Signature Schemes and Applications to Cryptographic Protocol Design

by

Anna Lysyanskaya

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author .... [illegible]
Department of Electrical Engineering and Computer Science
September 4, 2002

Certified by ....
Ronald L. Rivest
Viterbi Professor of Computer Science
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Signature Schemes and Applications to Cryptographic Protocol Design

by
Anna Lysyanskaya

Submitted to the Department of Electrical Engineering and Computer Science
on September 4, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Signature schemes are fundamental cryptographic primitives, useful as a stand-alone application, and as a building block in the design of secure protocols and other cryptographic objects. In this thesis, we study both the uses that signature schemes find in protocols, and the design of signature schemes suitable for a broad range of applications.

An important application of digital signature schemes is an anonymous credential system. In such a system, one can obtain and prove possession of credentials without revealing any additional information. Such systems are the best means of balancing the need of individuals for privacy with the need of large organizations to verify that the people they interact with have the required credentials. We show how to construct an efficient anonymous credential system using an appropriate signature scheme; we then give an example of such a signature scheme. The resulting system is the first one with satisfactory communication and computation costs.

The signature scheme we use to construct an anonymous credential system is of independent interest for use in other protocols. The special property of this signature scheme is that it admits an efficient protocol for a zero-knowledge proof of knowledge of a signature.

Further, we consider the question of revocation of signatures. We obtain an efficient revocation scheme. This has immediate consequences for revocation of credentials in our credential system.

We explore other uses for signature schemes as building blocks for designing cryptographic objects and secure protocols. We give a unique signature scheme which has implications for verifiable random functions and for non-interactive zero-knowledge proofs.

Finally, we consider the use of signatures for implementing a broadcast channel in a point-to-point network. It was previously shown that while broadcast was impossible without computational assumptions in a point-to-point network where one-third or more nodes exhibited adversarial behavior, using an appropriate set-up phase and a signature scheme, the impossibility could be overcome. We show that the situation is more complex than was previously believed. We consider the *composition* of protocols in this model, and discover severe limitations. We also show how to augment the model to overcome these limitations.

Thesis Supervisor: Ronald L. Rivest
Title: Viterbi Professor of Computer Science

Acknowledgments

My experience as a graduate student at MIT has been amazing. It would not have been the same without my mentors and friends.

First and foremost, I would like to thank my advisor Ron Rivest. Ron has been a perfect advisor for me, and I feel that I have been truly privileged to work under his supervision. I am grateful to him for giving me direction and confidence, for being patient, encouraging, and kind, and for everything that I learned from him.

Silvio Micali has had a tremendous impact on the way that I perceive research and life in general. I am grateful to him for the interest that he has taken in helping me develop both as a researcher and as a teacher, and for his overflowing energy and enthusiasm.

My sincere thanks to Shafi Goldwasser. Shafi has always taken an interest in my work, and I am grateful to her for her encouragement, kindness, and patience.

I am very grateful to Madhu Sudan for always being friendly, open and helpful. I also thank him for inviting me to take part in organizing the theory seminar at MIT.

My gratitude also goes to Charles Leiserson, who was my graduate counselor and helped me a great deal, always having some valuable insight and advice for me.

The theory group at MIT's Lab for Computer Science has been a nurturing and stimulating environment. I am grateful to Leo Reyzin, whom I can always rely upon for finding an approach to any problem, technical or otherwise, and whose friendship I greatly value. I thank Yevgeniy Dodis for his sincerity and friendship, and for giving me many technical insights. Many thanks to Alantha Newman for being my friend and for her sincere help and support throughout my time at MIT. I am grateful to Tal Malkin who befriended me the moment that I arrived. Eric Lehman deserves special thanks for being my friend, as well as the Theory Jeopardy Tsar. I am grateful to Stas Jarecki, Moses Liskov, Tal Malkin, Chris Peikert, Zully Ramzan, Leo Reyzin, Amit Sahai and Adam Smith for fruitful research collaborations. I am also very grateful to Moses Liskov for sharing in my first teaching experience.

Chocolate greatly helped! Thank you, Be Blackburn, for being the administrative mom of the theory group and for taking care of me! I am also grateful to Marilyn Pierce, the goddess of EECS.

With grief over the events of September 11, 2001 and his tragic death, I thank Danny Lewin, who was my first officemate and took care of me while I was a clueless first-year.

My two summers at IBM Zürich (1999 and 2000) were unforgettable. I am very grateful to Christian Cachin for making this experience possible, and to him, Victor Shoup and Jan Camenisch for hosting me there. Discussions with Victor have benefitted me greatly, and I am very grateful to him. Jan has been a great collaborator and a wonderful friend. The work I did jointly with Jan makes up a substantial part of my research experience, and part of this thesis is based on it. It has been a great pleasure to work with him. I am also happy to have collaborated with Klaus Kursawe.

I am very grateful to the IBM T.J.Watson's Crypto group who hosted me during the summer of 2001. My thanks go to Tal Rabin, as well as Ran Canetti, Rosario

Gennaro, Shai Halevi, Nick Howgrave-Graham, and Yehuda Lindell, for giving me such a fun experience and for working with me on research problems. Part of this thesis is based on joint work with Tal and Yehuda.

Many thanks to Peter Winkler, my mentor at Bell Laboratories, where I worked for two very happy summers (1997 and 1998), for teaching me to work on research problems, for advising me about life, and for all the great stories and beautiful puzzles he told me.

I am very grateful to my Smith College undergraduate advisors, Joe O'Rourke and Phyllis Cassidy, for sending me on this path. I worked on my first research project under Joe's supervision, and I would like to thank him for his encouragement and enthusiasm.

I would like to thank my roommates, Maja Razlog and Alice Wang, for putting up with me all this time and for being my closest friends.

My love and gratitude to my parents, Nadezhda and Alexander Lysyansky, for everything-everything-everything.

(Funding for my graduate studies has been provided by an NSF graduate research fellowship, Lucent Technologies graduate research program for women, and DARPA, Merrill Lynch and NTT grants given to Ron Rivest.)

Contents

Chapter 1

Introduction

In a broad practical sense, cryptography is the study of securing communication and computation against malicious adversaries. Two indispensable sides of this study are ensuring that no adversary will be able to disrupt transactions carried out by honest parties, and ensuring that no adversary will learn any of the private information held by the honest parties.

For transactions carried out in today's electronic world, the scenario with the adversary is, unfortunately, the right scenario. An electronic message passes through a number of routers on its way from sender to recipient, and some of these routers may be controlled by malicious parties, and may run malicious code. They may alter the message while it is in transit. They may leave it intact, but nothing prevents them from reading it. Hundreds of thousands of credits card numbers stolen in this fashion are evidence to this.

Hence, we cannot take integrity and privacy of communication for granted if we are communicating over the Internet. But even once we have solved the problem of ensuring the security of the network over which our sender and recipient communicate, there are still questions such as: Do they trust each other? Is there a chance that one of them is behaving maliciously, and what damage can it cause the other party?

Suppose that the two parties are a user who is buying some goods, and a vendor selling them. The issue at stake for the vendor is to ensure that he will be paid properly for the goods he delivers. The issue at stake for the user is that she receives the goods paid for.

A privacy-conscious user would also care that the vendor be discreet with the information he obtained about her. This may be in contradiction with the vendor's intent to sell the names of his customers to third parties, or to contribute them to statistical surveys that help analyze customer behavior.[1] And even if the vendor declares that he has no such intent, can the user trust him? What if the vendor goes out of business and all of his assets, including his sales records, are auctioned to third parties who made no such declaration? What if the vendor's computer is broken into?

[1]Note that just because the vendor is able to ship goods to the user doesn't mean he knows the user's real name and home address. The user can, for example, ask the mail sent to a special post office under an assumed name.

