

We must also guarantee that only legitimate signers can compute a signature. To do this an RSA-modulus $\mathbf{N}$ and random elements $\mathbf{g}, \mathbf{h}, \mathbf{y} \in SQ_{\mathbf{N}}$ are used. Each signer S_α is given as its secret key a p_αth root $\mathbf{z}_\alpha$ of $\mathbf{y}$ modulo $\mathbf{N}$, where p_α is a prime. Recall that under the strong RSA assumption it is infeasible to compute such a root without knowledge of the factorization of $\mathbf{N}$. When computing a signature it forms a commitment of its secret key $(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^t \mathbf{h}^{t'}, \mathbf{h}^t \mathbf{z}_\alpha)$, using some random exponents t and t', and proves knowledge of how to open the commitment to a non-trivial root of $\mathbf{y}$. Although proving knowledge of a root guarantees that the signer is legitimate, there is no mechanism for enforcing a relation between the plaintexts encrypted in the chain of ciphertexts and power of the root. To ensure this the signer essentially proves that if the integers e_{α_i} are concatenated, then the result equals the prime p_α. We now take a closer look at how the integers e_{α_i} are generated and concatenated.

The Identifiers

The short identification string, denoted by e_β, is assigned to each group manager M_β by simply enumerating the group managers on each level in the tree. Formally, this gives a map $I_M : \mathcal{V}(T) \setminus \mathcal{L}(T) \to [0, 2^{t_M} - 1]$ for some integer t_M such that 2^{t_M} bounds the number of group managers on each level in the tree. The identifier of the root is always $2^{t_M} - 1$ to ensure that its string begins with a one. Each signer S_α is also assigned a short identification string e_α. This string is not assigned in the same way as for group managers, but it is chosen such that no two signers on the same level have identical strings. Formally, this gives a map $I_S : \mathcal{L}(T) \to [0, 2^{t_M} - 1]$, where 2^{t_M} also upper bounds the number of signers. It is straightforward to modify our construction to be more efficient by using different values of t_M for different levels of the tree, in particular for the signers. We use a single value to simplify the presentation.

Before we define the map I_S we consider how the maps can be combined. Suppose that $\alpha_0, \ldots, \alpha_\delta$ is a path from the root $\omega = \alpha_0$ to some leaf $\alpha_\delta = \alpha$ in the tree. Then due to the uniqueness property of the identification strings, the signer S_α is uniquely identified by the concatenation $e_{\alpha_1} | e_{\alpha_2} | \cdots | e_{\alpha_\delta}$ of the strings e_{α_i} corresponding to its path. For technical reasons we introduce a string on the form $100\ldots0$

between each pair of identification strings. More precisely, we define $\lambda = 2^{t_M + t_P + 1}$, where $t_P = \kappa_r + \kappa_c$ is a parameter of the construction, and view the concatenation procedure as the map

$$\begin{aligned} \iota_\delta : [-(\lambda/2-1), \lambda/2-1]^\delta &\rightarrow [0, \lambda^\delta - 1] \\ \iota_\delta : (e_1, \ldots, e_\delta) &\mapsto \sum_{i=0}^{\delta-1} \lambda^{\delta-i-1}(\lambda/2 + e_i) \ . \end{aligned}$$

It is not hard to see that the map has the following property.

Lemma 11.1.2. *The function ι_δ is injective.*

Lemma 11.1.2. The proof follows by induction on δ. For $\delta = 1$ the claim follows by inspection. Suppose that the claim holds for some $\delta > 1$, and note that $\iota_{\delta+1}(e_1, \ldots, e_{\delta+1}) = \lambda^\delta(\lambda/2 + e_1) + \iota_\delta(e_2, \ldots, e_{\delta+1})$. Suppose now that $\iota_{\delta+1}(e_1, \ldots, e_{\delta+1}) = \iota_{\delta+1}(e'_1, \ldots, e'_{\delta+1})$. We know that $\iota_\delta(e_2, \ldots, e_{\delta+1}), \iota_\delta(e'_2, \ldots, e'_{\delta+1}) \in [0, \lambda^\delta - 1]$ and by the induction hypothesis we have $(e_2, \ldots, e_{\delta+1}) = (e'_2, \ldots, e'_{\delta+1})$. Thus, we conclude that $e_1 = e'_1$ and the claim follows. □

We now define the function I_S such that for each signer S_α the unique integer $p_\alpha = \iota_\delta(\alpha)$ assigned to it is prime. However, we still need to say a word on how such primes are found. We need an assumption stating that, given an integer in $[0, 2^{\kappa_m} - 1]$, an integer $s \in [0, 2^{\kappa_p} - 1]$ can be found such that $e = 2^{\kappa_p} m + s$ is prime. In practice this is not really a problem if the parameters κ_m and κ_p are chosen appropriately, since, loosely speaking, there are primes everywhere.

Definition 11.1.3 (f-Embedding Assumption). *The f-embedding assumption, for a function $f : \mathbb{N} \rightarrow \mathbb{N}$, states that there exists a probabilistic polynomial time algorithm Emb_f that on input 1^κ and $m \in [0, 2^\kappa - 1]$ with overwhelming probability outputs an integer $s \in [0, 2^{f(\kappa)} - 1]$ such that $e = 2^{f(\kappa)} m + s$ is a positive prime.*

Construction of the Proof of Knowledge

In addition to the chain of ciphertexts $((u_0, v_0), \ldots, (u_{\delta-1}, v_{\delta-1}))$, the signer computes a list $(\mathbf{c}_0, \ldots, \mathbf{c}_{\delta-1})$ of Fujisaki-Okamoto commitments [47] on the form $\mathbf{c}_i = \mathbf{g}^{t_i} \mathbf{h}_i^{e_{\alpha_{i+1}}} \bmod \mathbf{N}$, where

$$(\mathbf{h}_0, \ldots, \mathbf{h}_{\delta-3}, \mathbf{h}_{\delta-2}, \mathbf{h}_{\delta-1}) = (\mathbf{h}^{\lambda^{\delta-1}}, \ldots, \mathbf{h}^{\lambda^2}, \mathbf{h}^\lambda, \mathbf{h}) \ .$$

Then for $i = 0, \ldots, \delta-1$ the signer proves that the same integer $e_{\alpha_{i+1}}$ is used to form both (u_i, v_i) and $\mathbf{c}_i$ using a standard Schnorr-like proof of knowledge over groups of unknown order. In addition to showing that the same integer is used, this protocol also ensures that $e_{\alpha_i} \in [-\lambda/2+1, \lambda/2-1]$ and $e_{\alpha_\delta} \in [-\lambda/2+1, \lambda/2-1]$. The purpose of doing this is that if the signer is honest the product $\mathbf{c} = \prod_{i=0}^{\delta-1} \mathbf{h}_i^{\lambda/2} \mathbf{c}_i \bmod \mathbf{N}$ is on the form $\mathbf{g}^t \mathbf{h}^{p_\alpha} \bmod \mathbf{N}$. On the other hand, if the integers e_{α_i} do not correspond to a path to one of the signers, then the product of the commitments is on the form $\mathbf{g}^t\mathbf{h}^n \bmod \mathbf{N}$, where n is distinct from p_α for every signer S_α due to Lemma 11.1.2.

The proof that the e_{α_i} exponents are identical in the commitments and the ciphertexts may not disclose which public key (g, y_{α_i}) is used to form the ciphertext (u_i, v_i), since this would disclose that the signer is contained in the subtree rooted at α_i. To avoid this, the signer uses a randomized key $(g'_i, y'_i) = (g^{r'_i}, y_{\alpha_i}^{r'_i})$ for some random $r'_i \in \mathbb{Z}_Q$ of the public key (g, y_{α_i}) it uses, and gives the proof relative to this key instead. It is at this point the scheme is optimistic, since it is not ensured that the signer does not use some other arbitrary public key instead of (g, y_{α_i}).

11.2 The Algorithms of the Scheme

We are now ready to describe the details of the construction.

Algorithm 11.2.1 (Key Generation, $\mathsf{HGKg}(1^\kappa, T)$)**.**

1. Assign a short identification string e_α to each group manager M_α or signer S_α as described above. Formally, this assignment is part of *hpk*.

2. Choose a random κ-bit safe prime $P = 2Q + 1$, let G_Q be the group of squares modulo P, and choose $g, h, y \leftarrow_R G_Q$ randomly. For each group manager M_β choose $x_\beta \leftarrow_R \mathbb{Z}_Q$ randomly, compute $y_\beta \leftarrow g^{x_\beta}$, and define $(hpk(\beta), hsk(\beta)) \leftarrow (y_\beta, x_\beta)$. For the trusted party, choose $x_{\mathfrak{T}} \leftarrow_R \mathbb{Z}_Q$ randomly, compute $y_{\mathfrak{T}} \leftarrow g^{x_{\mathfrak{T}}}$, and define $sk_{\mathfrak{T}} \leftarrow (x_{\mathfrak{T}}, x_\omega)$.

3. Choose two random $\kappa/2$-bit safe primes p and q, define $\mathbf{N} \leftarrow pq$, and choose $\mathbf{g}, \mathbf{h}, \mathbf{y} \leftarrow_R SQ_{\mathbf{N}}$ randomly. Then define $(\mathbf{h}_0, \ldots, \mathbf{h}_{\delta-3}, \mathbf{h}_{\delta-2}, \mathbf{h}_{\delta-1}) = (\mathbf{h}^{\lambda^{\delta-1}}$, $\ldots$, $\mathbf{h}^{\lambda^2}$, $\mathbf{h}^\lambda$, $\mathbf{h})$. For each signer S_α, compute $\mathbf{z}_\alpha \leftarrow \mathbf{y}^{1/p_\alpha}$, where $p_\alpha = \iota_\delta(\alpha)$. Then define $(hpk(\alpha), hsk(\alpha)) \leftarrow (p_\alpha, \mathbf{z}_\alpha)$.

4. Redefine $hpk(\omega)$ to include $((P, g, h, y), (\mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}), y_{\mathfrak{T}})$, and output $(hpk, hsk, sk_{\mathfrak{T}})$.

Before we describe the signature algorithm we introduce some notation. Denote by $\mathsf{SigBody}$ the deterministic algorithm that takes two inputs, a common input s_{com} and a secret input s_{sec}. The common input consists of an integer $\delta > 0$, a safe prime $P = 2Q + 1$, $g, h, y_{\mathfrak{T}} \in G_Q$, an integer $\mathbf{N}$, and $\mathbf{g}, \mathbf{h}, \mathbf{y} \in SQ_{\mathbf{N}}$. The secret input consists of $y_i \in G_Q$, $r, r_i, r'_i, r''_i \in \mathbb{Z}_Q$ and $e_i, t_i \in \mathbb{Z}$. If the inputs are not on the expected form the algorithm outputs $s_{out} = \perp$. Otherwise the algorithm computes $e = \iota_\delta(e_1, \ldots, e_\delta)$,

$$\begin{array}{rcll}
(u'_0, v'_0) & \leftarrow & E^{\mathsf{elg}}_{(g,y_0)}(h^{e_1}, r''_0) & \textit{// For root group manager} \\
\mathbf{c}_0 & \leftarrow & \mathbf{g}^{t_0}\mathbf{h}_0^{e_1} & \\
\end{array}$$

$$\left(\begin{array}{rcl}
(g'_i, y'_i) & \leftarrow & (g^{r'_i}, y_i^{r'_i}) \\
(u_i, v_i) & \leftarrow & E^{\mathsf{elg}}_{(g,y)}(y_i, r_i) \\
(u'_i, v'_i) & \leftarrow & E^{\mathsf{elg}}_{(g'_i,y'_i)}(h^{e_{i+1}}, r''_i) \\
(u''_i, v''_i) & \leftarrow & E^{\mathsf{elg}}_{(g,y_{\mathfrak{T}})}(h^{e_{i+1}}, r''_i) \\
\mathbf{c}_i & \leftarrow & \mathbf{g}^{t_i}\mathbf{h}_i^{e_{i+1}}
\end{array}\right)_{i=1}^{\delta-1}
\begin{array}{l}
\textit{// Randomized public key} \\
\textit{// Commitment of public key} \\
\textit{// For group manager on level i} \\
\textit{// For trusted party} \\
\
\end{array}$$

$$(u, v) \leftarrow E^{\mathsf{elg}}_{(g,y)}(h^e, r)$$

and outputs $s_{out} = ((u'_0, v'_0), \mathbf{c}_0, \big((g'_i, y'_i), (u_i, v_i), (u'_i, v'_i), (u''_i, v''_i), \mathbf{c}_i\big)_{i=1}^{\delta-1}, (u, v))$.

Denote by $\mathcal{R}_{\mathsf{HGSig}}$ the relation consisting of pairs

$$\big((\delta, P, g, h, y_{\mathfrak{T}}, \mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}, s_{out}, (\mathbf{u}, \mathbf{v})), (r, (y_i, r_i, r'_i, r''_i, e_i, t_i)_i, t, t')\big)$$

such that $s_{out} = \mathsf{SigBody}(s_{com}, (r, (y_i, r_i, r'_i, r''_i, e_i, t_i)_i))$, $e_i \in [-\lambda/2 + 1, \lambda/2 - 1]$, $\mathbf{u} = \mathbf{g}^t\mathbf{h}^{t'} \bmod \mathbf{N}$, and $(\mathbf{u}/\mathbf{h}^t)^{\iota_\delta(e_1,\ldots,e_\delta)} = \mathbf{y} \bmod \mathbf{N}$. In Appendix 11.4 we construct a public-coin honest verifier zero-knowledge proof of knowledge $(P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$ for $\mathcal{R}_{\mathsf{HGSig}}$.[1] We use the notation $(P_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}, V_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)})$ for the signature scheme constructed by applying the Fiat-Shamir heuristic to $(P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$.

Algorithm 11.2.2 (Signing, $\mathsf{HGSig}(m, T, hpk, hsk(\alpha))$)**.** Let $(\alpha_0, \ldots, \alpha_\delta)$ be the path from the root $\omega = \alpha_0$ to $\alpha = \alpha_\delta$ in T and let $s_{com} = (\delta, P, g, h, y_{\mathfrak{T}}, \mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y})$.

1. Choose $r, r_i, r'_i, r''_i \in \mathbb{Z}_Q$, and $t, t', t_i \in [0, 2^{\kappa_r}\mathbf{N} - 1]$ randomly and compute $s_{out} \leftarrow \mathsf{SigBody}(s_{com}, (r, (y_{\alpha_i}, r_i, r'_i, r''_i, e_{\alpha_i}, t_i)_i))$ and $(\mathbf{u}, \mathbf{v}) \leftarrow (\mathbf{g}^t\mathbf{h}^{t'}, \mathbf{h}^t\mathbf{z}_\alpha)$.
2. Compute a proof $\pi = P_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}\big((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), (r, (y_{\alpha_i}, r_i, r'_i, r''_i, e_{\alpha_i}, t_i)_i, t, t')\big)$, and output the signature $\sigma = (s_{out}, (\mathbf{u}, \mathbf{v}), \pi)$.

Note that, although for an honest signer $e_{\alpha_i} \in [0, 2^{t_M} - 1]$, the proof of knowledge only guarantees that it lies in the larger interval $[-2^{t_P+t_M} + 1, 2^{t_P+t_M} - 1]$. This is sufficient for the scheme to be secure, and allows a more efficient construction of the proof of knowledge.

Algorithm 11.2.3 (Verification, $\mathsf{HGVf}(T, hpk, m, \sigma)$)**.** Let $s_{com} = (\delta, P, g, h, y_{\mathfrak{T}}, \mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y})$. On input a candidate signature $\sigma = (s_{out}, (\mathbf{u}, \mathbf{v}), \pi)$ it outputs $V_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), \pi)$.