The reason that a user should be more privacy-conscious in the context of electronic transactions as opposed to the transactions carried out in person, is that finding patterns in digitally stored data is much easier than connecting the dots in the paper-based world. We do not wear masks when we go to the supermarket, even though some of the things we buy there may be of a very private nature. Provided we do not run into someone we know, we can be reasonably sure that no one will ever find out that a transaction has taken place. (Although security cameras in the store may record enough information to make such faith ill-advised. Additionally, many shoppers use discount cards and thus give up their privacy in order to get a lower prices on purchases.) But on-line transactions, if they are stored in a vendor's database, may come back to haunt the user later. One database may be combined with another, and piece by piece a whole pattern may emerge, a picture this privacy-conscious user does not want the world to see.

This side of the scenario becomes more complex once, for example, the vendor is a pharmacist and needs to verify that his customer holds a valid prescription for the medication being sold. How do we balance the requirement that the pharmacist be able to verify that the user has the right prescription, with the desire of the user to maintain his or her privacy? Can the customer remain anonymous and yet present the proof that he has a valid prescription?

In this thesis, we address the question of ensuring data integrity. We also show how to ensure that verifiable personal information can be provided without giving up the privacy of individuals.

1.1 Digital Signature Schemes

The problem of ensuring integrity of data in a network such as the Internet is very complex. It consists of ensuring that the hardware components, such as routers, are able to handle a large volume of data; of making sure that this data reaches the destination even if some hardware components fail; and of making sure that even if part of the network is exhibiting adversarial behavior, the worst thing that can happen is failure to deliver the data. The design of reliable computer networks presents a compendium of challenging problems that have fascinated researchers in computer science, electrical engineering and mathematics alike for the past few decades, and are sure to continue to do so. Here, we will worry only about the third problem, i.e., what happens as data passes through adversarial hands.

In their seminal paper "New Directions in Cryptography" [DH76] Whitfield Diffie and Martin Hellman invented public-key cryptography and, in particular, digital signature schemes. A digital signature scheme mirrors the paper-based idea of a signature. In it, every user has an identity, represented by her *public key*, i.e., a publicly available string of bits. The user holds the corresponding *secret key*, i.e., a string of bits that is available to the user only. A signature on a message m can be computed using the secret key, and can be verified using the public key. Moreover, unless the message m was signed by the user, no adversary can produce a valid signature on this message, even if the user can be convinced to sign other, possibly related, messages of the adversary's choice.

In the above purchasing scenario, signatures protect the customer and the vendor. The vendor can promise to ship the goods upon receipt of the money, and provided that there are some means to check whether the vendor has received the money and whether the customer has received the goods, it is possible to carry out electronic transactions with the same or even greater confidence as transactions conducted in person, by mail or by telephone.

Of course, the use of signatures is pervasive not only in commerce, but in all forms of official transactions. With digital signatures, all such transactions can be carried out electronically. Just as paper-based signatures, digital signatures provide two guarantees: (1) the guarantee that a given piece of data came from the right person and remained intact in the transmission, even if it passed through adversarial hands; and (2) the guarantee that the recipient of signed data can prove to a third party that he received this data from the signer. Digital signatures have been recognized by the United States' and other governments as legally binding [2].

Digital signature schemes exist if and only if one-way functions exist [Rom90]. One-way functions are efficiently computable functions that are computationally infeasible to invert. For example, since no efficient integer factorization algorithm is known, and it is conjectured that one does not exist, integer multiplication is conjectured to be a one-way function[3]. At this point, it is still unknown whether one-way functions exist. However, it is widely conjectured that they do.

The first digital signature scheme was constructed by Rivest, Shamir and Adleman [RSA78], in the paper that also proposed the first public-key cryptosystem and paved the way for further study of cryptography. Their signature scheme is based on the assumption that they introduced, called "the RSA assumption." By now, the RSA assumption has become a standard cryptographic assumption. Early work on signatures was also carried out by Lamport [Lam79], Merkle [Mer90], and Rabin [Rab79].

Goldwasser, Micali, and Rivest [GMR88] gave the rigorous definition of security for signature schemes and provided the first construction that provably satisfied that definition, under a suitable assumption (namely, the assumption that claw-free pairs of permutations exist, which is implied by the assumption that integer factorization is hard). The earlier signature scheme due to Merkle [Mer90] was also shown to satisfy this definition. Rompel [Rom90] (see also Naor and Yung [NY89]) showed how to construct a digital signature scheme using any one-way function. The inefficiency of the above-mentioned constructions, and also the fact that these signature schemes require the signer's secret key to change between invocations of the signing algorithm, make these solutions less practical than desired.

[2]`http://usinfo.state.gov/topical/global/ecom/00063001.htm`

[3]To be precise, multiplication is conjectured to be a *weak* one-way function. A weak one-way function is a function that is hard to invert for a fraction of inputs, rather than all inputs. It has been shown that from weak one-way functions, one-way functions in the usual sense can be constructed [Yao82, Gol01].

Several more signature schemes have been shown to be secure in the so-called random-oracle model. The random-oracle model is a model of computation in which it is assumed that a given hash function behaves as an ideal random function. (An ideal random function is a function where the input domain is the set of all binary strings, such that each binary string is mapped to a random binary string of the same length.) Although this assumption is evidently false, it formalizes the notion that the hash function behaves essentially as a black box and its output cannot be determined until it is evaluated. It is considered that a proof of security in the random-oracle model gives some evidence of resilience to attack. The RSA [RSA78] signatures with special message formatting [BR93], the Fiat-Shamir [FS87], and the Schnorr [Sch91] signature schemes have been analyzed and proven secure in the random-oracle model. However, it is known [CGH98] that security in the random-oracle model does not imply security in the plain model.

Gennaro, Halevi, and Rabin [GHR99] and Cramer and Shoup [CS99] proposed the first signature schemes whose efficiency is suitable for practical use and whose security analysis does not assume an ideal random function. Their schemes are secure under the so-called *Strong RSA assumption.*

1.1.1 Use of Signatures in Protocols

In all of the work on signature schemes cited above, signature schemes were considered for use as a stand-alone application. However, signature schemes are used in numerous other cryptographic protocols, and it is important to design them with the broadest possible uses in mind.

Consider the use of signature schemes for constructing an anonymous credential system. While such a system can be constructed from any signature scheme using general techniques for cryptographic protocol design [LRSW99], doing it in this fashion is very inefficient. Let us explain this point in more detail.

In a credential system, a user can obtain access to a resource only by presenting a credential that demonstrates that he is authorized to do so. In the paper-based world, examples of such credentials are passports that allow us to prove citizenship, voter registration cards that authorize us to vote, driver's licenses that prove our authorization to drive cars, etc. In the digital world, it is reasonable to imagine that such a credential will be in the form of a digital signature. Let us imagine that each user has an identity *ID*. Let us think of a credential as a signature on the user's identity and possibly some other information.

A credential system is anonymous if it allows users to demonstrate such credentials without revealing any additional information about their identities. In essence, when the user shows up before the verifier and demonstrates that he has a credential, the verifier can infer nothing about who the user is other than that the user has the right credential. Additionally, an anonymous credential system allows the user to obtain a credential anonymously. Such systems were first envisioned by David Chaum [Cha85], and have been further studied by Chaum and Evertse [CE87], Brands [Bra99], and Lysyanskaya, Rivest, Sahai, and Wolf [LRSW99].

Using general techniques of zero-knowledge proofs and zero-knowledge proofs of knowledge [GMR85, GMR89, GMW86, GMW87b, BG92] it is possible to prove statements such as "I have a signature," without saying anything more than that (i.e., without disclosing what this credential looks like or the identity *ID* to which it was issued). However, doing so requires that the problem at hand be represented as, for example, a Boolean circuit, and then the proof that the statement is true requires a proof that the circuit has a satisfying assignment. This method requires expensive computations beyond what is considered practical.