Algorithm 11.2.4 (Optimistic Opening, $\mathsf{HGOptOpen}(T, hpk, hsk(\beta), m, \sigma)$)**.**

1. If $\mathsf{HGVf}(T, hpk, m, \sigma) = 0$ or $\beta \notin \mathcal{V}(T) \setminus \mathcal{L}(T)$, then return $\perp$.

[1] Strictly speaking this is not true, but given that $\mathbf{g}$, $\mathbf{h}$, $\mathbf{y}$ are squares we can extract a witness with overwhelming probability over the choice of $\mathbf{N}$ under the strong RSA assumption. The reader is referred to Appendix 11.4 for details.

2. Let $l \leftarrow \mathsf{level}_T(\beta)$ and compute $a \leftarrow D^{\mathsf{elg}}_{x_\beta}(u'_l, v'_l)$. If $l > 0$ and $(g'_l)^{x_\beta} \neq y'_l$, then return $\perp$.

3. If there exists $\alpha \in \beta$ such that $a = h^{e_\alpha}$, then return α and otherwise return $\perp$.

We remind the reader that due to the optimistic nature of our scheme the output of the opening algorithm may be $\perp$, despite that the signer belongs to the subtree rooted at the group manager M_β that tries to open the signature.

Algorithm 11.2.5 (Trusted Opening, $\mathsf{HGTrustOpen}(T, hpk, sk_\mathfrak{T}, \beta, m, \sigma)$)**.**

1. If $\mathsf{HGVf}(T, hpk, m, \sigma) = 0$ or $\beta \notin \mathcal{V}(T) \setminus \mathcal{L}(T)$, then return $\perp$. Parse $sk_\mathfrak{T}$ as $(x_\mathfrak{T}, x_\omega)$.

2. Let $l \leftarrow \mathsf{level}_T(\beta)$. If $l = 0$, then let $a \leftarrow D^{\mathsf{elg}}_{x_\omega}(u'_0, v'_0)$, and otherwise let $a \leftarrow D^{\mathsf{elg}}_{x_\mathfrak{T}}(u''_l, v''_l)$.

3. If there exists $\alpha \in \beta$ such that $a = h^{e_\alpha}$, then return α and otherwise return $\perp$.

On Distributing the Trusted Party

It may be desirable to execute the trusted opening algorithm as a distributed protocol, where only the initiating group manager learns the answer. Although we do not detail a solution to this problem it is easy to achieve using known techniques. To see this consider the following alternative way of implementing the trusted opening algorithm when the initiating group manager is not the root group manager, i.e., $l > 0$ in Step 2.

1. If $\mathsf{HGVf}(T, hpk, m, \sigma) = 0$ or $\beta \notin \mathcal{V}(T) \setminus \mathcal{L}(T)$, then return $\perp$.

2. Let $l \leftarrow \mathsf{level}_T(\beta)$. Choose $r \in \mathbb{Z}_Q$ randomly and output $D^{\mathsf{elg}}_{x_\mathfrak{T}}\big(u_l(u''_{l-1})^r,\ v_l(v''_{l-1}h^{-e_\beta})^r\big)$.

Note that if β is allowed to open σ, then $D^{\mathsf{elg}}_{x_\mathfrak{T}}\big(u_l(u''_{l-1})^r,\ v_l(v''_{l-1}h^{-e_\beta})^r\big) = D^{\mathsf{elg}}_{x_\mathfrak{T}}(u_l, v_l)$. However, if β should not be able to open the signature, then the decryption $D^{\mathsf{elg}}_{x_\mathfrak{T}}\big(u_l(u''_{l-1})^r,\ v_l(v''_{l-1}h^{-e_\beta})^r\big)$ is randomly distributed in G_Q. Thus, the algorithm either outputs h^{e_α} or a random element, corresponding to an output α or an output $\perp$ in the original formulation.

Note that the algorithm consists of simply randomly re-encrypting and then decrypting an ElGamal ciphertexts. A group of parties can jointly verifiably re-encrypt a ciphertext by simply re-encrypting it one after the other and proving knowledge of the used randomness. Decryption can be implemented in a distributed way using the scheme by Jarecki and Lysyanskaya [54] with two minor modifications. The first modification is a simplification due to the presence of a trusted key generator. Instead of jointly generating a key, the trusted key generator generates the key and distributes the shares. The second modification is that the plaintext is not revealed to the involved trusted parties, but only to the initiating group manager.

When the initiating is the root manager M_ω, then the implementation as a protocol is trivial. Since trusted opening for the root uses only x_ω which is known to M_ω, no interaction is necessary.

11.3 Proof of Security

Sketch of Proof of Security

Traceability. Given an adversary that breaks traceability, we simulate the traceability experiment and break the strong RSA assumption. For the simulation we need to handle queries to corrupt signers and simulate the signature oracle. The signature oracle is easily simulated without any non-trivial root in the standard way by programming the random oracle, and how the corruptions are handled depends on the type of forger.

The first case in the traceability experiment never occurs, since there is only one root group manager and $D^{\mathsf{elg}}_{x_{\mathfrak{T}}}(u''_l, v''_l) = h^{e_\beta}$ holds for at most one group manager M_β. If the second case occurs, then due to the knowledge extractor we can find an element $\mathbf{z}$ and an integer $\iota_\delta(\varepsilon_1, \ldots, \varepsilon_\delta) > 2$ such that $\mathbf{z}^{\iota_\delta(e_1,\ldots,e_\delta)} = \mathbf{y}$. A breaker of the strong RSA assumption taking input $(\mathbf{N}, \mathbf{y}')$ is then constructed. There are two subcases: either $\iota_\delta(e_1, \ldots, e_\delta)$ identifies an honest signer or it does not. In the former subcase we guess the signer $S_{\alpha'}$ under attack. Using a standard trick we then prepare all the roots needed to simulate the traceability experiment without the factorization of $\mathbf{N}$ (given that we guessed correctly). More precisely, we define $\mathbf{y} = (\mathbf{y}')^{\prod_\alpha p_\alpha}$, where the product is taken either over all signers S_α except $S_{\alpha'}$ or over all signers depending on the case. Then if we guessed correctly, the relative primality of $\iota_\delta(e_1, \ldots, e_\delta)$ and $\prod_\alpha p_\alpha$ imply that we can compute integers a and b such that $a\iota_\delta(e_1, \ldots, e_\delta) + b\prod_\alpha p_\alpha = 1$ and $((\mathbf{y}')^a\mathbf{z}^b)^{\iota_\delta(e_1,\ldots,e_\delta)} = \mathbf{y}'$. Under the strong RSA assumption this is infeasible. The third case in the traceability experiment never occurs, since there is at most one group manager M_β such that $(g'_l)^{x_\beta} = y'_l$.

Anonymity. Given an adversary that breaks anonymity, we construct a machine A' that simulates the anonymity experiment to the adversary and breaks the decision Diffie-Hellman assumption by breaking the semantic security or the anonymity of the ElGamal encryption scheme. This can be explained informally as follows. A' first guesses which two signers $S_{\alpha^{(0)}}$ and $S_{\alpha^{(1)}}$ the adversary will choose. With non-negligible probability the guess is correct. The two paths to the root from $\alpha^{(0)}$ and $\alpha^{(1)}$ are identical up to their least common ancestor, where they split. Taking part in an attack against the ElGamal encryption scheme, A'

inserts the ElGamal keys as keys for the group managers below the split. Since the commitments contained in a signature are statistically hiding, the distributions of signatures computed by $S_{\alpha^{(0)}}$ and $S_{\alpha^{(1)}}$ differ only in what chain of identities is encrypted and which keys are used. Guessing the signer with non-negligible probability implies breaking either the anonymity or the semantic security of the the ElGamal cryptosystem, which in turn can be used to break the DDH assumption.

Let us now describe how to simulate the opening oracle for the group managers below the split, for which the secret keys are not known to A', using the ciphertexts (u, v) and (u_i, v_i). The ciphertext (u, v) contains all the information needed to answer a trusted opening query. Thus, we are implicitly using the Naor-Yung double ciphertext trick [70] here to simulate the trusted opening oracle. Similarly, the ciphertexts (u_i, v_i) contain all the information needed to simulate the optimistic opening oracle.

Another, more intuitive, way to understand the need for the (u_i, v_i) ciphertexts is that otherwise an adversary could ask an opening query using a signature σ formed using *any* public key (g_i', y_i'). This would make it infeasible to simulate the optimistic opening algorithm correctly on queries (β, m, σ) with $\mathsf{level}_T(\beta) = i$ without the secret key x_β, i.e., a contradiction could not be reached.

The Full Proof

The security of the scheme follows from Lemma 11.3.1 and Lemma 11.3.3 below.

Lemma 11.3.1. *The optimistic hierarchical group signature scheme $\mathcal{HGS}$ has traceability under the strong RSA assumption in the random oracle model.*

Proof. Consider any adversary A and polynomial size tree T. We describe a reduction such that if $\Pr[\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T) = 1]$ is negligible, i.e., A breaks traceability, then there exists an adversary A_{rsa} that breaks the strong RSA assumption. We first modify the traceability experiment using "game hopping" and then describe the adversary A_{rsa}.

Denote by E_0 the traceability experiment $\mathbf{Exp}^{\mathsf{trace}}_{\mathcal{HGS},A}(\kappa, T)$. Denote by E_1 the experiment E_0, except that the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-oracle is modified as follows:

1. If not all keys x_α for $\alpha \in \mathcal{V}(T)$ are distinct, then output *collision*.
2. When a signature is computed, the values $\mathbf{u}, \mathbf{v} \in SQ_{\mathbf{N}}$ are chosen randomly.
3. Instead of computing $\pi = P_{\mathrm{hgs}}^{\mathsf{O}(m,\cdot)}\big((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), (r, (y_{\alpha_i}, r_i, r_i', r_i'', e_{\alpha_i}, t_i)_i, t, t')\big)$, a random challenge $c \in [0, 2^{\kappa_c} - 1]$ is chosen and the special statistical zero-knowledge simulator is invoked on input $(s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}))$ and c to produce a transcript $\pi = (C, c, d)$. If the random oracle has been queried on $((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), C)$ the experiment outputs *collision* and otherwise it defines $\mathsf{O}((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), C) = c$ and continues the simulation.

Claim 1. $|\Pr[E_0 = 1] - \Pr[E_1 = 1]| < \varepsilon$ for some negligible function ε of κ.

Proof. We clearly have $\Pr[E_0 = \textit{collision}] = 0$. We also have that $\Pr[E_1 = \textit{collision}]$ is negligible, since $\mathbf{v} \in SQ_{\mathbf{N}}$ and $x_\alpha \in \mathbb{Z}_Q$ are chosen randomly. (In fact other elements are chosen randomly as well.) Finally the statistical zero-knowledge property of $(P_{\mathrm{hgs}}, V_{\mathrm{hgs}})$ implies that $|\Pr[E_1 = 1 \mid E_1 \neq \textit{collision}] - \Pr[E_0 = 1]|$ is negligible. □

Denote by E_2 the experiment E_1, except that it chooses $\gamma \in \mathcal{L}(T) \cup \{\textit{nosigner}\}$ and $\mathbf{y}' \in SQ_{\mathbf{N}}$ randomly and defines $\mathbf{y}$ as follows

$$\mathbf{y} \leftarrow (\mathbf{y}')^{\prod_{\alpha \in \mathcal{L}(T) \setminus \{\gamma\}} p_\alpha} .$$

The roots $\mathbf{z}_\alpha$ for $\alpha \in \mathcal{L}(T)$ are left undefined to start with. If A requests $\textit{hsk}(\alpha)$ then E_2 does the following. If $\alpha = \gamma$, then it returns *badguess*. If $\alpha \in \mathcal{L}(T) \setminus \{\gamma\}$, then it computes $\mathbf{z}_\alpha = \mathbf{y}^{1/p_\alpha}$ as $\mathbf{z}_\alpha = (\mathbf{y}')^{\prod_{\gamma \in \mathcal{L}(T) \setminus \{\alpha\}} p_\gamma}$ and returns $\mathbf{z}_\alpha$. If $\alpha \notin \mathcal{L}(T)$ then it returns $\perp$.

After A has output its signature, define $D_{x_{\mathfrak{T}}}(u_{\delta-1}, v_{\delta-1}) = h^{e_{\alpha_\delta}}$ where α_δ is set equal $\perp$ if $\alpha_\delta \notin \mathcal{L}(T)$. If $\gamma = \textit{nosigner}$ and $\alpha_\delta \in \mathcal{L}(T)$, or if $\gamma \in \mathcal{L}(T)$ and $\gamma \neq \alpha_\delta$, then it return *badguess*.

Claim 2. $\Pr[E_2 = 1 \mid E_2 \neq \textit{badguess}] = \Pr[E_1 = 1]$ and $\Pr[E_2 \neq \textit{badguess}] \leq 1/(|\mathcal{L}(T)| + 1)$.

Proof. Since $\prod_{\alpha \in \mathcal{L}(T) \setminus \{\gamma\}} p_\alpha$ is relatively prime to the order of $SQ_{\mathbf{N}}$, the distributions of $\mathbf{y}$ in the simulations of E_1 and E_2 are identical. This implies the first claim, since unless A requests $\mathbf{z}_\alpha$ for an $\alpha = \gamma$ the distributions of E_1 and E_2 are then identical. The second claim follows from independence of the distribution of $\mathbf{y}$ and the distribution of γ. □

Denote by $(m, (s_{out}, (\mathbf{u}, \mathbf{v}), \pi))$ the message-signature pair output by A in the experiment E_2, and write $\pi = (C, c, d)$ for the random oracle proof. Denote by E_3 the experiment E_2 except that if A never queried the random oracle O on $(m, s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}), C)$, but A's output is still a valid signature, then E_3 halts with output *pureluck*.

Claim 3. $\Pr[E_3 = 1] \geq \Pr[E_2 = 1] - 2^{-\kappa_c}$.

Proof. The probability that $\mathsf{O}(m, s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}), C)$ equal c is $2^{-\kappa_c}$ over the random choice of O conditioned on A not having asked this query before, so $\Pr[E_2 = \textit{pureluck}] \leq 2^{-\kappa_c}$. The claim then follows from the union bound. Note that if the output of A is a forgery, then the experiment has not asked this question either so we may think of the output of O as undefined until the verification of the experiment is executed. □

We now construct a machine P^* that takes input $(\mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}')$, chooses a common input, and then interacts with the honest verifier V_{hgs} on that common input. Note that without loss we may assume that A never asks the same query twice in the simulation of E_3, since it can keep all its previous queries and the corresponding answers in a table. Denote by Q_A an upper bound on the number of queries A makes to the random oracle O in the simulation of E_3. The prover P^* simulates E_3 except that it chooses an integer $0 < l \leq Q_A$ randomly and instead of simulating the random oracle O on the lth query $query_l = (m_l, s_{com}, s_{out_l}, (\mathbf{u}_l, \mathbf{v}_l), C_l)$, it first outputs $(s_{com}, s_{out,l}, (\mathbf{u}_l, \mathbf{v}_l))$ as the common input. Then it outputs C_l as its first message in a protocol executed on this common input and waits for a challenge c from V_{hgs} and defines $\mathsf{O}(query_l) = c$. Finally, when the simulation of E_3 ends, it replies *failed* to V_{hgs} if the result is 0. Otherwise it interprets the signature and message pair output by A on the form $(m, (s_{out}, (\mathbf{u}, \mathbf{v}), \pi))$ with $\pi = (C, c, d)$, and replies d to V_{hgs}. Denote the list of input RSA parameters by $\mathbf{\Gamma} = (\mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}')$.