An additional complication is obtaining credentials in a secure way. The simple solution where the user reveals his identity *ID* to the credential granting organization, who in turn grants him the credential, is ruled out: we want to allow the user to be anonymous when obtaining credentials as well; we also want to make it possible to combat identity fraud, and in such a case, *ID* should never become known to anyone other than the user. (We further discuss the notion of identity in the electronic setting in Section 1.2.2.)

Here, general techniques of secure two-party computation [Yao86, GMW87a, Gol98, Lin01] save the day: the user and the organization can use a secure two-party protocol such that the user's output is a signature on his identity, while the organization learns nothing. But this is also very expensive: general secure two-party computation also represents the function to be computed as a Boolean circuit, and then proceeds to evaluate it gate by gate.

So far, we have described a construction of an anonymous credential system from a signature scheme using zero-knowledge proofs of knowledge and general two-party computation. We have also observed that this solution is not satisfactory as far as efficiency is concerned. A natural question therefore is whether we can design a signature scheme that will easily yield itself to an efficient construction of an anonymous credential system. In other words, we need signature schemes for which proving a statement such as "I have a signature," is a much more efficient operation than representing this statement as a circuit and proving something about such a circuit. We also need to enable the user to obtain a signature on his identity without compromising his identity.

In this thesis, we present signature schemes that meet these very general requirements. The generality of this approach enables the use of this signature scheme for other cryptographic protocols in which it is desirable to prove statements of the form "I have a signature," and to obtain signatures on committed values.

Under the Strong RSA assumption, we construct a signature scheme which is efficient and for which there are efficient zero-knowledge proofs of knowledge. In turn, this is of use in cryptographic protocols in general, and in anonymous credential systems in particular. We present this signature scheme in Chapter 4. We describe how to incorporate such a system to obtain an anonymous credential system in Chapter 3.

We also describe how to use this signature scheme to implement an unlinkable electronic cash scheme. An electronic cash scheme consists of a bank, a set of customers, and a set of vendors. Each user and vendor has a bank account. There are three protocols: for withdrawal (between the bank and a customer), spending (between the customer and the vendor), and depositing (between the bank and the vendor). As a result of the withdrawal protocol, the customer obtains a banknote, and his account balance is decreased by the appropriate amount. During the spending protocol, the banknote is given to the vendor (presumably in exchange for some goods that the vendor sells), and during the depositing protocol, the bank increments the vendor's account by the appropriate amount. An electronic cash scheme is *unlinkable* if, even if they cooperate, the bank and the vendor cannot link transactions carried out by the same customer. Unlinkable electronic cash was invented by David Chaum [Cha85].

Let us now return to our example of a pharmacist trying to verify that his customer has a valid prescription without learning any other information about the customer. Using our anonymous credential system, the customer can prove that he has a prescription. He can then pay for his medication using electronic cash.

1.1.2 Revocation of Signatures

Another important aspect of the use of signatures is their revocation. In the paper-based world, it is often possible to make a document no longer valid. For example, a driver's license can be revoked.

One way of doing so would be through black lists. This requires consulting a large database of revoked signatures every time a signature is verified. Consider the situation where proving possession of a signature is done without actually showing the signature itself or revealing any information about it, such as in an anonymous credential system described above. Having the database of revoked signatures as an input to the signature verification protocol renders the running time of this protocol dependent on the number of revoked signatures.

Another way of making sure that all issued signatures are valid, is to re-issue signatures frequently, and not to reissue signatures on documents that are no longer valid. The problem with this solution is that an operation, linear in the number of valid documents, has to be carried out frequently.

In Chapter 5, we describe a solution to this problem. Our solution is based on a generalization of one-way accumulators. One-way accumulators were introduced by Benaloh and de Mare [BdM94], as a way to combine a set of values into one short accumulator, such that there is a short witness that a given value was incorporated into the accumulator. At the same time, it is infeasible to find a witness for a value that was not accumulated. Extending the ideas due to Benaloh and de Mare [BdM94], Barić and Pfitzmann [BP97] give an efficient construction of so-called collision-resistant accumulators, based on the strong RSA assumption.

We propose a variant of the cited construction with the additional advantage that, using additional trapdoor information, the running time of deleting a value from an accumulator is independent of the number of accumulated values, and depends only on the security parameter. Better still, once the accumulator is updated, updating the witness that a given value is in the accumulator (provided that this value has not been revoked, of course!) can be done without the trapdoor information at unit cost. Accumulators with these properties are called *dynamic*.

In order to use dynamic accumulators for revocation of signatures, we propose that each signature issued should have some unique identifier that will be incorporated into the dynamic accumulator at the time of issue. Once this signature is revoked, this identifier will be deleted from the accumulator. In order to efficiently demonstrate a statement such as "I have a signature on my identity that is still valid," it is necessary to (1) prove a statement of the form "I have a signature on my identity;" and (2) prove the statement "The identifier of the signature on my identity is in the accumulator." We provide efficient protocols for such proofs.

1.1.3 Unique Signatures

Unique signature schemes are signature schemes where corresponding to each acceptable public key PK and message m, there is a unique string σ that the verification algorithm will accept as a valid signature on message m under public key PK.

Unique signatures are of interest for several reasons. As a stand-alone application, unique signatures give a computation/storage trade-off in verifying signatures. Imagine a signature scheme where coming up with a new signature on an already signed message is a more efficient operation than verifying a signature. Such a signature scheme is vulnerable to the following denial-of-service attack on behalf of a signer: The signer will send the same document with different signatures to a server and force the server to do a lot of work by verifying all these signatures. The solution where a server stores a list of signed documents and looks up a document instead of verifying the attached signature, is undesirable because it may lead to acceptance of a document from an unauthorized party. Just because a document has been signed in the past, does not mean that it ought to be accepted if received from a party that fails to produce a valid signature. On the other hand, using a unique signature scheme and a lot of storage, this problem can be solved by having the server store all the documents and the signatures it has verified in the past, and upon receipt of a document, check if it is already stored. If so, instead of performing a separate computation to verify the attached signature on the document, compare the attached signature to the one stored. If they are different, reject.

Unique signature schemes are also related to verifiable random functions [MRV99] and non-interactive zero knowledge proofs [GO92], and as a result are valuable tools in the design of cryptographic protocols. Verifiable random functions (VRFs) are cryptographic objects that produce random-looking bits, and yet there is a mechanism to convince a third party that all these bits came from the same source. This has proved useful in the design of resettable zero-knowledge proof protocols [MR01, Rey01].

Another interesting application of verifiable random functions, due to Micali and Rivest [MR02], is their use in non-interactive lottery systems that in turn are used for micropayments. Here, the lottery organizer holds a public key PK of a VRF. A participant creates his lottery ticket t himself and sends it to the organizer. The organizer computes the value of the verifiable random function on the lottery ticket, and the corresponding proof. The value of the function determines whether the user wins, while the proof guarantees that the organizer cannot cheat. Since a VRF is hard to predict, the user has no idea how to bias the lottery in his favor. In order to guarantee fairness of the lottery, a user can supply an additional random value to be XORed with the value of the VRF.

In this thesis, we give a new construction of a unique signature scheme, and related VRF constructions. Chapter 6 is dedicated to this theme.

1.1.4 Other Applications of Signatures in Protocols

With the use of cryptography, and digital signature schemes in particular, tasks that are impossible in the plain model become feasible. One example of this phenomenon was discovered by Pease, Shostack, and Lamport [PSL80] and Lamport, Shostack and Pease [LSP82], in the context of Byzantine agreement protocols.

Byzantine agreement is the problem of implementing a broadcast channel in a point-to-point network where some nodes behave maliciously. This is of great importance for protocol design: it is much more convenient to design protocols for the broadcast model than for the point-to-point network.

Pease, Lamport, and Shostack [PSL80] and Lamport, Shostack and Pease [LSP82] showed that Byzantine agreement was infeasible if the number of malicious nodes is at least one-third of the total number of nodes. They also showed that using a signature schemes and a public-key infrastructure, any number of malicious nodes can be tolerated.