Claim 4. $\Pr[\langle V_{\mathrm{hgs}}(s), P^*(r, \mathbf{\Gamma})\rangle = 1] \geq \Pr[E_3 = 1]/Q_A$, where the probability is taken over $\mathbf{\Gamma}$ and the internal randomness s and r of P^* and V_{hgs} respectively.

Proof. The claim is implied by the following observations:

1. The simulated experiment E_3 never outputs 1 if the signature output by A in the simulation was received from the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-oracle. This is easy to see from the description of the experiment.

2. If in the simulation of P^* the events $E_3 = 1$ and $query_l = (m, s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}), C)$ occurs, then the verifier accepts,

3. Conditioned on the event $E_3 = 1$ in the simulation carried out by P^* the probability that $query_l = (m, s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}), C)$ is at least $1/Q_A$, since l is independently distributed to all intermediate values of E_3.

□

Denote by r the internal random tape of P^*. Define $\phi = \Pr_{s,r,\mathbf{\Gamma}}[\langle V_{\mathrm{hgs}}(s), P^*(r, \mathbf{\Gamma})\rangle = 1]$ and let S_{good} be the set of pairs $(\mathbf{\Gamma}, r)$ such that $\Pr_s[\langle V_{\mathrm{hgs}}(s), P^*(r, \mathbf{\Gamma})\rangle = 1] \geq \frac{1}{2}\phi$.

Claim 5. $\phi \geq \frac{1}{Q_A}\left(\frac{1}{|\mathcal{L}(T)|}(\Pr[E_0 = 1] - \varepsilon) - 2^{-\kappa_c}\right)$ and $\Pr_{r,\mathbf{\Gamma}}[(r, \mathbf{\Gamma}) \in S_{good}] \geq \frac{1}{2}\phi$.

Proof. The first claim follows from the previous claims. The second follows by a simple averaging argument. □

Denote by A_{rsa} the algorithm that on input $\mathbf{\Gamma}$ simulates P^* on input $(r, \mathbf{\Gamma})$ using a random tape r, and then invokes the knowledge extractor on P^* and the common input $(s_{com}, s_{out,l}, (\mathbf{u}_l, \mathbf{v}_l))$ it generates. We now consider only executions where $(r, \mathbf{\Gamma}) \in S_{good}$. By definition the knowledge extractor runs in expected time $p'(\kappa)/(\phi/2 - \varepsilon)$ for some polynomial $p'(\kappa)$ and negligible function ε, and extracts a witness w such that $((s_{com}, s_{out,l}, (\mathbf{u}_l, \mathbf{v}_l)), w) \in \mathcal{R}_{\mathsf{HGSig}} \vee \mathcal{R}_{\mathrm{SRSA}}$. (Here we abuse notation and assume that the components in the instance are ordered such that it starts with the Fujisaki-Okamoto parameters $(\mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y})$.) Thus, we either have directly a pair $(((\mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}), \ldots), w) \in \mathcal{R}_{\mathrm{SRSA}}$, which case the witness is simply output, or we have a witness $(r, (y_i, r_i, r_i', r_i'', e_i, t_i)_i, t, t')$ such that

$$\big((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), (r, (y_i, r_i, r_i', r_i'', e_i, t_i)_i, t, t')\big) \in \mathcal{R}_{\mathsf{HGSig}}$$

where

$$\begin{aligned} s_{com} &= (\delta, P, g, h, y_{\mathfrak{T}}, \mathbf{N}, \mathbf{g}, \mathbf{h}, \mathbf{y}) \\ s_{out} &= ((u_0', v_0'), \mathbf{c}_0, \big((g_i', y_i'), (u_i, v_i), (u_i', v_i'), (u_i'', v_i''), \mathbf{c}_i\big)_{i=1}^{\delta-1}, (u, v)) \end{aligned}$$

and such that one of the three events listed in the traceability experiment is satisfied. If $\iota_\delta(e_1, \ldots, e_\delta)$ and $\prod_{\alpha \in \mathcal{L}(T)\setminus\{\gamma\}} p_\alpha$ are relatively prime, then the adversary A_{rsa} computes a and b such that $a\iota_\delta(e_1, \ldots, e_\delta) + b \prod_{\alpha \in \mathcal{L}(T)\setminus\{\gamma\}} p_\alpha = 1$ and outputs $((\mathbf{g}')^a \mathbf{z}^b, \iota_\delta(e_1, \ldots, e_\delta))$. Note that $((\mathbf{g}')^a \mathbf{z}^b)^{\iota_\delta(e_1, \ldots, e_\delta)} = \mathbf{g}'$.

Claim 6. If the extractor finds a witness $(r, (y_i, r_i, r_i', r_i'', e_i, t_i)_i, t, t')$, then $\iota_\delta(e_1, \ldots,$ $e_\delta)$ and $\prod_{\alpha \in \mathcal{L}(T)\setminus\{\gamma\}} p_\alpha$ are relatively prime.

Proof. Each prime p_α is contained in $[2^{(\delta+1)t_M - 1} - 1, 2^{(\delta+1)t_M}]$ by construction, and by assumption $e_i \in [-\lambda/2 + 1, \lambda/2 - 1]$ so $2 < \iota_\delta(e_1, \ldots, e_\delta) < 2^{(\delta+1)t_M}$. Thus, it suffices to show that $\iota_\delta(e_1, \ldots, e_\delta) \neq p_\alpha$ for all $\gamma \neq \alpha \in \mathcal{L}(T)$.

Define the values α_i as in the traceability experiment. If $\alpha_{i+1} \notin \alpha_i$ for some $0 \leq i \leq \delta$, then the fact that $e_i \in [-\lambda/2 + 1, \lambda/2 - 1]$ and Lemma 11.1.2 imply that $\iota_\delta(e_1, \ldots, e_\delta) \neq p_\alpha$ for all $\alpha \in \mathcal{L}(T)$. Thus, we assume that $\alpha_{i+1} \in \alpha_i$ for $i = 0, \ldots, \delta - 1$. Under this assumption Event 1 can not occur, since the short identifier strings are unique on each level in the tree. Furthermore, if we define $a = D_{\log_{g_l'} y_l'}(u_l', v_l')$, then Event 3 in the experiment can only occur if $a \neq D_{x_{\mathfrak{T}}}(u_l'', v_l'')$, but then the extracted witness can not be a witness for the relation $\mathcal{R}_{\mathsf{HGSig}}$. □

We conclude from Claim 5 and Claim 6 that ϕ must be negligible, since otherwise the strong RSA assumption is broken. This implies that $\Pr[E_0 = 1]$ is negligible and the proposition holds. □

In proving security, we use the standard list generalization of the Diffie-Hellman assumption. For completeness we provide a proof.

Experiment 11.3.1 (m-List Decision Diffie-Hellman, $\mathbf{Exp}^{\mathsf{ddh-list}-b}_{G_n,m,A}(\kappa)$).

```
$g \leftarrow_R G_n$
for $i = 1, \ldots, m$ do
    $\alpha_i, \beta_i, \gamma_i \leftarrow_R \mathbb{Z}_n$
    $(X_i, Y_i, Z_i) \leftarrow (g^{\alpha_i}, g^{\beta_i}, g^{(b\gamma_i + (1-b)\alpha_i\beta_i)})$
end for
return $A(g, (X_i, Y_i, Z_i)_{i=1}^m)$
```

The advantage of the adversary is

$$\mathbf{Adv}^{\mathsf{ddh-list}}_{G_n,m,A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ddh-list}-0}_{G_n,m,A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ddh-list}-1}_{G_n,m,A}(\kappa) = 1]| \ .$$

Lemma 11.3.2. *If there exists a probabilistic T-time machine A such that the advantage $\mathbf{Adv}^{\mathsf{ddh-list}}_{G_n,m,A}(\kappa) = c$, then there exists a probabilistic T-time machine A_{ddh} with $\mathbf{Adv}^{\mathsf{ddh}}_{G_n,A}(\kappa) \geq c/m$.*

Proof. Let A be a Turing machine with advantage c in the experiment on an m-list. Let D_l be the distribution of the output of A when the input is a list with m elements of which the first l are DDH triples and the rest are random triples. By assumption $\mathsf{dist}(D_m, D_0) = c$. Therefore, by an averaging argument, there exists $t \in [0, m-1]$ such that $\mathsf{dist}(D_t, D_{t+1}) \geq c/m$.

We now construct A_{ddh}. On input (X, Y, Z) it constructs a list L where the first t elements are DDH triples, (X, Y, Z) is inserted as the $(t+1)$th triple, and the rest of the triples are random. It invokes A on input L, and outputs the answer from A_{ddh}.

If (X, Y, Z) is a random triple, then the output is distributed according to D_t, and if (X, Y, Z) is a DDH triple, the output is distributed according to D_{t+1}. Since the distance between the two distribution is at least c/m, A_{ddh} has advantage at least c/m. □

Lemma 11.3.3. *The optimistic hierarchical group signature scheme $\mathcal{HGS}$ has anonymity under the DDH assumption and the strong RSA assumption in the random oracle model.*

Proof. We proceed similarly as in the proof of the previous lemma. Consider any adversary A and polynomial size tree T. We describe a reduction such that if the advantage $\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{HGS},A}(\kappa, T)$ of the adversary in the anonymity is non-negligible, then there exists an adversary A_{ddh} that breaks the DDH assumption. We do this by step-wise modifying the anonymity experiment until it is independent of its bit parameter.

Denote by E_0^b the anonymity experiment $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},A}(\kappa, T)$. Denote by E_1^b the experiment E_0^b except that the $\mathsf{HGSig}(\cdot, T, hpk, hsk(\cdot))$-oracle is modified as follows:

1. When a signature is computed, the values $\mathbf{u}, \mathbf{v} \in SQ_{\mathbf{N}}$ are chosen randomly.
2. Instead of computing $\pi = P^{\mathsf{O}(m,\cdot)}_{\mathrm{hgs}}\big((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})),\ (r, (y_{\alpha_i},\ r_i,\ r_i',\ r_i'',\ e_{\alpha_i},\ t_i)_i,\ t, t')\big)$, a random challenge $c \in [0, 2^{\kappa_c} - 1]$ is chosen and the special statistical zero-knowledge simulator is invoked on input $(s_{com}, s_{out}, (\mathbf{u}, \mathbf{v}))$ and c to produce a transcript $\pi = (C, c, d)$. If the random oracle has been queried on $((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), C)$ the experiment outputs *collision* and otherwise it defines $\mathsf{O}((s_{com}, s_{out}, (\mathbf{u}, \mathbf{v})), C) = c$ and continues the simulation.

Claim 1. $|\Pr[E_0^b = 1] - \Pr[E_1^b = 1]| < \varepsilon_0$ for some negligible function ε_0 of κ.

Proof. This follows similarly as Claim 1 in the proof of Lemma 11.3.1 above. □

Denote by E_2 the experiment E_1 except that instead of computing (u_i, v_i) and (u, v) of the challenge signature given to A, these values are chosen randomly in G_Q.

Claim 2. $|\Pr[E_1^b = 1] - \Pr[E_2^b = 1]| < \varepsilon_1$ for some negligible function ε_1 of κ.

Proof. Denote by A_{ddh} the algorithm that takes as input a description of a group G_Q, an generator g of the group, and a triple $(\bar{y}, \bar{u}, \bar{v})$. The task is to decide if $\log_g \bar{y} \log_g \bar{u} = \log_g \bar{v}$ or not, i.e., decide if the input is a DDH-triple or not.

It simulates the experiment E_1^b, except that during key generation it sets $y = \bar{y}$, and instead of computing (u_i, v_i) and (u, v) of the challenge signature, it replaces these values by $(g^{r_i}\bar{u}^{s_i}, y^{r_i}\bar{v}^{s_i}y_{\alpha_i})$ and $(g^r\bar{u}^s, y^r\bar{v}^s h^{p_\alpha})$ respectively, where $r, r_i, s, s_i \in \mathbb{Z}_Q$ are chosen randomly. Note that this does not change the distribution of y. Furthermore, the resulting challenge signature is identically distributed to the challenge signature in the experiment E_1^b or E_2^b depending on if $(\bar{y}, \bar{u}, \bar{v})$ is a DDH triple or not. Thus, if $\Pr[E_1^b = 1] - \Pr[E_2^b = 1]|$ is non-negligible, then A_{ddh} breaks the DDH assumption. □

Consider the indices $\alpha^{(0)}$ and $\alpha^{(1)}$ chosen by A. Note that if either index is not contained in $\mathcal{L}(T)$, the experiment outputs 0. Furthermore, if B is defined as in the anonymity experiment and $B \cap \mathcal{C} \neq \emptyset$, then the experiment also outputs 0. Thus, we may assume that an adversary always outputs $\alpha^{(0)}, \alpha^{(1)} \in \mathcal{L}(T)$ and that $B \cap \mathcal{C} \neq \emptyset$. Formally, this can be seen by replacing A by a new adversary A' that encapsulates the original, satisfies our requirements and outputs the guess 0 whenever A violates our restriction.

Denote by E_3^b the experiment E_2^b except that before the simulation starts an index $\beta^{(b)} \in \mathcal{L}(T)$ is chosen randomly and if A requests $hsk(\beta^{(b)})$ or if $\alpha^{(b)} \neq \beta^{(b)}$, then the experiment halts with output *badguess*.

Claim 3. $\Pr[E_3^b = 1 \mid E_3^b \neq \mathit{badguess}] = \Pr[E_2^b = 1]$ and $\Pr[E_3^b \neq \mathit{badguess}] \leq 1/|\mathcal{L}(T)|$.

Proof. The first claim follows by the independent choice of $\beta^{(b)}$. The second claim follows from our assumption on the adversary A above, and the independence of $\beta^{(b)}$. □

Denote by E_4^b the experiment E_3^b except that the way oracle queries are answered is changed as follows. The element y is generated as $y = g^x$ for a random $x \in G_Q$, where x is stored by the simulator.

Given a query (β, m, σ) the $\mathsf{HGTrustOpen}(T, hpk, sk_{\mathfrak{T}}, \cdot, \cdot, \cdot)$-oracle never reads $sk_{\mathfrak{T}}$. This means that it can not decrypt (u_l, v_l). Instead it computes $a' = D_x(u, v)$ and if possible the list $(e_1, \ldots, e_\delta)$ such that $a' = h^{\iota_\delta(e_1,\ldots,e_\delta)}$. Finally, it defines $a = h^{e_l}$.

Given a query (β, m, σ) to the $\mathsf{HGOptOpen}(T, hpk, hsk(\cdot), \cdot, \cdot)$-oracle defines a above, but it also computes $y_i = D_x(u_i, v_i)$. Then if $l > 0$ and $y_i \neq y_\beta$ it returns $\perp$ and otherwise it proceeds as in the original definition.

Claim 4. $|\Pr[E_3^b = 1] - \Pr[E_4^b = 1]| < \varepsilon_3$ for some negligible function ε_3 of κ.

Proof. Assume there is a non-negligible difference between E_3^b and E_4^b. We follow the paradigm of Claim 1 of the proof of Lemma 11.3.1 and construct an interactive proof of knowledge. Let $q(\kappa)$ be a polynomial which upper-bounds the number of calls to the oracles. First $l \leftarrow_R [q(\kappa)$ is chosen randomly. Informally we guess that the lth oracle call will yield an incorrect answer. For the lth oracle call, we proceed as in Claim 1 and construct a prover as follows. It outputs the lth query as theorem to prove, and after the challenge c is received from the verifier, the random oracle is programmed to return c when queried on the lth query. Thus we have constructed an interactive prover.