Although this is a natural application of signature schemes, it was not previously well-studied. In this thesis, we consider the problem of running Byzantine agreement more than once. It turns out that signatures do not always help in case that the Byzantine agreement is invoked more than once without session identifiers. Specifically, if several Byzantine agreement protocols are executed concurrently without appropriate session identifiers for each invocation, then this protocol cannot tolerate one third or more malicious nodes. The same impossibility applies to sequential composition of deterministic Byzantine agreement protocols. However, there is a randomized Byzantine agreement protocol that composes sequentially without session identifiers.

Finally, appropriately chosen session identifiers enable composition of Byzantine agreement for any number of malicious nodes.

These results illustrate both the power and the limitations of the use of signatures in the design of distributed algorithms. They are exhibited in Chapter 7.

1.2 Privacy of Data

Several applications described in this thesis, such as anonymous credential systems, efficient revocation of anonymous credentials, and electronic cash, are aimed at protecting personal information in on-line transactions and giving users control over what they do and do not disclose about themselves. Let us discuss this topic in a broader context.

Ensuring that information of personal nature does not become public knowledge has become an increasing concern a few decades ago as computer databases became widely used. In the past, when most recorded information required a considerable human effort to retrieve, putting together two unrelated pieces of information and linking them to the same individual was not an easy task. As a result, even if virtually all personal information was at some point or other recorded and stored in a public archive, one did not need to worry that someone would be able easily to put it all together to get a complete picture of one's the private affairs. Clearly, in the age of databases, this is no longer the case.

According to Simson Garfinkel's book *Database nation* [Gar00], the U.S. government recognized that this was a problem and created an executive committee on this issue. The contribution of this committee was "a bill of rights for the computer age ... called the Code of Fair Information Practices."

A natural question, raised by David Brin in his book *Transparent Society* [Bri98], is whether it is even possible to protect the privacy of individuals given the surveillance technology increasingly available today. While Brin makes a strong case for making sure that there are public mechanisms for monitoring all the information collected by the government, it still requires a leap of faith to conclude that *all* information about *all* individuals is necessarily collectable, and that a trade-off cannot be found that would satisfy both the public's needs for safety, and the individual's need for privacy. Just because there is technology to invade people's privacy, does not mean that this technology will be used in order to do so.

There are several approaches to the problem of protecting privacy. On the one hand, there is the approach of defining exactly what privacy is in the society one lives in, how much privacy an individual is entitled to (for example, maybe it is all right to buy a prescription medication in private, but driving incognito should not be allowed: should an accident happen, the driver should be accountable for it), and introduce legislation that enforces this level of individual privacy (so a pharmacist selling goods on-line will erase the user's name and address as soon as the medication is shipped). On the other hand, one can approach the problem by providing a cryptographic solution which, even in case some of the players are malicious and act illegally, still protects the honest parties involved.

To ensure privacy, both a legislative and a technological effort is required. Let us briefly argue why.

Privacy can only exist in a society that recognizes it as a right. In a totalitarian state, where everything is the government's business, not simply as a law, but as a way of living, a mentality, there can be no such thing. With or without modern technology, totalitarian states such as Tsarist Russia and the Soviet Union that succeeded it, have been successful in carrying out total surveillance over their subjects. In fact, as early as the middle of the nineteenth century, the Russian secret police was well capable of surveillance over whomever it found worthy of such attention. Mikhail Lermontov, one of the key Russian poets and intellectuals of that age, wrote in 1841 of the Russian secret police as having an "all-seeing eye" (as a Cyclope-like monster) and "all-hearing ears." A modern Russian intellectual, Alexander Solzhenitsyn described some blood-chilling, but still primitive on the technological level, solutions to the total surveillance problem, — those deployed by Stalin — in his novel *The First Circle.*

Although intellectuals found this state of affairs stifling, it was known and accepted as a fact of life that every person was watched. It is no surprise, then, that the Russian language does not contain an adequate translation for the very word "privacy."

On the other hand, in a free society, personal privacy is of supreme importance. It is no coincidence that George Orwell's *1984*, as opposed to books on monsters and vampires, is the ultimate horror novel. A feeling of being watched, and of one's past being always on record, is repugnant in a society that is based on the principle that people ought to be free to say what they think and do as they like, without subjecting themselves to the scrutiny and interference of others.

However, mere respect for privacy without a set of liabilities attached to violating it, is not a sufficient guarantee. This is because an organization can respect its customers' privacy and not want to generally disclose their personal records, but not object to using this information when there is something to be gained by using it. Examples are selling customer data to telemarketers, or aggregating this data for statistical purposes and publishing the results. Many people found such practices disturbing, and as a result, privacy regulations have emerged both in the United States and in the European Union, and policy with respect to privacy in electronic transactions continues to be a subject of discussion for legal scholars around the world.

Garfinkel's book *Database Nation* does an outstanding job of bringing attention to the problem of privacy in the electronic world. He brings attention to some fundamental flaws in practices widely deployed today, and suggests a number of approaches that legislators should consider.

However, a legally enforced commitment to privacy implies increased liabilities for organizations. No matter how much a given organization strives to protect the privacy of its customers, it may not be aware that someone broke into its database; it may also be vulnerable to insider attacks. Thus, it is desirable to adopt a privacy-enhancing technology that gives such an organization the opportunity to learn as little about others' personal information as is at all possible. That is to say, a technology that would enable an organization to learn only what it *needs* to learn, and to make sure that no other information is transmitted to the organization.

Another reason for privacy-enhancing technologies is that legal standards for privacy are not uniform throughout the world; yet people expect the same level of privacy no matter where the other party is located.

1.2.1 Review of Existing Technologies

The most basic privacy-enhancing technology deployed today, is the P3P protocol introduced by the World-Wide-Web Consortium (`http://www.w3c.org/p3p`). In a nutshell, this technology enables the client and server to negotiate a privacy level that both parties are comfortable with; this is done transparently to the user. The idea is that a user's privacy preferences are specified in a standard form and given to his browser. The server's privacy policy is written in a form that the user's browser can understand and compare to the user's preferences, determining whether the two are compatible, and if so, negotiate the privacy terms in compliance with both policies.

It is evident that while P3P is a desirable technology, it is only effective in situations when a server in question honestly declares what data it is going to collect and what it intends to do with the data obtained. A user may or may not believe it. Moreover, many users are not aware of P3P and unwittingly waive away their privacy. Thus, P3P is powerless against malicious attacks.

Another available technology is anonymizing services such as the ones provided by Anonymizer.com. A user of such a service can browse the Web in such a way that the server accessed cannot trace this user. In other words, such a service provides users with anonymous channels. For on-line privacy, a service such as this is vital. Without it, information about all of a given user's on-line activities can be retrieved by examining the log files of this user's Internet service providers. However, anonymous channels alone are not sufficient in order to effectively work with organizations that require that their customers have some credentials. An anonymous credential system is also necessary in such a case.

Anonymous credential systems were suggested by David Chaum [Cha85]. The first proposed solution was a proof of concept and involved a semi-trusted third party [CE87]. Later, Damgård [Dam90] and Chen [Che95] worked on the problem. Their solutions did not have a trusted third party, but the notion of user's identity was not well-developed, and therefore it was not clear how their systems could function in case malicious users shared their credentials with others. Both of these solution were not satisfactory from the point of view of efficiency, while Chen's solution also lacked a rigorous definition and proof of security.

Stefan Brands [Bra99] worked on the problem of making possible a public-key infrastructure (PKI) where encoded into each user's public keys would be some attributes. For example, attributes could encode this person's age, level of education, etc. A holder of a public key could choose which attributes to disclose; additionally, it is possible to disclose any one of a rich set of algebraic relations on these attributes. Brands proposes solutions based on the discrete-logarithm problem and its variants.

While Brands' work developed a set of powerful techniques and lay a foundation for privacy-enhancing public-key infrastructures, it only gives a solution for a special case of the general problem of anonymous credential systems.

As in any public-key infrastructure, Brands' PKI requires that a certification authority (CA) certify that public keys belong to valid users. In Brands' work, the CA is the *only* credential-granting organization.

Second, if a user talks to two different entities and wants to prevent them from discovering that they are talking to the same person, he needs to have two separate, unlinkable public keys. That requires that the CA certify both of these keys. Therefore, both the communication and the computational load on the CA is linear in the total number of unlinkable transactions taking place in the entire public-key infrastructure.

1.2.2 Identity and Anonymity in the Electronic Setting

Above, we mentioned that it is desirable a user's identity *ID* should remain secret to prevent identity fraud. Let us now clarify what is meant by an identity and why, if the value *ID* is leaked, identity fraud may result.

In an electronic setting, it is difficult to put a finger on what exactly an identity is. Is any given computer terminal an identity? Or any given e-mail address? Or IP address?

Let us analyze the notion of an identity as it relates to credentials. In a pragmatic sense, a credential is issued to whomever it is that can later succeed in demonstrating ownership of this credential. Therefore, the means that enable such a demonstration define the identity of the credential owner.

For example, to prove that I may drive, I can demonstrate my driver's license which has my picture on it. Therefore, it is my picture, and also the fact that I have my driver's license in my wallet, that define that it is I who can prove possession of this driver's license, and therefore that the license was issued to me rather than anyone else.

In general, in the electronic setting, the only means available for demonstrating anything is the data stored on one's machine, be it on the hard drive or on a special dedicated device such as a smartcard. Therefore, the data stored on one's machine defines one's electronic identity, and leaking this data can result in identity fraud. Moreover, unless by some means this data can be implanted into humans, electronic identity will not correspond to physical identity of people, since several people may have access to a given machine, and also because electronic data can (almost always) be duplicated.

It is therefore evident that the identity to which a credential is issued can be defined by a string of bits. However, the contents of any machine changes over time. Therefore, this string of bits should not be the entire contents of this user's machine, but simply a string that is sufficiently long to be unique to this machine and to be hard to guess. This is true whether the electronic credentials are anonymous or not. What is challenging in this context is discouraging users from making their credentials available to others.

In order to link an electronic identity *ID* to a publicly recognizable entity such as a person, an organization, or a pseudonym, some function $f(ID)$ is published and certified by a corresponding certification authority as corresponding to a person's physical identity.

An example of the above general scenario is a public-key infrastructure. In it, a user's *ID* is his secret key *SK*. The corresponding public key *PK* is published, and the certification authority binds it to some publicly recognizable entity, such as this user's name. The user is the owner of this public key *PK* because the corresponding secret key *SK* is stored on his machine. This is what enables the user to read mail that is encrypted with this *PK* and to issue signatures under this *PK*.

In the context of a public-key infrastructure, discouraging users from making their credentials available to others is simpler than in other contexts. It was suggested by

Dwork, Lotspiech and Naor [DLN96] that making *SK* too valuable to give away, and necessary for proving possession of a credential, makes the user unwilling to share this credential.

How can *SK* be made too valuable to give away? For example, by having a physical liability attached to disclosing it. Having to pay a lot of money should one's secret key *SK* become known to the police would be an incentive.

Without such external means, it is possible to discourage sharing the string *ID* by making all of one's credentials retrievable using this string [CL01].

1.3 Summary of This Thesis

As stated earlier, cryptography helps make sure that electronic transactions bring the integrity and privacy that individuals expect them to, and signature schemes are instrumental in achieving that goal. In this thesis, we consider the uses of signature schemes in the context of cryptographic protocols.

In Chapter 2 we introduce notation, and recall definitions for digital signature schemes, commitment schemes, zero-knowledge proofs and proofs of knowledge, as well as describe state-of-the-art on the composition of these protocols.

Next, in Chapters 3, 4 and 5, we turn to the use of signature schemes for anonymous credential systems. In Chapter 3 we define an anonymous credential system and show how to construct it given an appropriate signature scheme. In Chapter 4 we give an appropriate signature scheme. Chapter 5 gives mechanisms for revocation of anonymous credentials. Preliminary versions of the credential system and and revocation mechanisms presented here have been previously published [CL01, CL02].

Finally, Chapter 6 gives a new construction of unique signatures, and Chapter 7 explores the limits of applicability of signature schemes in protocols. Preliminary versions of these have also been previously published [Lys02, LLR02].

Chapter 2

Preliminaries

This chapter introduces the notation and definitions of security for such cryptographic constructions as signature schemes, commitment schemes, zero-knowledge proofs and proofs of knowledge. The aim of this chapter is to make this thesis self-contained. A far better treatment of this subject can be found in Oded Goldreich's book "Foundations of Cryptography"[Gol01].

2.1 Notation

Let A be an algorithm. By $A(\cdot)$ we denote that A has one input (resp., by $A(\cdot, \ldots, \cdot)$ we denote that A has several inputs). By $A_{(\cdot)}$ we will denote that A is an indexed family of algorithms.

$y \leftarrow A(x)$ denotes that y was obtained by running A on input x. In case A is deterministic, then this y is unique; if A is probabilistic, then y is a random variable. If S is a finite set, then $y \leftarrow S$ denotes that y was chosen from S uniformly at random.

Let A and B be interactive Turing machines. By $(a \leftarrow A(\cdot) \leftrightarrow B(\cdot) \rightarrow b)$, we denote that a and b are random variables that correspond to the outputs of A and B as a result of their joint computation. Alternatively, we will use the notation $OUT_A(A(\cdot) \leftrightarrow B(\cdot))$ for the output of A. We will also use the notation $VIEW_A(A(\cdot) \leftrightarrow B(\cdot))$ to denote the entire computation history of A, including its input, the contents of its random tape, and all the messages A received from B. In case of more than two players, we will also use the notation OUT and $VIEW$ in the analogous fashion. We will sometimes use notation such as $[VIEW_A, OUT_B](A \leftrightarrow B)$ to denote the joint distribution on the view of A and output of B.

Let b be a boolean function. The notation $(y \leftarrow A(x) : b(y))$ denotes the event that $b(y)$ is true after y was generated by running A on input x.

The statement

$$\Pr[\{x_i \leftarrow A_i(y_i)\}_{1 \le i \le n} \ : \ b(x_n)] = \alpha$$

means that the probability that $b(x_n)$ is TRUE after the value x_n was obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$, is α, where the probability is over the random choices of the probabilistic algorithms involved.

By $A^{(\cdot)}(\cdot)$, we denote a Turing machine that makes oracle queries. I.e., this machine will have an additional (read/write-once) query tape, on which it will write its queries in binary; once it is done writing a query, it inserts a special symbol "#". By external means, once the symbol "#" appears on the query tape, an oracle is invoked and its answer appears on the query tape adjacent to the "#" symbol. By

$$Q = Q(A^O(x)) \leftarrow A^O(x)$$

we denote the contents of the query tape once A terminates, with oracle O and input x. By $(q, a) \in Q$ we denote the event that q was a query issued by A, and a was the answer received from oracle O.

By $A \overset{\blacksquare}{\rightarrow} B$ we denote that algorithm A has black-box access to algorithm B, i.e. it can invoke B with an arbitrary input and reset B's state.

We say that $\nu(k)$ is a negligible function, if for all polynomials $p(k)$, for all sufficiently large k, $\nu(k) < 1/p(k)$.