If the lth query is answered incorrectly in the simulation experiment, then the corresponding interactive prover will prove a false statement. By the soundness of the proof of knowledge such a prover can be used to break the strong RSA assumption of the decision Diffie-Hellman. Since with probability at least $1/p(\kappa)$ the query is indeed malformed, the claim follows. □

At this point we are almost done. The only difference between the experiments E_4^0 and E_4^1 is how the ciphertexts (g_i', y_i'), (u_i', u_i') and (u_i'', u_i'') in the challenge signature are formed.

Define l_a to be the depth of the least common ancestor of $\alpha^{(0)}$ and $\alpha^{(1)}$. Denote by E_5^b the experiment E_4^b except that $g_i', y_i', u_i', v_i' \in G_Q$ are chosen randomly. Define $\alpha_i^{(b)}$ as in the anonymity experiment.

Claim 5. $|\Pr[E_4^b = 1] - \Pr[E_5^b = 1]| < \varepsilon_4$ for some negligible function ε_4 of κ.

Proof. Denote by A_{ddh} the adversary that accepts a list $((\bar{y}_i, \bar{u}_i, \bar{v}_i))_{l_a}^{\delta-1}$ as input and simulates E_4^b except that it defines $y_{\alpha_i^{(b)}} = \bar{y}_i$, $(g_i', y_i') = (\bar{u}_i, \bar{v}_i)$, and $(u_i', v_i') = (g^{r_i}\bar{u}_i^{s_i}, y_{\alpha_i^{(b)}}^{r_i}\bar{v}_i^{s_i}h^{e_{\alpha_{i+1}^{(b)}}})$, where $r_i, s_i \in \mathbb{Z}_Q$ are randomly chosen. Note that if the input list is a list of DDH triples, then the simulation is identically distributed to E_4^b and otherwise it is identically distributed to E_5^b. From the generalized DDH assumption, Lemma 11.3.2, we conclude that the claim holds. □

Denote by E_5^b the experiment E_4^b except that $u_i'', v_i'' \in G_Q$ are chosen randomly.

Claim 6. $|\Pr[E_5^b = 1] - \Pr[E_6^b = 1]| < \varepsilon_5$ for some negligible function ε_5 of κ.

Proof. Denote by A_{ddh} the adversary that accepts $(\bar{y}, \bar{u}, \bar{v})$ and simulates E_5^b except that it defines $y_{\mathfrak{T}} = \bar{y}$ and $(u_i'', v_i'') = (g^{r_i}\bar{u}^{s_i}, y_{\mathfrak{T}}^{r_i}\bar{v}_i^{s_i}h^{e_{\alpha_{i+1}^{(b)}}})$, where $r_i, s_i \in \mathbb{Z}_Q$ are randomly chosen. Again, if the input is a DDH triple, then the simulation is identically distributed to E_5^b and otherwise it is identically distributed to E_5^b. We conclude from the DDH assumption that the claim holds. □

By construction the experiments E_6^0 and E_6^1 are identical. Thus, the above claims imply the lemma. □

11.4 Proof of Knowledge

Here we give the full details of the proof of knowledge used to generate the signature.

A Computationally Convincing Proof of Knowledge

The proof of knowledge we will describe is not a proof of knowledge in the strict sense. It is possible for a prover to produce proofs such that no algorithm is able to extract the private input. However, if such a prover exists it can be used to break the strong RSA assumption. The proofs can be seen as a proof of either the private input *or* an RSA root. Such proofs are called *computationally convincing proofs of knowledge* and are described in Section 8.5. Under the strong RSA assumption they can be treated as a proof of knowledge if the prover is polynomially bounded.

More precisely, we will show that a prover which constructs a valid proof that cannot be extracted, the prover can be used to compute one of the following:

1. Integers $x_1 \neq 0$ and x_2, x_3, at least one of which non-zero, and $\mathbf{b} \in SQ_{\mathbf{N}}$ such that $\mathbf{b}^{x_1} = \mathbf{g}^{x_2}\mathbf{h}^{x_3}$ and x_1 does not divide both x_2 and x_3.
2. Integers x_1, x_2, at least one non-zero, such that $\mathbf{g}^{x_1}\mathbf{h}^{x_2} = 1$.

Both cases imply that the strong RSA assumption is broken.

Protocol 11.4.1.
COMMON INPUT: $g, h, y \in G_Q, \mathbf{g}, \mathbf{h} \in SQ_{\mathbf{N}}, (u, v) \in G_Q^2, \mathbf{c} \in SQ_{\mathbf{N}}$.
PRIVATE INPUT: $\rho \in \mathbb{Z}_Q, \tau \in [0, 2^{\kappa_r}\mathbf{N} - 1], \varepsilon \in [0, 2^{t_M} - 1]$ such that

$$\begin{aligned} (u, v) &= E^{\mathsf{elg}}_{(g,y)}(h^{\varepsilon}, \rho) \\ \mathbf{c} &= \mathbf{g}^{\tau}\mathbf{h}^{\varepsilon} \end{aligned}$$

INTERVALS SHOWN: $\varepsilon \in [-2^{\kappa_c+\kappa_r+t_M} + 1, 2^{\kappa_c+\kappa_r+t_M} - 1]$.

1. The prover randomly selects $a \leftarrow_R \mathbb{Z}_Q$, $b_1 \leftarrow_R [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$, $b_2 \leftarrow_R [0, 2^{\kappa_c+\kappa_r+t_M} - 1]$ and computes

$$\begin{aligned} (A_1, A_2) &\leftarrow (g^a, y^a h^{b_2}) \\ \mathbf{A} &\leftarrow \mathbf{g}^{b_1}\mathbf{h}^{b_2} \ . \end{aligned}$$

   The prover hands $(A_1, A_2, \mathbf{A})$ to the verifier.

2. The verifier randomly selects $c \leftarrow_R [0, 2^{\kappa_c} - 1]$ and hands to the prover.

3. The prover computes

$$d_1 \leftarrow c\rho + a_1 \bmod Q \tag{11.1}$$

$$d_2 \leftarrow c\tau + a_2 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} \tag{11.2}$$

$$d_3 \leftarrow c\varepsilon + a_3 \bmod 2^{\kappa_c+\kappa_r+t_M} \tag{11.3}$$

   and hands (d_1, d_2, d_3) to the verifier.

4. The verifier accepts if

$$
\begin{aligned}
(A_1 u^c, A_2 v^c) &\overset{?}{=} (g^{d_1}, y^{d_1} h^{d_3}) && (11.4)\\
\mathbf{A}\mathbf{c}^c &\overset{?}{=} \mathbf{g}^{d_2}\mathbf{h}^{d_3} && (11.5)
\end{aligned}
$$

Lemma 11.4.1. *Protocol 11.4.1 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. It is easily checked that the verifier accepts unless there is a reduction in the computation of d_2 or d_3. The probability of a reduction in each computation is $2^{-\kappa_r}$, and the union bound gives a probability of $2 \cdot 2^{-\kappa_r}$ for a reduction in either d_2 or d_3. Since this negligible, the protocol has overwhelming completeness.

We now show special soundness. Given two accepting executions (A_1, A_2, $\mathbf{A}$, c, d_1, d_2, d_3) and (A_1, A_2, $\mathbf{A}$, c', d_1', d_2', d_3') where $c \neq c'$ the extractor can solve Equations (11.1 – 11.3) for $\rho^* = \frac{d_1 - d_1'}{c - c'} \in \mathbb{Z}_Q$, $\tau^* = \frac{d_2 - d_2'}{c - c'} \in \mathbb{Z}$, $\varepsilon^* = \frac{d_3 - d_3'}{c - c'} \in [-2^{\kappa_c + \kappa_r + t_M} + 1, 2^{\kappa_c + \kappa_r + t_M} - 1]$. The interval constraint on ε^* holds since $|d_3 - d_3'| < 2^{\kappa_c + \kappa_r + t_M}$.

Since the computation of τ^* and ε^* are over the integers, one must show that both $(d_2 - d_2')$ and $(d_3' - d_3)$ are divisible by $(c - c')$. Assume this is not the case. Then Case 1 holds by setting $\mathbf{b} = \mathbf{c}$ and $x_1 = c - c'$, $x_2 = d_2 - d_2'$, and $x_3 = d_3 - d_3'$. It follows that either the extractor produces ρ, τ, ε such that $(u, v) = E^{\mathsf{elg}}_{(g,y)}(h^{\varepsilon^*}, \rho^*)$ and $\mathbf{c} = \mathbf{g}^{\tau^*}\mathbf{h}^{\varepsilon^*}$ or it breaks the strong RSA assumption.

Showing special zero-knowledge is straight-forward. Given c the simulator randomly selects d_1, d_2, d_3 and computes $A_1, A_2, \mathbf{A}$ from Equations (11.4 – 11.5). This gives a distribution which is identical to that of an honest execution. □

Protocol 11.4.2.
COMMON INPUT: $g, h, y, z \in G_Q, \mathbf{g}, \mathbf{h} \in SQ_{\mathbf{N}}$, $(g', y') \in G_Q^2$, $(u, v) \in G_Q^2$, $(u', v') \in G_Q^2$, $(u'', v'') \in G_Q^2$, $\mathbf{c} \in SQ_{\mathbf{N}}$.
PRIVATE INPUT: $\rho, \rho', \rho'' \in \mathbb{Z}_Q$, $\tau \in [0, 2^{\kappa_r}\mathbf{N} - 1]$, $\varepsilon \in [0, 2^{t_M} - 1]$, $\upsilon \in G_Q$ such that

$$
\begin{aligned}
(g', y') &= (g^{\rho'}, \upsilon^{\rho'})\\
(u, v) &= E^{\mathsf{elg}}_{(g,y)}(\upsilon, \rho)\\
(u', v') &= E^{\mathsf{elg}}_{(g',y')}(h^{\varepsilon}, \rho'')\\
(u'', v'') &= E^{\mathsf{elg}}_{(g,z)}(h^{\varepsilon}, \rho'')\\
\mathbf{c} &= \mathbf{g}^{\tau}\mathbf{h}^{\varepsilon}
\end{aligned}
$$

INTERVALS SHOWN: $\varepsilon \in [-2^{\kappa_c + \kappa_r + t_M} + 1, 2^{\kappa_c + \kappa_r + t_M} - 1]$.

1. The prover selects $s \leftarrow_R \mathbb{Z}_Q$ and computes $B_1 \leftarrow g^s\,(\upsilon / y^r)$.

2. The prover selects $a_1, a_2, a_3, a_4 \leftarrow_R \mathbb{Z}_Q$, $b_1 \leftarrow_R [0, 2^{\kappa_c + 2\kappa_r}\mathbf{N} - 1]$, $b_2 \leftarrow_R [0, 2^{\kappa_c+\kappa_r+t_M} - 1]$ and computes

$$\begin{aligned} A_1 &\leftarrow g^{a_1} y^{-a_2} \\ A_2 &\leftarrow g^{a_1}(y')^{a_3} \\ A_3 &\leftarrow g^{a_2} \\ A_4 &\leftarrow (g')^{a_3} \\ (A_5, A_6) &\leftarrow \big((g')^{a_4}, (y')^{a_4} h^{b_2}\big) \\ (A_7, A_8) &\leftarrow \big(g^{a_4}, z^{a_4} h^{b_2}\big) \\ \mathbf{A} &\leftarrow \mathbf{g}^{b_1}\mathbf{h}^{b_2} \ . \end{aligned}$$

   It hands $B_1, B_2, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, \mathbf{A}$ to the verifier.

3. The verifier chooses $c \leftarrow_R [0, 2^{\kappa_c} - 1]$ and hands to the prover.

4. The prover computes

$$\begin{aligned} d_1 &\leftarrow cs + a_1 \bmod Q && (11.6) \\ d_2 &\leftarrow c\rho + a_2 \bmod Q && (11.7) \\ d_3 &\leftarrow c/\rho' + a_3 \bmod Q && (11.8) \\ d_4 &\leftarrow c\rho'' + a_4 \bmod Q && (11.9) \\ d_5 &\leftarrow c\tau + b_1 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.10) \\ d_6 &\leftarrow c\varepsilon + b_2 \bmod 2^{\kappa_c+\kappa_r+t_M} && (11.11) \end{aligned}$$

   and hands $(d_1, d_2, d_3, d_4, d_5, d_6, d_7)$ to the verifier.

5. The verifier checks that

$$\begin{aligned} A_1(B_1/v)^c &\stackrel{?}{=} g^{d_1} y^{-d_2} && (11.12) \\ A_2 B_1^c &\stackrel{?}{=} g^{d_1}(y')^{d_3} && (11.13) \\ A_3 u^c &\stackrel{?}{=} g^{d_2} && (11.14) \\ A_4 g^c &\stackrel{?}{=} (g')^{d_3} && (11.15) \\ A_5 (u')^c &\stackrel{?}{=} (g')^{d_4} && (11.16) \\ A_6 (v')^c &\stackrel{?}{=} (y')^{d_4} h^{d_6} && (11.17) \\ A_7 (u'')^c &\stackrel{?}{=} g^{d_4} && (11.18) \\ A_8 (v'')^c &\stackrel{?}{=} z^{d_4} h^{d_6} && (11.19) \\ \mathbf{A}\mathbf{c}^c &\stackrel{?}{=} \mathbf{g}^{d_5}\mathbf{h}^{d_6} && (11.20) \end{aligned}$$

   and accepts if all equalities hold.

Lemma 11.4.2. *Protocol 11.4.2 is a* $[0, 2^{\kappa_c} - 1]$-Σ*-protocol.*

Proof. The verifier accepts if there is no reduction in the computation of d_5 and d_6. Hence the protocol has overwhelming completeness.

Now consider the special soundness of the protocol. Assume the extractor is given the output of two accepting executions (B_1, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, $\mathbf{A}$, c, d_1, d_2, d_3, d_4, d_5, d_6) and (B_1, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, $\mathbf{A}$, c', d_1', d_2', d_3', d_4', d_5', d_6'). The extractor solves Equations (11.6 – 11.11) for s^*, ρ^*, ρ'^*, ρ''^*, τ^*, ε^*.

From Equation (11.12) it follows that $B_1 = g^{\frac{d_1 - d_1'}{c-c'}} y^{-\frac{d_2-d_2'}{c-c'}} v = g^{s^*} y^{\rho^*}$, and from Equation (11.13) that $B_1 = g^{\frac{d_1-d_1'}{c-c'}} (y')^{\frac{d_3-d_3'}{c-c'}} = g^{s^*} (y')^{\rho'^*}$, which implies $\left(\frac{v}{y^{\rho^*}}\right)^{\rho'^*} = y'$. From Equations (11.14) and (11.15) it follows that $g' = g^{\rho'^*}$ and $u = g^{\rho^*}$. By setting $v^* = \frac{v}{y^{\rho^*}}$ it holds that $y' = v^{*\rho'^*}$ and $v = y^{\rho^*} v^*$. It is straight-forward to show that the extracted values satisfy the remaining equalities. As in the previous protocol it can also be seen that $\varepsilon^* \in [-2^{\kappa_c+\kappa_r+t_M} + 1, 2^{\kappa_c+\kappa_r+t_M} - 1]$, and that either the computations of τ^* and ε^* yield integers, or Case 1 holds.

To show special zero-knowledge, we describe how to build a simulator which is given c as input. It chooses B_1 as well as d_1, d_2, d_3, d_4, d_5, d_6 at random and computes A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, $\mathbf{A}$ from Equations (11.12 – 11.20). This gives a distribution identical to that of an honest execution. □

Protocol 11.4.3.
Common Input: $g, h, y \in G_Q, \mathbf{g}, \mathbf{h} \in SQ_{\mathbf{N}}, (u, v) \in G_Q^2, (\mathbf{u}, \mathbf{v}) \in SQ_{\mathbf{N}}^2, \mathbf{c} \in SQ_{\mathbf{N}}$.
Private Input: $\rho \in \mathbb{Z}_Q, \sigma, \tau, \tau' \in [0, 2^{\kappa_r}\mathbf{N} - 1], \pi \in [0, 2^{(t_P+t_M)(\delta-1)} - 1]$ such that

$$(u, v) = E^{\mathsf{elg}}_{(g,y)}(h^\pi, \rho) \tag{11.21}$$
$$\mathbf{c} = \mathbf{g}^\sigma \mathbf{h}^\pi \tag{11.22}$$
$$(\mathbf{u}, \mathbf{v}) = (\mathbf{g}^\tau \mathbf{h}^{\tau'}, \mathbf{h}^\tau \mathbf{g}^{1/\pi}) \tag{11.23}$$

In the description of this protocol we let $t_\pi = (t_P + t_M)(\delta - 1)$.

1. The prover selects $s, t \leftarrow_R [0, 2^{\kappa_r}\mathbf{N} - 1]$ and sets
$$(\mathbf{B}_1, \mathbf{B}_2) \leftarrow (\mathbf{g}^s \mathbf{h}^t \mathbf{u}^\pi, \mathbf{h}^s \mathbf{v}^\pi)$$
2. The prover selects $a_1 \leftarrow_R \mathbb{Z}_Q$, $a_2 \leftarrow_R [0, 2^{\kappa_c+\kappa_r+t_\pi} - 1]$, $a_3, a_4 \leftarrow_R [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$, $a_5, a_6 \leftarrow_R [0, 2^{\kappa_c+2\kappa_r+t_\pi}\mathbf{N} - 1]$, $a_7, a_8, a_9 \leftarrow_R [0, 2^{\kappa_c+2\kappa_r}\mathbf{N} - 1]$ and computes
$$(A_1, A_2) \leftarrow (g^{a_1}, y^{a_1} h^{a_2}) \tag{11.24}$$
$$(\mathbf{A}_3, \mathbf{A}_4) \leftarrow (\mathbf{g}^{a_3} \mathbf{h}^{a_4} \mathbf{u}^{a_2}, \mathbf{h}^{a_3} \mathbf{v}^{a_2}) \tag{11.25}$$
$$(\mathbf{A}_5, \mathbf{A}_6) \leftarrow (\mathbf{g}^{a_5} \mathbf{h}^{a_6}, \mathbf{h}^{a_5}) \tag{11.26}$$
$$\mathbf{A}_7 \leftarrow \mathbf{g}^{a_7} \mathbf{h}^{a_8} \tag{11.27}$$
$$\mathbf{A}_8 \leftarrow \mathbf{h}^{a_9} \mathbf{g}^{a_2}\ . \tag{11.28}$$

The prover hands $(\mathbf{B}_1, \mathbf{B}_2, A_1, A_2, \mathbf{A}_3, \mathbf{A}_4, \mathbf{A}_5, \mathbf{A}_6, \mathbf{A}_7, \mathbf{A}_8)$ to the verifier.

3. The verifier selects $c \leftarrow_R [0, 2^{\kappa_c} - 1]$ and hands to the verifier.
4. The prover computes

$$
\begin{aligned}
d_1 &\leftarrow c\rho + a_1 \bmod Q && (11.29)\\
d_2 &\leftarrow c\pi + a_2 \bmod 2^{\kappa_c+\kappa_r+t_\pi} && (11.30)\\
d_3 &\leftarrow cs + a_3 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.31)\\
d_4 &\leftarrow ct + a_4 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.32)\\
d_5 &\leftarrow c(\pi\tau + s) + a_5 \bmod 2^{\kappa_c+2\kappa_r+t_\pi}\mathbf{N} && (11.33)\\
d_6 &\leftarrow c(\pi\tau' + t) + a_6 \bmod 2^{\kappa_c+2\kappa_r+t_\pi}\mathbf{N} && (11.34)\\
d_7 &\leftarrow c\tau + a_7 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.35)\\
d_8 &\leftarrow c\tau' + a_8 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.36)\\
d_9 &\leftarrow c\sigma + a_9 \bmod 2^{\kappa_c+2\kappa_r}\mathbf{N} && (11.37)
\end{aligned}
$$

and hands $(d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9)$ to the verifier.

5. The verifier accepts if

$$
\begin{aligned}
(u^c A_1, v^c A_2) &\stackrel{?}{=} (g^{d_1}, y^{d_1} h^{d_2}) && (11.38)\\
(\mathbf{B}_1^c \mathbf{A}_3, \mathbf{B}_2^c \mathbf{A}_4) &\stackrel{?}{=} (\mathbf{g}^{d_3}\mathbf{h}^{d_4}\mathbf{u}^{d_2}, \mathbf{h}^{d_3}\mathbf{v}^{d_2}) && (11.39)\\
(\mathbf{B}_1^c \mathbf{A}_5, (\mathbf{B}_2/\mathbf{g})^c \mathbf{A}_6) &\stackrel{?}{=} (\mathbf{g}^{d_5}\mathbf{h}^{d_6}, \mathbf{h}^{d_5}) && (11.40)\\
\mathbf{u}^c \mathbf{A}_7 &\stackrel{?}{=} \mathbf{g}^{d_7}\mathbf{h}^{d_8} && (11.41)\\
\mathbf{c}^c \mathbf{A}_8 &\stackrel{?}{=} \mathbf{h}^{d_9}\mathbf{g}^{d_2} && (11.42)
\end{aligned}
$$

Lemma 11.4.3. *Protocol 11.4.3 is a* $[0, 2^{\kappa_c} - 1]$*-*Σ*-protocol.*

Proof. The verifier accepts if there is no reduction is the computation of d_2, d_3, d_4, d_5, d_6, d_7, d_8, or d_9. Such a reduction happens with negligible probability, and hence the protocol has overwhelming completeness.

We now show special soundness. Assume an extractor is given the transcript of two accepting executions $(\mathbf{B}_1, \mathbf{B}_2, A_1, A_2, \mathbf{A}_3, \mathbf{A}_4, \mathbf{A}_5, \mathbf{A}_6, \mathbf{A}_7, \mathbf{A}_8, c, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9)$ and $(\mathbf{B}_1, \mathbf{B}_2, A_1, A_2, \mathbf{A}_3, \mathbf{A}_4, \mathbf{A}_5, \mathbf{A}_6, \mathbf{A}_7, \mathbf{A}_8, c', d_1', d_2', d_3', d_4', d_5', d_6', d_7', d_8', d_9')$ where $c \neq c'$.

The extractor first computes $\tau^* \leftarrow \frac{d_7 - d_7'}{c - c'}$, $\tau'^* \leftarrow \frac{d_8 - d_8'}{c - c'}$ such that

$$\mathbf{u} = \mathbf{g}^{\tau^*}\mathbf{h}^{\tau'^*} \ .$$

It extracts $s^* = \frac{d_3 - d_3'}{c - c'}$, $t^* = \frac{d_4 - d_4'}{c - c'}$, $\pi^* = \frac{d_2 - d_2'}{c - c'}$ such that

$$(\mathbf{B}_1, \mathbf{B}_2) = (\mathbf{g}^{s^*}\mathbf{h}^{t^*}\mathbf{u}^{\pi^*}, \mathbf{h}^{s^*}\mathbf{v}^{\pi^*}) \ .$$

It also computes $\gamma^* = \frac{d_5 - d_5'}{c - c'}$, $\gamma'^* = \frac{d_6 - d_6'}{c - c'}$ such that

$$(\mathbf{B}_1, \mathbf{B}_2) = (\mathbf{g}^{\gamma^*}\mathbf{h}^{\gamma'^*}, \mathbf{h}^{\gamma^*}\mathbf{g}) \ .$$

We now show that $\gamma^* = \pi^*\tau^* + s^*$ and $\gamma'^* = \pi^*\tau'^* + t^*$. Assume this is not the case. Since $\mathbf{B}_1 = \mathbf{g}^{s^*}\mathbf{h}^{t^*}\mathbf{u}^{\pi^*} = \mathbf{g}^{\pi^*\tau^*+s^*}\mathbf{h}^{\pi^*\tau'^*+t^*}$ this would give two representations of $\mathbf{B}_1$ and fulfill Case 2.

Next we show that $\mathbf{v}$ is a commitment to a root of $\mathbf{g}$. We have $\mathbf{B}_2 = \mathbf{h}^{s^*}\mathbf{v}^{\pi^*}$ and $\mathbf{B}_2 = \mathbf{h}^{\gamma^*}\mathbf{g}$, which gives $\mathbf{v}^{\pi^*} = \mathbf{h}^{\pi^*\tau^*}\mathbf{g}$. This implies $\mathbf{v} = \mathbf{h}^{\tau^*}\mathbf{g}^{1/\pi^*}$.

It is easily verified that $\rho^* = \frac{d_1 - d_1'}{c - c'}$ satisfies $(u, v) = E^{\mathsf{elg}}_{(g,y)}(h^{\pi^*}, \rho^*)$. We conclude that the protocol is special-sound.

It remains to show that the protocol has special zero-knowledge by describing a simulator which on input c produces a transcript distributed as an honest execution of the protocol. It selects $\mathbf{B}_1, \mathbf{B}_2$ as well as $d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9$ randomly and computes $A_1, A_2, \mathbf{A}_3, \mathbf{A}_4, \mathbf{A}_5, \mathbf{A}_6, \mathbf{A}_7, \mathbf{A}_8$ from Equations (11.38 – 11.42). This gives a distribution that is identical to an honest execution. This concludes the proof. □

We are now ready to use the above subprotocols to construct the complete protocol.

Protocol 11.4.4.
COMMON INPUT: $g, h, y, y_{\alpha_0}, y_{\mathfrak{T}} \in G_Q, \mathbf{g}, \mathbf{h} \in SQ_{\mathbf{N}}, (u, v) \in G_Q^2, (\mathbf{u}, \mathbf{v}) \in SQ_{\mathbf{N}}^2, \mathbf{c} \in SQ_{\mathbf{N}}$, $(u_0', v_0') \in G_Q^2$, $\mathbf{c}_0 \in SQ_{\mathbf{N}}$, $((g_i', y_i'), (u_i, v_i), (u_i', v_i'), (u_i'', v_i''), \mathbf{c}_i) \in G_Q^8 \times SQ_{\mathbf{N}}$ for $i = 1, \ldots, \delta - 1$, $(u, v) \in G_Q^2$ and $(\mathbf{u}, \mathbf{v}) \in SQ_{\mathbf{N}}$.
PRIVATE INPUT: $\rho, \rho_i, \rho_i', \rho_i'' \in \mathbb{Z}_Q$, $\tau, \tau', \tau_i \in \mathbb{Z}$, $\varepsilon_i \in [0, 2^{t_M} - 1], \upsilon_i \in G_Q$ for $i = 1, \ldots, \delta - 1$, and $\boldsymbol{\zeta} \in SQ_{\mathbf{N}}$ such that

$$\begin{array}{rcl} (u_0', v_0') & = & E^{\mathsf{elg}}_{(g, y_{\alpha_0})}(h^{\varepsilon_1}, \rho_0'') \\ \mathbf{c}_0 & = & \mathbf{h}^{\tau_0}\mathbf{g}_0^{\varepsilon_1} \end{array}$$

$$\left(\begin{array}{rcl} (g_i', y_i') & = & (g^{\rho_i'}, \upsilon_i^{\rho_i'}) \\ (u_i, v_i) & = & E^{\mathsf{elg}}_{(g,y)}(\upsilon_i, \rho_i) \\ (u_i', v_i') & = & E^{\mathsf{elg}}_{(g_i', y_i')}(h^{\varepsilon_{i+1}}, \rho_i'') \\ (u_i'', v_i'') & = & E^{\mathsf{elg}}_{(g, y_{\mathfrak{T}})}(h^{\varepsilon_{i+1}}, \rho_i'') \\ \mathbf{c}_i & = & \mathbf{g}^{\tau_i}\mathbf{h}_i^{\varepsilon_{i+1}} \end{array}\right)_{i=1}^{\delta-1}$$

$$\begin{array}{rcl} (u, v) & = & E^{\mathsf{elg}}_{(g,y)}(h^{\pi}, \rho) \\ (\mathbf{u}, \mathbf{v}) & = & (\mathbf{g}^{\tau}\mathbf{h}^{\tau'}, \mathbf{h}^{\tau}\boldsymbol{\zeta}) \\ & \boldsymbol{\zeta}^{\pi} = \mathbf{g} & \end{array} \quad .$$

where

$$\pi = \sum_{i=0}^{\delta-2} \varepsilon_{i+1} 2^{(t_M + t_P)i} + \varepsilon_\delta 2^{(t_M + t_P)(\delta-1)}$$

INTERVALS SHOWN: $\pi \in [-2^{t_\pi}+1, 2^{t_\pi}-1]$.

The following protocols are executed in parallel, using a common challenge $c \in [0, 2^{\kappa_c}-1]$:

- Protocol 11.4.1 on common input $g, h, y_{\alpha_0}, \mathbf{g}_0, \mathbf{h}, \mathbf{c}_0$ and private input ρ_0'', τ_0, ε_1.
- Protocol 11.4.2 on common input g, h, y, $y_{\mathfrak{T}}$, $\mathbf{g}_i$, $\mathbf{h}$, (g_i', y_i'), (u_i, v_i), (u_i', v_i), (u_i'', v_i''), $\mathbf{c}_i$ and private input ρ_i, ρ_i', ρ_i'', τ_i, ε_{i+1}, v_i for $i = 1, \ldots, \delta-1$.
- Protocol 11.4.3 on common input g, h, y, $\mathbf{g}$, $\mathbf{h}$, (u, v), $(\mathbf{u}, \mathbf{v})$, $\mathbf{c} = \prod_{i=0}^{\delta-1} \mathbf{c}_i$ and private input ρ, $\sum_{i=0}^{\delta-1} \tau_i$, τ, τ', π.

Lemma 11.4.4. *Protocol 11.4.4 is a* $[0, 2^{\kappa_c}-1]$*-*Σ*-protocol.*

Proof. The protocol has overwhelming completeness since all the sub-protocols have overwhelming completeness.

To show special soundness, we begin by observing that ρ, ρ_i, ρ_i', ρ_i'', τ, τ', τ_i, v_i, ε_i can be extracted by the special soundness of the subprotocols. What remains to be shown is that π^* extracted in Protocol 11.4.3 equals the sum $\pi = \sum_{i=0}^{\delta-2} \varepsilon_{i+1} 2^{(t_M+t_P)i} + \varepsilon_\delta 2^{(t_M+t_P)(\delta-1)}$. Assume this is not the case. By construction $\mathbf{c} = \mathbf{g}^\tau \mathbf{h}^\pi$. If $\pi \neq \pi^*$, this would give two representations of $\mathbf{c}$, which fulfills Case 2. The interval constraint on π is satisfied due to do the interval constraints on ε_i imposed by the subprotocols. Hence the protocol is special-sound.

Special zero-knowledge follows since the sub-protocols have special zero-knowledge. □

11.5 Complexity Analysis

Let us now examine the number of exponentiations for a tree of depth δ. The part s_{out} requires $4 + 9(\delta-1)$ exponentiations, and the proof needs 3 exponentiations for Protocol 11.4.1, Protocol 11.4.2 needs $9(\delta-1)$ exponentiations, and Protocol 11.4.3 requires 10 exponentiations, which gives a total of $17 + 18(\delta-1)$ exponentiations for a signature, where we have counter dual-base exponentiations as a single exponentiations, but without using any other tricks. For a typical example with a three-level hierarchy, a total of 71 exponentiations are required. Since many of these exponentiations are fixed-base exponentiations, similar tricks as those mentioned in Section 10.6 can be used to reduce the running time.