Let $A(1^k)$ and $B(1^k)$ be two probabilistic algorithms. By $A \overset{c}{\equiv} B$ we denote that for all probabilistic polynomial-time families of adversaries $\{\mathcal{A}_k\}$, there exists a negligible function ν such that for all k,

$$\Pr[x_0 \leftarrow A(1^k); x_1 \leftarrow B(1^k); b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}_{(x_b)} \ : \ b = b'] = \nu(k)$$

2.2 Definitions of Security for Signature Schemes

The following definition is due to Goldwasser, Micali, and Rivest [GMR88], and has become the standard definition of security for signature schemes. Schemes that satisfy it are also known as signature schemes secure against *adaptive chosen-message attack.*

Definition 2.2.1 (Signature scheme). *Probabilistic polynomial-time algorithms* $(G(\cdot), \texttt{Sign}_{(\cdot)}(\cdot), \texttt{Verify}_{(\cdot)}(\cdot,\cdot))$, *where G is the key generation algorithm,* `Sign` *is the signature algorithm, and* `Verify` *the verification algorithm, constitute a digital signature scheme for a family (indexed by the public key PK) of message spaces $\mathcal{M}_{(\cdot)}$ if:*

Correctness *If a message m is in the message space for a given public key PK, and SK is the corresponding secret key, then the output of* $\texttt{Sign}_{SK}(m)$ *will always be accepted by the verification algorithm* $\texttt{Verify}_{PK}$. *More formally, for all values m and k:*

$$\Pr[(PK, SK) \leftarrow G(1^k); \sigma \leftarrow \texttt{Sign}_{SK}(m) \ : \ m \in \mathcal{M}_{PK} \wedge \neg\texttt{Verify}_{PK}(m,\sigma)] = 0$$

Security *Even if an adversary has oracle access to the signing algorithm which provides signatures on messages of the adversary's choice, the adversary cannot create a valid signature on a message not explicitly queried. More formally,* *for all families of probabilistic polynomial-time oracle Turing machines $\{A_k^{(\cdot)}\}$, there exists a negligible function $\nu(k)$ such that*

$$\begin{aligned}\Pr[(PK, SK) \leftarrow G(1^k); (Q, x, \sigma) \leftarrow A_k^{\mathtt{Sign}_{SK}(\cdot)}(1^k) :& \\ \mathtt{Verify}_{PK}(m, \sigma) = 1 \wedge \neg(\exists \sigma' \mid (m, \sigma) \in Q)] &= \nu(k)\end{aligned}$$

2.3 Commitment Schemes

A commitment scheme is a two-party protocol between a committer and a receiver. It consists of two stages: the *Commit* stage and the *Reveal* stage. The committer receives a value x as input to the *Commit* stage, and reveals this value x to the receiver at the *Reveal* stage. A protocol is a secure commitment scheme if at the end of the *Commit* stage the receiver cannot compute anything about the committed value, and there exists only one value that the committer can reveal at the *Reveal* stage.

Commitment schemes sometimes have a pre-processing phase during which the parameters of the commitment scheme are set up.

For the purposes of this thesis, we will consider a less general class of commitments only, namely the class of non-interactive commitments. Sometimes, it is desirable to reduce the amount of communication needed in a commitment scheme to one round per stage.

Commitment are important building blocks for essentially all cryptographic protocols, from zero-knowledge proofs to multi-party computation. Here, we give a definition of non-interactive commitment schemes. In general, a commitment scheme need not be interactive. However, for the constructions we give in the sequel, there is no need for the more general definition.

A non-interactive commitment scheme consists of a set-up algorithm $\mathcal{G}$ that sets up the public parameters for the scheme, and a commitment function `Commit`. $\mathcal{G}$ takes as input just the security parameter 1^k and produces the public key PK. (Some commitment schemes do not need a set-up phase; for these, we can imagine that $\mathcal{G}$ simply outputs the unique $PK = 1^k$ on input 1^k.) Corresponding to each public key PK, there is a message space $\mathcal{M}_{PK}$. `Commit` is a deterministic algorithm that takes as input the values, PK, $x \in \mathcal{M}_{PK}$, and a k-bit random string r.

In order to commit to a value $x \in \mathcal{M}_{PK}$, the Committer generates a random r, computes $C = \mathtt{Commit}(PK, x, r)$ and sends C to the receiver. In the Reveal stage, the Committer reveals the values x and r, and the Receiver verifiers that $C = \mathtt{Commit}(PK, x, r)$. More formally:

Definition 2.3.1 (Commitment scheme). *Algorithms $(\mathcal{G}, \mathtt{Commit})$ constitute a non-interactive public-key commitment scheme if the key generation algorithm $\mathcal{G}$ runs in probabilistic polynomial time, while* `Commit` *is a polynomial-time computable function that takes as input a value x and a random string R and outputs a commitment C such that the following two properties hold:*

Hiding (a.k.a. semantic security)*: Even if the adversary chose the input distribution D for the value x to which the Committer commits, he learns no more information about this value than what is computable from the input distribution itself. More formally, there exists an efficient simulator algorithm S such that for all adversaries $\mathcal{A}$, there exists a negligible function ν such that*

$$\begin{aligned} \Pr[PK \leftarrow \mathcal{G}(1^k); (u, D) \leftarrow \mathcal{A}(1^k, PK);& \\ r \leftarrow \{0,1\}^k; x \leftarrow D;& \\ C_0 \leftarrow \texttt{Commit}(PK, x, r); C_1 \leftarrow S(PK, D);& \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(u, C_b) &\quad : \quad b' = b] \leq 1/2 + \nu(k) \end{aligned}$$

Binding*: No probabilistic polynomial-time non-uniform adversary $\mathcal{A}$ can open a commitment in two different ways. More formally for all $\mathcal{A}$, there exists a negligible function ν such that*

$$\begin{aligned} \Pr[PK \leftarrow \mathcal{G}(1^k) \quad ; \quad & (x_1, r_1, x_2, r_2) \leftarrow \mathcal{A}(PK) : x_1, x_2 \in \mathcal{M}_{PK} \wedge x_1 \neq x_1 \\ & \wedge \texttt{Commit}(PK, x_1, r_1) = \texttt{Commit}(PK, x_2, r_2)] = \nu(k) \end{aligned}$$

Remark: The definition given above requires semantic security of the committed value. It is a classical result due to Goldwasser and Micali [GM84] that equivalently, we could state this definition as follows: the adversary chooses x_0 and x_1, and receives a commitment to a random one of them. The security of the commitment scheme means that the adversary cannot tell a commitment to which one of them it received any better than by random guessing.

Flavors of commitment schemes Since commitment schemes are of fundamental importance in cryptographic protocols, many special flavors of them have been considered in the literature. Most notably, there are unconditionally hiding (resp., binding) commitments for which the hiding (resp., binding) property described above holds even against a computationally unbounded recipient (resp., committer); non-malleable and mutually independent commitments [DDN00, CKOS01, LLM$^+$01] where multiple players are committers and recipients at the same time; and universally composable commitments [CF01] which are commitments whose security approaches the security attainable with the use of an ideal trusted party.

2.3.1 Trapdoor commitments

For many cryptographic applications, a special flavor of commitment schemes, namely *trapdoor* commitments are very attractive. A trapdoor commitment scheme, only possible for commitments with a public key, are schemes in which corresponding to the public commitment scheme PK, there is a trapdoor key TK. The knowledge of TK allows one to open a commitment in any desirable way. The key generation is carried out in such a way that the committer does not know the value TK, and so a trapdoor commitment can be a secure commitment.

Trapdoor commitments are very useful in cryptographic protocols, especially ones whose definition requires simulation. This is because while an actual party in the computation cannot know the trapdoor key corresponding to a commitment and will therefore be bound by a commitment, the simulator can be given this key TK, thus making it easier to create the view of the adversary. As a result, trapdoor commitments are useful, for example, in zero-knowledge proofs [Dam00], threshold cryptography [JL00], and signature schemes [ST01].

We will now give a formal definition of a trapdoor commitment scheme.

Definition 2.3.2. *A commitment scheme* $(\mathcal{G}, \texttt{Commit})$ *is* trapdoor *if there exists a key generation procedure* $\mathcal{G}'$ *such that*

1. *Procedure* $\mathcal{G}'(1^k)$ *outputs a key pair* (PK, TK).

2. *Let* $\mathcal{G}''$ *be a procedure that runs* $\mathcal{G}'$ *and outputs the* PK*-part of its output only.* $\mathcal{G}(1^k) \overset{c}{\equiv} \mathcal{G}''(1^k)$.

3. *There is a procedure* Alter *such that if* $(PK, TK) \in \mathcal{G}'(1^k)$, *then on input* $x_1, x_2 \in \mathcal{M}_{PK}$, *and* $r_1 \in \{0,1\}^k$, *it outputs* $r_2 \in \{0,1\}^k$ *such that* $(PK, r_1) \overset{c}{\equiv} (PK, r_2)$ *and* $\texttt{Commit}(PK, x_1, r_1) = \texttt{Commit}(PK, x_2, r_2)$.

Remark: There are other notions of trapdoor commitments. A weaker notion would be a commitment where the Alter procedure does not necessarily take as input just any value r_1: r_1 can come from a special distribution, indistinguishable from random. A stronger notion would be one that did not require that Alter takes x_1 and r_1 as input, but instead has Alter take just the values (TK, C, x) and find a value r such that $C = \texttt{Commit}(PK, x, r)$.

2.4 Zero-Knowledge Proofs

Introduced by Goldwasser, Micali and Rackoff [GMR85] in their seminal paper "On the Knowledge Complexity of Interactive Proof Systems," zero-knowledge proofs constitute a central class of cryptographic protocols.

In an interactive proof system, there are two players, the Prover, and the Verifier. The Prover wants to convince the Verifier of the validity of some statement. For example, the statement can be of the form "Formula ϕ is satisfiable," or "Graph G is three-colorable." There are two requirements of a proof system: that it be *complete*, that is to say, if the statement is true, then the Prover should be able to convince the verifier with high probability $1-c$; and that it be *sound*, that is to say, if the statement is false, then no prover should be able to convince the verifier with probability larger than some small value s. Although in general a proof system can be defined for any completeness and soundness so long as $1 - c - s$ is non-negligible, we are only interested in protocols with perfect completeness.

Definition 2.4.1 (Interactive proof system). *A pair of interactive probabilistic Turing machines (P, V) constitute an interactive proof system for language L if:*

Completeness *If $x \in L$, then V always accepts, that is* $\Pr[P(x) \leftrightarrow V(x) \rightarrow b \ : \ b = 1] = 1$.

s-Soundness *For all $x \notin L$ for all provers P^*,* $\Pr[P^* \leftrightarrow V(x) \rightarrow b \ : b = 1] \leq s(|x|)$.

We will be interested mostly in proof systems that have negligible s.

Note that a proof system can only be of interest if the Verifier alone cannot determine whether $x \in L$. Of particular interest are proof systems where the Verifier is probabilistic polynomial-time.

On the other hand, when a protocol is designed for use in practice, it is a very natural question how a Prover can be more computationally powerful than probabilistic polynomial-time. Therefore, it makes sense to consider interactive *arguments*, where a Prover's computational powers are also limited to probabilistic polynomial-time, and the Prover's power comes from the fact that the Prover also takes some auxiliary input. More formally:

Definition 2.4.2 (Interactive argument). *A pair of interactive probabilistic polynomial-time Turing machines (P, V) constitute an interactive argument for language L if:*

Completeness *If $x \in L$, then there exists a witness w such that*

$$\Pr[P(x, w) \leftrightarrow V(x) \rightarrow b \ : \ b = 1] = 1$$

Soundness *For all $x \notin L$, for all probabilistic polynomial-time provers P^*,* $\Pr[P^* \leftrightarrow V(x) \rightarrow b \ : b = 1] \leq s(|x|)$.

It is clear that any language in NP has a trivial interactive argument. For example, if L is the language of all satisfiable formulae, then the Prover's auxiliary input is the satisfying assignment w. To convince the Verifier that the formula ϕ is satisfiable, the Prover can simply give w to the Verifier.

The interesting thing about interactive arguments (and also about interactive proofs) is that it is possible to convince the Verifier that ϕ is satisfiable *without revealing any information about w*. An interactive proof system (resp., argument) with this property is called a *zero-knowledge* proof system (resp., argument).

At the heart of the definition of a zero-knowledge proof system is the following intuition: a proof system is zero-knowledge if, no matter what the verifier's code, anything that this verifier learns as a result of this interaction, it could compute on its own. In other words, for any verifier V there is a simulator S that, without talking to the Prover at all, but just assuming that $x \in L$, computes whatever it is that will induce the Verifier to accept, and the Verifier will not even notice the difference. A natural question is, of course, why should such a proof system be sound - since now the Verifier will believe anything? The power that the simulator S has that the actual prover does not, is that the simulator can run and re-run V as it sees fit. For example, if it issues a message that induces V to reject, it can simply rewind V and issue a different message. The actual Prover does not have that ability, hence the simulator can get away with things that the Prover could not possibly get away with.

Definition 2.4.3 (Simulator). *A probabilistic polynomial-time Turing machine S is called a* simulator *for machine A's interaction with machine B on input x, if there exists a polynomial $p(\cdot)$ such that for all inputs a of length at most $p(|x|)$, $VIEW_A(A(a,x) \leftrightarrow B_x) \overset{c}{\equiv} VIEW_A(A(a,x) \overset{\blacksquare}{\leftarrow} S(1^k,x))$.*

We are now ready to give definitions for two flavors of zero-knowledge. The first if these definitions, Definition 2.4.4, is the most general definition of zero-knowledge proofs. The second definition is that of black-box zero knowledge (Definition 2.4.5). This definition is provably stronger, and perhaps unnecessarily strong, but it is frequently the most convenient for use in protocol design.

Definition 2.4.4 (Zero-knowledge proof system). *A proof system (resp., argument) (P, V) is a zero-knowledge proof system (resp., argument) for language L if for all verifiers V^* there exists a machine S such that for all $x \in L$, S is a simulator (as in Definition 2.4.3) for V^*'s interaction with $P(x)$ (resp., $P(x,w)$) on input x.*

The stronger definition of black-box zero-knowledge proofs is the same, except that it has the quantifiers reversed:

Definition 2.4.5 (Black-box zero knowledge proof system). *A proof system (resp., argument) (P, V) is a black-box zero-knowledge proof system (resp., argument) for language L if there exists a machine S such that for all verifiers V^* for all $x \in L$, S is a simulator (as in Definition 2.4.3) for V^*'s interaction with $P(x)$ (resp., $P(x,w)$) on input x.*

2.5 Proofs of Knowledge

Here we recall a definition of a proof of knowledge protocol. Proofs of knowledge are, as Bellare and Goldreich [BG92] wrote, "one of the many conceptual contributions of the work of Goldwasser, Micali and Rackoff [GMR85]." They were first defined by Feige, Fiat and Shamir [FFS88] and by Tompa and Woll [TW87], and further refined by Bellare and Goldreich [BG92].

A proof of knowledge is a protocol whereby a verifier is convinced that a certain quantity w that satisfies some polynomial-time computable relation R is known to the prover. For example, w can be a satisfying assignment to a formula ϕ in which case $R(\phi, w) = \phi(w)$. It is convenient for the purposes of notation to imagine R as taking two inputs: one known to both the prover and the verifier (in this example, the formula ϕ) and the other w, known to the prover only.

A proof of knowledge can be done trivially by producing w. However, it can also be done in such a way that the verifier learns nothing new about w. A protocol that satisfies this latter condition is called a *zero-knowledge proof of knowledge* protocol.