As comparison the scheme in Chapter 10 requires an equivalent of about 1000 general exponentiations for a three-level hierarchy.

Chapter 12

Universally Composable Hierarchical Group Signatures

In this section we define the notion of a hierarchical group signature scheme as an ideal functionality. We relate this functionality to the definition of security given in Chapter 8, and discuss the advantages and disadvantages of casting hierarchical group signatures in the UC-framework. Then we proceed by giving a slightly weaker functionality which is securely realized by the optimistic protocol in Chapter 11.

12.1 Notation

We use a model where the ideal functionality is linked to the players through a communication network $\mathcal{C}_{\mathcal{I}}$. The communication network forwards a message m from a player P as (P, m) to the ideal functionality. When $\mathcal{C}_{\mathcal{I}}$ receives (P, m) from the functionality, it forwards the message m to player P. Except for immediate functions, defined as a message from a player P immediately followed by a response to the same player P, the ideal adversary $\mathcal{S}$ is informed of when a message is sent, but not of the content. The ideal adversary is allowed to delay the delivery of such a message, but not change its content.

12.2 Proper Hierarchical Group Signatures

The ideal functionality must capture the anonymity expected from a secure scheme. We solve this by generating signatures that do not depend on the signer, and therefore do not reveal any information about who the signer is. However, if a group manager is corrupt, the environment is able to distinguish between certain signers. Below we describe how to construct the private keys to handle this issue.

Given a map $hsk : T \to \{0,1\}^*$ and a subset $\mathcal{C}$ of T, we define two new maps $hsk_{\mathcal{C}} : T \to \{0,1\}^*$ and $hpk_{\mathcal{C}} : T \to \{0,1\}^*$ as follows. We start with $hsk_{\mathcal{C}} \leftarrow hsk$, $hpk_{\mathcal{C}} \leftarrow hpk$ and then repeatedly redefine $hsk_{\mathcal{C}}$, $hpk_{\mathcal{C}}$ according to the following simple rule for $j = 1, \ldots, \delta$. As long as there exist nodes $\alpha_i^{(0)}$ and $\alpha_i^{(1)}$ in $T \setminus \mathcal{C}$ such that

$$\alpha_j^{(0)} \in \alpha_{j-1}^{(0)} \in \alpha_{j-2}^{(0)} \in \ldots \in \alpha_{t+1}^{(0)} \in \alpha_t \ni \alpha_{t+1}^{(1)} \ni \ldots \ni \alpha_{j-2}^{(1)} \ni \alpha_{j-1}^{(1)} \ni \alpha_j^{(1)} \quad , \tag{12.1}$$

and $\alpha_j^{(0)}$ is before $\alpha_j^{(1)}$ in the lexicographical order, redefine $(hpk_\mathcal{C}(\alpha_j^{(1)}), hsk_\mathcal{C}(\alpha_j^{(1)}))$ to equal $(hpk_\mathcal{C}(\alpha_j^{(0)}), hsk_\mathcal{C}(\alpha_j^{(0)}))$. This is illustrated in Figure 12.1.

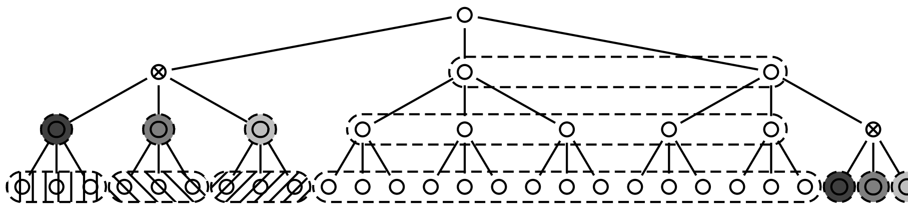


Figure 12.1: Nodes grouped together on the same level are given identical keys by the ideal functionality.

Ideal functionalities for signature schemes may either make the public key an explicit part of the protocol, as in [31] and [5], or the public key can be hidden from the environment. Here we use the second approach, since it makes the exposition easier to follow.

An Ideal Functionality for Hierarchical Group Signatures

In Figure 12.2 we give an ideal functionality for hierarchical group signatures.

Let us discuss why the functionality given in Figure 12.2 captures the notion of hierarchical group signatures. Consider the anonymity requirement. An adversary should not be able to distinguish two signatures σ_0 and σ_1 of some message m computed by honest $S_{\alpha_\delta^{(0)}}$ and $S_{\alpha_\delta^{(1)}}$ respectively, provided that all M_β for all indices β in the chains in Equation (12.1) are honest. The construction of $hsk_\mathcal{C}$ and $hpk_\mathcal{C}$ ensures that signatures of a message m of all such signers are identically distributed. Thus, no information on the identity of the signer is leaked by the signatures.

Next we consider the traceability property. As explained in Canetti [31] a signature functionality must accept any signature computed with the key of a corrupted party. In the hierarchical group signature setting this corresponds to accepting all signatures that traces to a corrupted party if the keys hsk are used. Thus, the functionality accepts a signature if and only if a signer has previously requested a signature from $\mathcal{F}_{\mathcal{HGS}}$ and it has stored it or the signature traces to a corrupted signer. The opening of a signature is handled similarly, but here we must take care to find the index related to the index of the party requesting the opening.

Functionality 12.2.1 (Hierarchical Group Signatures)**.** The ideal *hierarchical group signature* functionality $\mathcal{F}_{\mathcal{HGS}}$ running with parties $\{M_\alpha\}_{\alpha \in T}$ for a tree T with all leaves at the same depth, and ideal adversary $\mathcal{S}$ which corrupts M_α with $\alpha \in \mathcal{C}$.

1. Execute $(hpk, hsk) \leftarrow \mathsf{HGKg}(T, 1^\kappa)$ and hand $((\mathcal{S}, \mathtt{Keys}, hpk, hsk(\mathcal{C})))$ to $\mathcal{C}_\mathcal{I}$.
2. Then handle incoming messages as follows.
   - **Sign.** Upon reception of $(M_\alpha, \mathtt{Sig}, m)$ with $M_\alpha \in \mathcal{L}(T)$, compute $\sigma \leftarrow \mathsf{HGSig}(m, T, hpk_\mathcal{C}, hsk_\mathcal{C}(\alpha))$ and store $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$, where the third component is the path from the root $\alpha_0 = \rho$ to a leaf $\alpha_\delta = \alpha$. Then hand $(M_\alpha, \mathtt{Sig}, m, \sigma)$ to $\mathcal{C}_\mathcal{I}$.
   - **Verify.** Upon reception of $(M_\beta, \mathtt{Vf}, m, \sigma)$, check if $(m, \sigma, (\cdots))$ is stored. If so set $b \leftarrow 1$.

     If not, compute $b \leftarrow \mathsf{HGVf}(T, hpk, m, \sigma)$. If $b = 1$, then proceed as follows: Let $\alpha_0 \leftarrow \rho$. For $j = 0, \ldots, \delta - 1$ set $\alpha_{j+1} \leftarrow \mathsf{HGOpen}(T, hpk, hsk(\alpha_j), m, \sigma)$ and set $\alpha_{j+1} \leftarrow \perp$ if $\alpha_{j+1} \notin \alpha_j$. If $\alpha_\delta \in \mathcal{C}$, then store $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$. Otherwise set $b \leftarrow 0$.

     Hand $(M_\beta, \mathtt{Vf}, m, \sigma, b)$ to $\mathcal{C}_\mathcal{I}$.
   - **Open.** Upon reception of $(M_\beta, \mathtt{Open}, m, \sigma)$ with $M_\beta \notin \mathcal{L}(T)$, execute $(\mathtt{Vf}, m, \sigma)$. Then check if $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$ is stored for a path $(\alpha_0, \ldots, \alpha_\delta)$ in T where $\beta = \alpha_t$. If so, set $\alpha \leftarrow \alpha_{t+1}$, and otherwise set $\alpha \leftarrow \perp$.

     Hand $(M_\beta, \mathtt{Open}, m, \sigma, \alpha)$ to $\mathcal{C}_\mathcal{I}$.

Figure 12.2: The definition of $\mathcal{F}_{\mathcal{HGS}}$.

It may seem contrived to let the functionality output real signatures instead of random strings, since this requires that we give a new functionality for each construction of a HGS. However, the functionality must output random strings that are indistinguishable from real signatures since otherwise the environment can trivially distinguish the functionality from the real protocol. Thus, in one way or another the ideal signature functionality must depend on the signature scheme we wish to model. For standard signatures, Canetti [30, 31] solves this problem by allowing the adversary to compute the signature. This reveals to the adversary at what time every message is signed, and this is information which the protocol should not leak. Note that the functionality also depends on the set of parties which are corrupted by $\mathcal{S}$. This may seem odd at first, but a functionality is nothing more than a conceptual model, and in any simulation the ideal adversary knows precisely which parties are corrupted by A, so this is not a problem.

We conclude that the above definition captures the notion of hierarchical group signatures.

It is natural to ask how the definition of a secure hierarchical group signature scheme given in Trolin and Wikström [84] relates to the above definition, i.e., if it is the "right definition". Suppose we are given a hierarchical group signature scheme $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$ as defined in [84]. In Section 12.4 we describe an ideal functionality $\mathcal{F}_{\mathsf{HKg}}$ that corresponds to the key generation algorithm HGKg. We also give a straight-forward protocol $\pi_{\mathcal{HGS}}$ which essentially consists of the three algorithms HGSig, HGVf, and HGOpen. Then we prove the following theorem.

Theorem 12.2.1. *Let $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOpen})$ be a secure hierarchical group signature scheme according the definition in [84]. Then $\pi_{\mathcal{HGS}}$ securely realizes $\mathcal{F}_{\mathcal{HGS}}$ in the $\mathcal{F}_{\mathsf{HKg}}$-hybrid model for a static adversary.*

Thus, security according to the definition in [84] implies UC-security in the $\mathcal{F}_{\mathsf{HKg}}$-hybrid model, if we ignore some minor technical modifications. It is not clear that the opposite implication holds, since the in the original definitions the adversary is adaptive. On the other hand the adaptivity is restricted and seems not strong enough to allow an adaptive adversary in the UC-framework.

Adaptive vs. Static Adversaries. One of the obstacles when trying to construct a functionality that is secure against an adaptive adversary can be formulated as follows. The signatures which the functionality creates should be intuitively anonymous, i.e., should contain no knowledge about the signer. On the other hand, if group manager β is corrupted, it may leak its key to the environment, Therefore, signers belonging to different subtrees of β must be distinguishable with regard to β's secret key, which contradicts the requirement that the signatures should be indistinguishable. For an adaptive adversary, the functionality does not know which group managers will be corrupted after the signature has been constructed.

At its best the UC-framework allows simple and intuitive functionalities that can also be realized. In contrast to what was originally claimed [30], signature schemes [31], and even more so (hierarchical) group signature schemes require complicated definitions as functionalities. It has been argued [81] that such functionalities are better understood using classical experiments such as those in our original definitions. On the other hand we need a composability theorem to use these schemes as building blocks in a rigorous way. Theorem 12.2.1 allows us to use the more intuitively appealing original definitions, and still have the guarantee that the result is universally composable.

12.3 Optimistic Hierarchical Group Signatures

Next we consider how the functionality can be modified such that it supports an optimistic protocol. We modify the ideal functionality to accommodate for optimistic opening, which may fail but never points out an innocent signer, and joint

Functionality 12.3.1 (Optimistic Hierarchical Group Signatures)**.** The ideal *optimistic hierarchical group signature* functionality $\mathcal{F}_{\mathcal{HGS}}^{\mathrm{Opt}}$ running with parties $\{M_\alpha\}_{\alpha \in T}$ for a tree T with all leaves at the same depth, and ideal adversary $\mathcal{S}$ which corrupts M_α with $\alpha \in \mathcal{C}$.

1. Execute $(hpk, hsk, sk_{\mathfrak{T}}) \leftarrow \mathsf{HGKg}(T, 1^\kappa)$ and hand $(\mathcal{S}, \mathtt{Keys}, hpk, sk_{\mathfrak{T}})$ to $\mathcal{C}_{\mathcal{I}}$.
2. Then handle incoming messages as follows.
   - **Sign.** Upon reception of $(M_\alpha, \mathtt{Sig}, m)$ with $M_\alpha \in \mathcal{L}(T)$, compute $\sigma \leftarrow \mathsf{HGSig}(m, T, hpk_{\mathcal{C}}, hsk_{\mathcal{C}}(\alpha))$ and store $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$, where the third component is the path from the root $\alpha_0 = \omega$ to a leaf $\alpha_\delta = \alpha$. Then hand $(M_\alpha, \mathtt{Sig}, m, \sigma)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Verify.** Upon reception of $(\beta, \mathtt{Vf}, m, \sigma)$, check if $(m, \sigma, (\cdots))$ is stored. If so set $b = 1$.

     If not, compute $b \leftarrow \mathsf{HGVf}(T, hpk, m, \sigma)$. If $b = 1$, then proceed as follows: Let $\alpha_0 \leftarrow \omega$. For $j = 0, \ldots, \delta - 1$ compute $\alpha_{j+1} \leftarrow \mathsf{HGTrustOpen}(T, hpk, sk_{\mathfrak{T}}, m, \sigma)$. If $\alpha_{j+1} \notin \alpha_j$, then set $\alpha_{j+1} \leftarrow \perp$. If $\alpha_\delta \in \mathcal{C}$, then store $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$. Otherwise set $b \leftarrow 0$.

     Hand $(\beta, \mathtt{Vf}, m, \sigma, b)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **OptOpen.** Upon reception of $(\beta, \mathtt{OptOpen}, m, \sigma)$ with $M_\beta \in T \setminus \mathcal{L}(T)$, execute $(\perp, \mathtt{Vf}, m, \sigma)$. Then check if $(m, \sigma, (\alpha_0, \ldots, \alpha_\delta))$ is stored for a path $(\alpha_0, \ldots, \alpha_\delta)$ in T where $\beta = \alpha_t$. If so set $\alpha \leftarrow \mathsf{HGOptOpen}(T, hpk, hsk(\alpha_t), m, \sigma)$ and set $\alpha \leftarrow \perp$ if $\alpha \neq \alpha_{t+1}$.

     Hand $(\beta, \mathtt{OptOpen}, m, \sigma, \alpha)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **TrustedOpen.** Upon reception of $(\beta, \mathtt{TrustOpen}, m, \sigma)$, execute $(\cdot, \mathtt{Vf}, \sigma, m)$.

     Let α be such that $(\sigma, m, (\ldots, \beta', \beta, \alpha, \ldots))$ for $\beta \in \beta'$ is stored, and set $\alpha \leftarrow \perp$ if no such chain is found.

     Return $((\mathfrak{T}, \mathtt{TrustOpen}, m, \sigma, \alpha), (\mathcal{S}, \mathtt{TrustOpen}, \beta, m, \sigma))$ to $\mathcal{C}_{\mathcal{I}}$.

Figure 12.3: The definition of $\mathcal{F}_{\mathcal{HGS}}^{\mathrm{Opt}}$.

opening, which answers correctly. We also include a mechanism for a group manager to prove that it has opened a signature correctly. The functionality is given in Figure 12.3.

The functionality $\mathcal{F}_{\mathcal{HGS}}^{\mathrm{Opt}}$ is identical to $\mathcal{F}_{\mathcal{HGS}}$ if the function HGOpen is replaced by $\mathsf{HGTrustOpen}$, and a call $\mathsf{HGOptOpen}$ with the property that it always outputs either the same as $\mathsf{HGTrustOpen}$ or $\perp$ is added.

We define the protocol $\pi_{\mathcal{HGS}}^{\mathrm{Opt}}$ from the five algorithms HGKg, HGSig, HGVf,

$\mathsf{HGOptOpen}$, $\mathsf{HGTrustOpen}$ in the obvious way and prove the following theorem.

Theorem 12.3.1. *Let $\mathcal{HGS} = (\mathsf{HGKg}, \mathsf{HGSig}, \mathsf{HGVf}, \mathsf{HGOptOpen}, \mathsf{HGTrustOpen})$ be a secure optimistic hierarchical group signature scheme according to Definition 8.3.4. Then $\pi^{\mathrm{Opt}}_{\mathcal{HGS}}$ securely realizes $\mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$ in the $\mathcal{F}^{\mathrm{Opt}}_{\mathsf{HKg}}, \mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$-hybrid model for a static adversary.*

Hence also in the case of optimistic hierarchical group signatures security in the classical sense implies security in the UC-model.

12.4 The Real Protocols

We now describe the complete protocols using the algorithms defined previously.

Hierarchical Group Signatures

We begin by defining the ideal key generator, and the we proceed by giving the full protocol.

Functionality 12.4.1 (Ideal HGS Key Generator $\mathcal{F}_{\mathsf{HKg}}$)**.** The ideal HGS *key generator* functionality $\mathcal{F}_{\mathsf{HKg}}$ running with parties $\{M_\alpha\}_{\alpha\in T}$ for a tree T with all leaves at the same depth, and ideal adversary $\mathcal{S}$.

- Execute $(hpk, hsk) \leftarrow \mathsf{HGKg}(T, 1^\kappa)$. Then hand $((\mathcal{S}, \texttt{Keys}),(M_\alpha, \texttt{Keys}, hpk, hsk(\alpha))_{\alpha\in T})$ to $\mathcal{C_I}$.

Then we simply translate the three algorithms HGSig, HGVf, and HGOpen into a single protocol.

Protocol 12.4.1 (Generic UC-HGS Protocol)**.**
The protocol $\pi_{\mathcal{HGS}} = \{M_\alpha\}_{\alpha\in T}$ running with functionality $\mathcal{F}_{\mathsf{HKg}}$ and parties M_α in a tree T with all leaves at the same depth is defined as follows.

First all parties wait for $(\texttt{Keys}, hpk, hsk(\alpha))$ from $\mathcal{F}_{\mathsf{HKg}}$. Then they do as follows.

All Parties

- Upon reception of $(M_\alpha, \texttt{Vf}, m, \sigma)$, compute $b = \mathsf{HGVf}(T, hpk, m, \sigma)$ and output $(\texttt{Vf}, m, \sigma, b)$.

Leaves $M_\alpha \in \mathcal{L}(T)$.

- Upon reception of $(M_\alpha, \texttt{Sig}, m)$ compute $\sigma = \mathsf{HGSig}(m, T, hpk, hsk(\alpha))$ and output $(\texttt{Sig}, m, \sigma)$.

Group Managers $M_\beta \notin \mathcal{L}(T)$.

- Upon reception of $(M_\beta, \texttt{Open}, m, \sigma)$, compute $\alpha = \mathsf{HGOpen}(T, hpk, hsk(\beta), m, \sigma)$ and output $(\texttt{Open}, m, \sigma, \alpha)$.

Optimistic Hierarchical Group Signatures

Using the algorithms defined above we describe the complete protocol. First we need to define the ideal key generator and the ideal trusted open functionality. We choose to separate the two, since the trusted opening appears to be possible to realize as a distributed protocol using known techniqueswhereas we are not aware of any such protocol (except by using general methods) for the key generation.

Functionality 12.4.2 (Ideal Optimistic HGS Key Generator $\mathcal{F}_{\mathsf{HKg}}^{\mathrm{Opt}}$)**.**
The ideal optimistic HGS *key generator* functionality $\mathcal{F}_{\mathsf{HKg}}^{\mathrm{Opt}}$ running with parties $\{M_\alpha\}_{\alpha \in T}$ for a tree T with all leaves at the same depth, and ideal adversary $\mathcal{S}$.

- Execute $\mathsf{HGKg}(1^\kappa, T)$ except for the generation of the trusted key of Step 2 and store the result as (hpk, hsk). Then hand $((\mathcal{S}, \texttt{Keys}), (M_\alpha, \texttt{Keys}, hpk, hsk(\alpha))_{\alpha \in T})$ to $\mathcal{C}_{\mathcal{I}}$.

Functionality 12.4.3 (Ideal Trusted Open Functionality $\mathcal{F}_{\mathsf{TrOpen}}^{\mathrm{Opt}}$)**.**
The ideal optimistic HGS *trusted open* functionality $\mathcal{F}_{\mathsf{TrOpen}}^{\mathrm{Opt}}$ running with parties denoted $\{M_\alpha\}_{\alpha \in T}$ for a tree T with all leaves at the same depth, and ideal adversary $\mathcal{S}$.

- Execute Step 2 of $\mathsf{HGKg}(1^\kappa, T)$ and store the result as $(pk_{\mathfrak{T}}, sk_{\mathfrak{T}})$. Then hand $((\mathcal{S}, \texttt{TrustedKey}, pk_{\mathfrak{T}}, sk_{\mathfrak{T}}), (M_\alpha, \texttt{TrustedKey}, pk_{\mathfrak{T}})_{\alpha \in T})$ to $\mathcal{C}_{\mathcal{I}}$.
- Then handle incoming messages as follows.
  - **Trusted Open.** Upon reception of $(\beta, \texttt{TrustOpen}, m, \sigma)$, execute $\alpha \leftarrow \mathsf{HGTrustOpen}(T, hpk, sk_{\mathfrak{T}}, \beta, m, \sigma)$. Hand $((\mathcal{S}, \texttt{TrustOpen}, m, \sigma), (\beta, \texttt{TrustOpen}, m, \sigma, \alpha))$ to $\mathcal{C}_{\mathcal{I}}$.

Note that the keys for trusted opening is generated by $\mathcal{F}_{\mathsf{TrOpen}}^{\mathrm{Opt}}$ rather than by $\mathcal{F}_{\mathsf{HKg}}^{\mathrm{Opt}}$.

Protocol 12.4.2 (Optimistic HGS Protocol)**.**
The optimistic hierarchical group signature protocol $\pi_{\mathcal{HGS}}^{\mathrm{Opt}}$ running with ideal functionalities $\mathcal{F}_{\mathsf{HKg}}^{\mathrm{Opt}}$, $\mathcal{F}_{\mathsf{TrOpen}}^{\mathrm{Opt}}$ and parties $((M_\beta)_{\beta \in T}$ for a tree T is defined as follows.

First all parties wait for a message $(\texttt{Keys}, hpk, hsk(\alpha))$ from $\mathcal{F}_{\mathsf{HKg}}^{\mathrm{Opt}}$ and for a message$(\texttt{TrustedKey}, pk_{\mathfrak{T}})$ from $\mathcal{F}_{\mathsf{TrOpen}}^{\mathrm{Opt}}$. They incorporate $pk_{\mathfrak{T}}$ into hpk. Then they do as follows.

All Parties

- On input $(M_\alpha, \texttt{Vf}, m, \sigma)$, compute $b = \mathsf{HGVf}(T, hpk, m, \sigma)$ and output $(\texttt{Vf}, m, \sigma, b)$.

Leaves $M_\alpha \in \mathcal{L}(T)$.

- On input (M_α, $\texttt{Sig}$, m) compute $\sigma = \mathsf{HGSig}(m, T, \mathit{hpk}, \mathit{hsk}(\alpha))$ and output $(\texttt{Sig}, m, \sigma)$.

Group Managers $M_\beta \notin \mathcal{L}(T)$.

- On input (M_β, $\texttt{OptOpen}$, m, σ), compute $\alpha \leftarrow \mathsf{HGOptOpen}(T, \mathit{hpk}, \mathit{hsk}(\beta), m, \sigma)$ and output ($\texttt{OptOpen}$, m, σ, α).

- On input (M_β, $\texttt{TrustOpen}$, m, σ), hand ($\mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$, $\texttt{TrustOpen}$, m, σ) to $\mathcal{C}_\mathcal{I}$. Upon reception of (M_β, $\texttt{TrustOpen}$, m, σ, α), output ($\texttt{TrustOpen}$, m, σ, α).

12.5 The Proofs of Security

Hierarchical Group Signatures

Here we prove Theorem 12.2.1.

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 UC-secure then we can break the security of $\mathcal{HGS}$.

The Ideal Adversary $\mathcal{S}$. Let $\mathcal{C}$ be the set of indices of participants corrupted by $\mathcal{A}$. The ideal adversary $\mathcal{S}$ corrupts the dummy participants $\tilde{M}_\alpha$ for which $\alpha \in \mathcal{C}$, returning $(\mathit{hpk}, \mathit{hsk}(\alpha))$ to $\mathcal{A}$. The ideal adversary is best described by starting with a copy of the original hybrid ITM-graph of the real protocol and replacing $\mathcal{Z}$ with a machine $\mathcal{Z}'$. The adversary $\mathcal{S}$ simulates all machines in V except those in $\mathcal{A}'$, and the corrupted machines M_α for $\alpha \in \mathcal{C}$ under $\mathcal{A}'$s control. We now describe how each machine is simulated.

Simulation of Links $(\mathcal{Z}, \mathcal{A})$, $(\mathcal{Z}, M_\alpha)$ *for* $\alpha \in \mathcal{C}$. $\mathcal{S}$ simulates $\mathcal{Z}'$ and $\tilde{M}_\alpha$ for $\alpha \in \mathcal{C}$, such that it appears as if $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ and M_α for $\alpha \in \mathcal{C}$ 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 $\mathcal{Z}$ and $\mathcal{A}$ being linked directly.

2. When $\mathcal{Z}'$ receives m from M_α for $\alpha \in \mathcal{C}$, m is written to $\mathcal{Z}$ by $\tilde{M}_\alpha$. When $\tilde{M}_\alpha$, $\alpha \in \mathcal{C}$, receives m from $\mathcal{Z}$, m is written to M_α by $\mathcal{Z}'$. This is equivalent to $\mathcal{Z}$ and M_α being linked directly for $\alpha \in \mathcal{C}$.

Note that since the protocol is completely non-interactive the honest simulated parties M_α for $\alpha \notin \mathcal{C}$ need actually not do anything, since they do not communicate with the corrupted parties at all.

The adversary waits until it receives $(\texttt{Keys}, \mathit{hpk}, \mathit{hsk}(\mathcal{C}))$ from $\mathcal{F}_{\mathcal{HGS}}$. Then it simulates $\mathcal{F}_{\mathsf{HKg}}$ except that it outputs the keys $(\mathit{hpk}, \mathit{hsk}(\mathcal{C}))$ instead of generating new keys.

Reaching a Contradiction. Suppose that $\mathcal{S}$ does not imply that $\mathcal{F}_{\mathcal{HGS}}$ is secure. Then there exists an adversary $\mathcal{A}'$, an environment $\mathcal{Z}$ with auxiliary input $z = \{z_n\}$, a constant $c > 0$ and an infinite index set $\mathcal{N} \subset \mathbb{N}$ such that for $n \in \mathcal{N}$ the advantage in of $\mathcal{Z}$ on auxiliary input z when distinguishing between the real protocol and the ideal functionally is non-negligible.

We define a sequence of hybrids as follows. Define π_0 to be the ideal protocol. We let π_j be identical to π_{j-1} except that in the ith signing request, with $i \leq j$, $\mathcal{F}_{\mathcal{HGS}}$ computes a signature σ with $hsk(\alpha)$ instead of with $hsk_{\mathcal{C}}(\alpha)$. The number of signing requests is bounded by a polynomial $q(\kappa)$. We let H_i be the output of $\mathcal{Z}_z$ when interacting with π_i.

Claim 7. There exists a negligible function $\varepsilon(\kappa)$ such that

$$|\Pr[H_0 = 1] - \Pr[H_{q(\kappa)} = 1]| < \varepsilon(\kappa) \ .$$

Proof. If the claim is false a hybrid argument implies that $|\Pr[H_{j-1} = 1] - \Pr[H_j = 1]|$ is non-negligible for some $0 < j \leq q(\kappa)$.

We construct an adversary H that breaks hierarchical anonymity of $\mathcal{HGS}$. H is identical to H_{j-1} except that $\mathcal{F}_{\mathsf{HKg}}$ does not generate any keys, but use the keys from the hierarchical anonymity experiment. It also requests the secret keys of all of the corrupted group managers to be able to hand these to the corrupted parties run as black boxes. Every invocation of the HGOpen-algorithm is replaced by a call to the HGOpen-oracle.

The jth sign request to $\mathcal{F}_{\mathcal{HGS}}$ must come from an honest dummy party, since otherwise H_j and H_{j-1} would be identically distributed. Let $S_{\alpha^{(1)}}$ be the signer which requests the jth signature on some message m, and let $S_{\alpha^{(0)}}$ be a signer such that $hsk_{\mathcal{C}}(\alpha^{(1)}) = hsk(\alpha^{(0)})$.

The adversary H does not decide whether to simulate π_{j-1} or π_j. Instead it outputs $(state, \alpha^{(0)}, \alpha^{(1)}, m)$, where $state$ is its internal state. The random variable $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{HGS},H}(\kappa, T)$ is identically distributed to H_{j-1} or H_j depending on if $b = 0$ or $b = 1$ respectively. Thus, $|\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{HGS},H}(\kappa, T) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{HGS},H}(\kappa, T) = 1]|$ is non-negligible which contradicts the hierarchical anonymity of $\mathcal{HGS}$ and the claim follows. □

Next we consider another family of hybrids. Define π'_0 to be equal to $\pi_{q(\kappa)}$. Let π'_j be identical to π'_{j-1} except that in the ith verify or open request, with $i \leq j$, $\mathcal{F}_{\mathcal{HGS}}$ defines $b \leftarrow \mathsf{HGVf}(T, hpk, m, \sigma)$ for a verification request and it defines $\alpha \leftarrow \mathsf{HGOpen}(T, hpk, hsk(\beta), m, \sigma)$ for an open request. The number of verification or open requests is bounded by a polynomial $q'(\kappa)$.

Claim 8. There exists a negligible function $\varepsilon'(\kappa)$ such that

$$|\Pr[H'_0 = 1] - \Pr[H'_{q'(\kappa)} = 1]| < \varepsilon'(\kappa) \ .$$

Proof. If the claim is false a hybrid argument implies that $|\Pr[H'_{j-1} = 1] - \Pr[H'_j = 1]|$ is non-negligible for some $0 < j \leq q'(\kappa)$.

We construct an adversary H' that breaks the hierarchical traceability of $\mathcal{HGS}$.

H' simulates π'_j, except that the simulated $\mathcal{F}_{\mathsf{HKg}}$ does not generate any keys, but use the keys from the hierarchical traceability experiment. Instead it requests the secret keys of all of the corrupted signers to be able to hand these to the corrupted parties run as black boxes. Every invocation of the HGSig-algorithm is also replaced by a call to the HGSig-oracle.

When the jth verify, $(M_\beta, \mathtt{Vf}, m, \sigma)$, or open, $(M_\alpha, \mathtt{Open}, m, \sigma)$, is handed to $\mathcal{F}_{\mathcal{HGS}}$, H' simply outputs (m, σ). Let $p = |\Pr[H'_{j-1} = 1] - \Pr[H'_j = 1]|$. The jth query must be answered differently for π_{j-1} and π_j with at least probability p, which is non-negligible.

If the jth query is a verify query, the only way the answer can differ is that (m, σ) is a signature such that $\mathsf{HGVf}(T, hpk, m, \sigma) = 1$, although no signing query for m has been given. Then H' wins the experiment for hierarchical traceability with advantage p.

If the jth query is an open query, $\mathsf{HGOpen}(T, hpk, hsk(\beta), m, \sigma)$ can differ from the ideal functionality in one of the following two ways.

1. The signature opens to $\perp$.
2. The signature σ opens to a corrupt party and the functionality has not been asked to sign the message m on behalf of the corrupted party.

In both these cases H' wins the experiment for hierarchical traceability with probability p. □

By construction we have that $H'_{q'(\kappa)}$ is identically distributed the output of $\mathcal{Z}$ when it executes the real protocol. Thus, we reached a contradiction, since the distance between H_0 and $H'_{q'(\kappa)}$ is assumed to be non-negligible, but at the same time it is at most the some of a polynomial number of negligible functions.

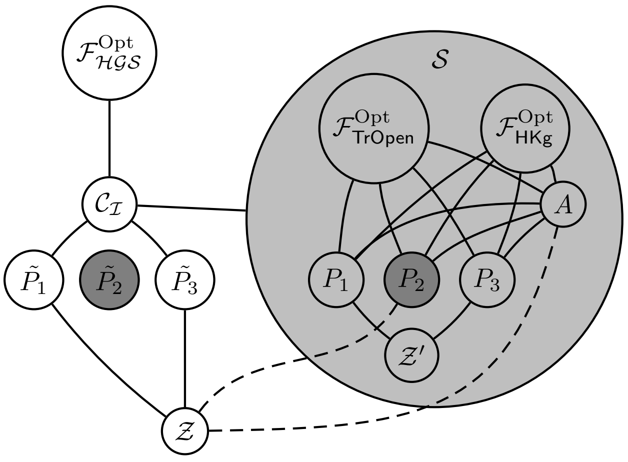
This concludes the proof. □

Optimistic Hierarchical Group Signatures

Here we prove Theorem 12.3.1.

Proof. Defining the Hybrids. We prove the theorem with a hybrid argument. We build a polynomial-size chain of protocols $\pi_0, \pi_1, \ldots, \pi_t$ such that $\pi_0 = \mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$ and $\pi_t = \pi^{\mathrm{Opt}}_{\mathcal{HGS}}$. Then we show that if there exists an adversary A which can distinguish between π_t and π_{t+1} for some t, then A can be used to break the security of the hierarchical group signature scheme.

We let $\pi_0 = \pi$. We define π_0^t to be π_0^{t-1} with the difference that the tth call to `Sig` produces a valid signature rather than a signature by a dummy signer. Let $\pi_1^0 = \pi_0^m$, and define π_1^t to be π_1^{t-1} with the difference that `OptOpen` runs according to the protocol rather than according to the ideal functionality. $\pi_2^0 = \pi_1^m$, and π_2^t is π_2^{t-1} with the difference that the tth call to `TrustOpen` runs according to the real protocol rather than according to the ideal protocol.



$\mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$
$\mathcal{S}$
$\mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$
$\mathcal{F}^{\mathrm{Opt}}_{\mathsf{HKg}}$
$\mathcal{C}_{\mathcal{I}}$
A
$\tilde{P}_1$
$\tilde{P}_2$
$\tilde{P}_3$
P_1
P_2
P_3
$\mathcal{Z}'$
$\mathcal{Z}$

Figure 12.4: The simulator $\mathcal{S}$ for a protocol with three players where P_2 is corrupted. The dashed edges represent simulated connections.

Building the Simulator. By assumption $\mathcal{Z}$ distinguishes between $\mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$ and $\pi^{\mathrm{Opt}}_{\mathcal{HGS}}$ for any ideal adversary. In particular it distinguishes between the two protocols for the adversary defined as follows.

For each player P_i that the real-world adversary A corrupts, the ideal adversary $\mathcal{S}$ corrupts the corresponding dummy player $\tilde{P}_i$ and outputs the internal state of P_i containing the public and the private key. When a corrupted dummy player $\tilde{P}_i$ receives a message m from $\mathcal{Z}$, the simulator $\mathcal{S}$ lets $\mathcal{Z}'$ send m to P_i. When a corrupted P_i outputs a message m to $\mathcal{Z}'$, then $\mathcal{S}$ instructs the corrupted $\tilde{P}_i$ to output m to $\mathcal{Z}$. This corresponds to P_i being linked directly to $\mathcal{Z}$.

The simulated real-world adversary A is connected to $\mathcal{Z}$, i.e., when $\mathcal{Z}$ sends m to $\mathcal{S}$, $\mathcal{Z}'$ hands m to A, and when A outputs m to $\mathcal{Z}'$, $\mathcal{S}$ hands m to $\mathcal{Z}$. All non-corrupted players are simulated honestly. The corrupted players run according to their respective protocols.

The adversary waits until it receives $(\mathtt{Keys}, hpk, hsk(\mathcal{C}))$ from $\mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$. It then simulates $\mathcal{F}^{\mathrm{Opt}}_{\mathsf{HKg}}$ and $\mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$ with the difference that the keys $(hpk, hsk(\mathcal{C}))$ and $pk_{\mathfrak{T}}$ are used.

If $\mathcal{S}$ receives the message $(\mathtt{TrustOpen}, \beta, m, \sigma)$ from $\mathcal{F}^{\mathrm{Opt}}_{\mathcal{HGS}}$, then it instructs $\mathcal{Z}'$ to send $(\mathtt{TrustOpen}, m, \sigma)$ to β. All other functions are local and need not be simulated for A.

Assume $\mathcal{Z}$ distinguishes between π_0^{t-1} and π_0^t. We show how to use $\mathcal{Z}$ to break the hierarchical anonymity of $\mathcal{HGS}$ by constructing a machine A that participates in Experiment 8.3.2 using $\mathcal{Z}$ as a black box. There is no key generation performed by the functionality, but instead the keys $(hpk, hsk(\mathcal{L}(T)), pk_{\mathfrak{T}})$ received from the experiments are used. For each $\beta \in \mathcal{C}$ A requests the group manager key in the corrupt phase. During the execution A simulates signature generation calls honestly using the signing keys it received in the experiment. Opening is performed honestly using oracle queries in the experiment. $\mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$ is simulated by querying the HGTrustOpen oracle of the experiment.

Let α be the signer that receives the tth `Sig` query, and let the query be to sign m. A then returns (α, α', m) on the choose query, where α' is a signer given the same key as α in the ideal functionality. The challenge σ is returned as result to the `Sig` query. Note that if the σ is produced by α, the protocol is π_0^t, and otherwise the protocol is π_0^{t-1}. Since by assumption $\mathcal{Z}$ is able to distinguish between the two with non-negligible probability, A has a non-negligible advantage in its experiment.

Assume $\mathcal{Z}$ distinguishes between π_1^{t-1} and π_1^t. We show how to construct a machine A which takes part in Experiment 8.3.1 and breaks the hierarchical traceability of $\mathcal{HGS}$. Rather than constructing the keys itself, the keys $(hpk, hsk(T \setminus \mathcal{L}(T)), pk_{\mathfrak{T}})$ are taken as the are taken from the experiment. For each $\beta \in \mathcal{C}$, A requests the private signing key in corrupt phase. The queries `OptOpen` and `TrustOpen` are answered honestly using the group manager keys given to A in the experiment. Signing queries are answered using the signing oracle provided to A in the experiment, and $\mathcal{F}^{\mathrm{Opt}}_{\mathsf{TrOpen}}$ uses $sk_{\mathfrak{T}}$ from the experiment. Let (σ, m) be the tth `OptOpen` query, and let β be the recipient. By assumption there is a non-negligible probability p that the response differs between π_1^{t-1} and π_1^t, which implies that (σ, m) are opened differently by the table lookup in the ideal functionality and the HGOptOpen algorithm. A honestly computes $(\beta', \tau) = \mathsf{HGOptOpen}(T, hpk, hsk(\beta), \sigma, m)$, and returns $(m, \sigma, \beta', \tau, \perp)$ as response to the choose query. Since the signature (σ, m) is opened differently by the ideal functionality and the protocol the signature, the ideal functionality has not created σ and β' is not corrupt. Therefore the advantage of A is p, and hence A breaks the hierarchical traceability of $\mathcal{HGS}$.

Assume $\mathcal{Z}$ distinguishes between π_2^{t-1} and π_2^t. Proceeding as in the paragraph above, A intercepts the tth `TrustOpen` query, runs HGTrustOpen honestly and outputs the result. The advantage of A in Experiment 8.3.1 is the same as the advantage of $\mathcal{Z}$, which by assumption is non-negligible. $\square$

Chapter 13

Conclusion of Part III

We have introduced a generalization of group signatures which we call hierarchical group signatures. We have suggested the definitional framework for the new notion as well as three constructions. Our first construction is secure assuming only the existence of a trapdoor permutation, but it is very inefficient. Our second construction requires the strong RSA assumption, the decisional Diffie-Hellman assumption and the random oracle model, but is more efficient. Our last construction is practical and requires the strong RSA assumption, the decisional Diffie-Hellman assumption and the random oracle model, but is secure under a relaxed definition, where additional interaction between parties may be necessary in case of a corrupted signer.

There are different ways to improve our results. As it stands now, there is a scheme which is secure under plausible number theoretic assumptions, but which is not as efficient as one could wish, and there is a scheme which is practical for use on ordinary PCs, but which is secure only under a relaxed definition. One would want a scheme which satisfies the stronger security definition and is practical.

Our definitions and schemes are only for static groups. It would be desirable to extend these to cover also dynamic groups by elminating the trusted key generator. Most likely such definitions would be quite cumbersome.

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] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *4th ACM Conference on Computer and Communications Security – CCS*, pages 7–17. ACM Press, 1997.

[3] 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.

[4] 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.

[5] M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. In *Information Security Conference – ISC 2004*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–74. Springer Verlag, 2004. Full version at `http://eprint.iacr.org/2003/240`.

[6] 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.

[7] 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.

[8] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *SIAM Journal on Computing*, 39(1):214–233, 1992.

[9] 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.

[10] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security – CCS*, pages 62–73. ACM Press, 1993.

[11] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *RSA Conference 2005, Cryptographers' Track 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2004/077`.

[12] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference – TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 60 – 79. Springer Verlag, 2006. Full version at `http://eprint.iacr.org/2005/304`.

[13] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.

[14] M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems – TOCS*, 1(2):175–193, 1983.

[15] 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.

[16] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[17] 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.

[18] 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.

[19] 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.

[20] S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer Verlag, 1994.

[21] E. Brickell, P. Gemmell, and D. Kravitz. Tracing extensions to anonymous cash and the making of anonymous change. In *6th Annual ACM-SIAM Symposium on Discrete Algorithms – SODA*, pages 457–466. ACM Press, 1995.

[22] 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.

[23] J. Camenisch. Efficient and generalized group signatures. In *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer Verlag, 1997.

[24] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks – SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[25] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2005/060`.

[26] 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.

[27] J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160–174. Springer Verlag, 1999.

[28] 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.

[29] 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.

[30] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science – FOCS*. IEEE Computer Society Press, 2001. Full version at `http://eprint.iacr.org/2000/067`.

[31] R. Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop – CSFW*, pages 219–235. IEEE Computer Society Press, 2004. Full version at `http://eprint.iacr.org/2003/239`.

[32] 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.

[33] M. Chase and A. Lysyanskaya. On signatures of knowledge. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer Verlag, 2006. Full version at `http://eprint.iacr.org/2006/184`.

[34] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] T. ElGamal. A public key cryptosystem and a signiture scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[43] 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.

[44] N.T. Ferguson. Single term off-line coins. In *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 318–328. Springer Verlag, 1993.

[45] 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 186–194. Springer Verlag, 1987.

[46] M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer Verlag, 2006.

[47] 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.

[48] 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.

[49] O. Goldreich and L. A. 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.

[50] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[51] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[52] 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.

[53] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Advances in Cryptology – CRYPTO'88*, volume 600 of *Lecture Notes in Computer Science*, pages 8–26. Springer Verlag, 1990.

[54] S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *Advances in Cryptology – EUROCRYPT2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 221–242. Springer Verlag, 2000. See also `http://eprint.iacr.org/2000/019`.

[55] 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.

[56] 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`.

[57] A. Kiayias and M. Yung. Efficient secure group signatures with dynamic joins and keeping anonymity against group managers. In *Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 151–170. Springer Verlag, 2005.

[58] 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.

[59] N. Koblitz. *Algebraic Aspects of Cryptography.* Springer Verlag, 1998.

[60] A. Lindgren. *Mästerdetektiven Blomkvist.* Rabén & Sjögren, 1946. Title in English: Bill Bergson, Master Detective.

[61] M. Liskov and S. Micali. Amortized e-cash. In *Financial Cryptography 2001*, volume 2339 of *Lecture Notes in Computer Science*, pages 1–20. Springer Verlag, 2001.

[62] 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.

[63] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[64] R. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1980.

[65] 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.

[66] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, 1988.

[67] B. Möller. *Public-Key Cryptography – Theory and Practice.* PhD thesis, Technische Universität Darmstadt, 2003.

[68] T. Nakanishi, M. Shiota, and Y. Sugiyama. An efficient online electronic cash with unlinkable exact payments. In *Information Security Conference – ISC 2004*, volume 3225 of *Lecture Notes in Computer Science*, pages 367–378. Springer Verlag, 2004.

[69] T. Nakanishi and Y. Sugiyama. Unlinkable divisible electronic cash. In *Information Security Workshop – ISW 2000*, volume 1975 of *Lecture Notes in Computer Science*, pages 121–134. Springer Verlag, 2000.

[70] 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.

[71] A. M. Odlyzko. Discrete logarithms: The past and the future. *Designs, Codes, and Cryptography*, 19(2/3):129–145, 2000.

[72] T. Okamoto and K. Ohta. Disposable zero-knowledge authentication and their application to untraceable electronic cash. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 481 – 496. Springer Verlag, 1990.

[73] T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 324–337. Springer Verlag, 1992.

[74] The prime pages. `http://primes.utm.edu`, March 2004.

[75] The proth search page. `http://www.prothsearch.net`, March 2004.

[76] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM Symposium on the Theory of Computing – STOC*, pages 387–394. ACM Press, 1990.

[77] 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.

[78] T. Sander and A. Ta-Shma. Auditable, anonymous electronic cash. In *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 555–572. Springer Verlag, 1999.

[79] 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.

[80] A. De Santis and G.Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd IEEE Symposium on Foundations of Computer Science – FOCS*, pages 427–436. IEEE Computer Society Press, 1992.

[81] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/2004/332`.

[82] 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.

[83] M. Trolin. A universally composable scheme for electronic cash. In *Advances in Cryptology – INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 347 – 360. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2005/341`.

[84] M. Trolin and D. Wikström. Hierarchical group signatures. In *International Colloquium on Automata, Languages and Programming – ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 446 – 458. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2004/311`.

[85] 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.

[86] V. Varadharajan, K.Q. Nguyen, and Y. Mu. On the design of efficient RSA-based off-line electronic cash schemes. *Theoretical Computer Science*, 226:173–184, 1999.

[87] V. Wei. More compact e-cash with efficient coin tracing. Cryptology ePrint Archive, Report 2005/411, 2005. `http://eprint.iacr.org/2005/411`.

[88] 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.

[89] C. Zamfir, A. Damian, I. Constandache, and V. Cristea. An efficient ecash platform for smart phones. In *E_COMM_LINE 2004*, pages 5–9, 2004. Also available at `http://linux.egov.pub.ro/~ecash/`.