Roughly, a verifier V is a knowledge verifier for a relation R if for any prover P that induces the verifier to accept, there is a knowledge extractor K that, using V and P, computes w such that $R(x, w)$ is satisfied, where x is the common input. As discussed by Bellare and Goldreich [BG92], the subtleties in formally defining proofs of knowledge arise when it is necessary to demand a dependency of the running time of K on the probability with which V accepts when talking to P. An important quantity in the definition of Bellare and Goldreich, is the knowledge error κ which quantifies how often a cheating prover is allowed to convince the verifier. The definition then demands that for any prover P that convinces the verifier with probability $p > \kappa$, the knowledge extractor computes w in expected time $O(\text{poly}(|x|)/(p - \kappa))$.

The work on defining proofs of knowledge that capture the most intuition is on-going, as new protocol design techniques are discovered. A recent example of this trend is Barak and Lindell's [BL02] non-black-box knowledge extractor following Barak's [Bar01] study of non-black-box zero knowledge protocols. Their non-black-box knowledge extractor, instead of using P just as an oracle, takes the description of P as a separate input. The motivation for defining proofs of knowledge in this fashion is that it yields strictly polynomial time of the knowledge extractor K.

Here, we recall the definition due to Bellare and Goldreich. However, we only consider proofs of knowledge where knowledge error function κ is exponentially small in the length of a witness w. The reason that we insist on this stronger definition is that it gives good composition results, and that the protocols we exhibit satisfy it. Moreover, Bellare and Goldreich show how to construct protocols satisfying this requirement out of weaker proof of knowledge protocols.

When the knowledge error is exponentially small, it is easier to examine how the protocol behaves under composition. The theorem we will show is in fact stronger than what was known before, although it seems to have been used implicitly in the analysis of some cryptographic protocols. In particular we show that the property that a protocol is a proof of knowledge is preserved under concurrent composition.

Definition 2.5.1 (Proof of knowledge (POK)). *Let $R(\cdot, \cdot)$ be a polynomially computable relation. Let $u(x)$ be such that $|w| \leq u(x)$ for all w such that $R(x, w)$ holds, and assume that some such $u(x) = poly(|x|)$ is efficiently computable.*

A verifier V is a knowledge verifier with respect to R if:

Non-triviality*: There exists a prover P such that for all x, w, if $R(x, w) = 1$, then*

$$\Pr[P(x, w) \leftrightarrow V(x) \to b \ : \ b = 1] = 1$$

Extraction with knowledge error $2^{-u(x)}$*: There exists an extractor algorithm K and a constant c such that for all x, for all adversaries $\mathcal{A}$, if*

$$p(x) = \Pr[\mathcal{A} \leftrightarrow V(x) \to b \ : \ b = 1] > 2^{-u(x)}$$

then, on input x and with access to the prover, K computes a value w such that $R(x,w)$ holds, within an expected number of steps bounded by $\frac{(|x|+u(x))^c}{p(x)-2^{-u(x)}}$.

We say that V is a verifier with respect to language L if V is a verifier with respect to relation $R(x,w)$ where R is true iff w is a witness to the statement $x \in L$.

2.5.1 Concurrent Composition

We now turn to the question of concurrent composition of proofs of knowledge. Concurrent execution of several protocols is an execution where the messages from different instances of the protocol are arbitrarily, even adversarially, interleaved between each other. In other words, we assume that the adversary controls the scheduling of all the messages, subject to the constraint that within each protocol, all messages are scheduled in the right order.

In the cryptographic context, concurrent composition was first considered by Dwork, Naor, and Sahai [DNS98] in their work on concurrent zero-knowledge proofs. Unlike what we show here about proofs of knowledge under concurrent composition, zero-knowledge proofs do not necessarily remain zero-knowledge under concurrent composition. In fact, no constant-round black-box zero-knowledge proof is zero-knowledge under concurrent composition [CKPR01].

Let us now explain what it means to concurrently compose proofs of knowledge. The behavior of the composed verifier is as expected: every time a new prover contacts the verifier and wants to run a proof of knowledge protocol, a new copy of the verifier is created, each running separately on its corresponding input. More formally:

We will assume that the instructions to initiate the protocols come from the "outside," i.e., from some higher-level process that may be acting adversarially. The composed verifier V^* acts as follows:

1. V^* receives as input the security parameter 1^k that determines the size of instances x.

2. Upon receiving an "Initiate x, *ID*" message from the "outside," it creates a copy of the verifier V on input $(1^k, x)$. If this is the i'th "Initiate" message, then we denote this copy of the verifier by V_i, and this input by x_i. The identifier *ID* is used as a label to denote that a message received from the prover is concerning the protocol invocation associated with the given identifier *ID*.

3. If V_i computes a message m to be sent to its prover, the composed verifier V^* sends the message "*ID*,m" to the prover, where *ID* is the identifier corresponding to this copy of the verifier.

4. Upon receiving a message "*ID*,m" from the prover, the composed verifier forwards it to the verifier V_i that corresponds to the identifier *ID*.

5. Upon receiving the "Halt" message from the outside, the composed verifier halts and outputs the vector X of all the inputs received, and an index set I that indicates for which inputs the proofs of knowledge were accepted.

Note that the composed verifier accepts some input values and rejects others. The property we want is that, instead of running the composed verifier, we should have been able to substitute him with a knowledge extractor such that the prover would not notice the difference, and yet the knowledge extractor would output all the witnesses for which the proof of knowledge went through successfully. More formally:

Definition 2.5.2 (Concurrent POK). *Let V be a knowledge verifier for relation R, and let V^* be the concurrent version of V. We say that V can be concurrently composed if there exists an extractor K^* that has access to the prover, and runs in expected polynomial time such that*

> Extraction*: $K^*(1^k)$ runs with (black-box) access to the adversary $\mathcal{A}$ and outputs (1) a value $v = (X, I)$, where X is a vector of instances, $X = (x_1, \dots, x_\ell)$, and $I \subseteq [\ell]$ is a set of indices; and (2) for each $i \in I$, a witness w_i such that $R(x_i, w_i)$ holds.*
>
> Simulation*: The adversary's view is the same whether talking to the verifier V or to the knowledge extractor K. More formally, for all adversaries $\mathcal{A}$,*
>
> $$[\mathit{VIEW}_{\mathcal{A}}, \mathit{OUT}_V](\mathcal{A} \leftrightarrow V^*(1^k)) \overset{c}{\equiv} [\mathit{VIEW}_P, v_{K^*}](\mathcal{A} \overset{\blacksquare}{\leftarrow} K^*(1^k))$$
>
> *where v_{K^*} denotes the v-part of the output of the extractor K^*.*

Theorem 2.5.1 (Concurrent Composition of POK). *If V is a knowledge verifier with respect to R, then V can be concurrently composed with respect to R.*

Proof. Let us first describe the knowledge extractor algorithm K^*. We will then analyze its correctness and running time.

Let a *combined* extractor K' be a machine that, on input x and with black-box access to a prover P, does two things in parallel: it runs the extractor K, and it also searches through all the $u(x)$-bit strings for a witness w such that $R(x, w) = 1$. If K outputs such a w, K' halts and outputs w, otherwise, K' continues its search. If K' discovers that no string w of length $u(x)$ is a witness to $R(x, w)$, then K' rejects.

K^* acts as follows:

1. Run the composed verifier V^*, observe its output, and save its computation history.

2. For each $x_i \in I$, extract a witness w_i, as follows:

   (a) Make a new copy of the computation, and rewind this copy to the moment when the message "Initiate x_i" was first issued.

   (b) Replace the verifier V_i with the combined extractor $K'(x_i)$ described above. Run K' to either obtain w_i or discover that no w_i exists, as follows:

      - When K' sends a message to the adversary, this message is forwarded to the adversary with the right *ID*.

- When the adversary sends a message to V_i, this message is forwarded to K'.
- When K' needs to rewind the adversary, the adversary is rewound as directed by K'.

(c) In case K' rejects, K^* will reject everything (output an empty index set) and halt. Otherwise, it will store w_i and proceed to extract the next witness.

3. K^* outputs the same values X and I output by the verifier V^*, and all the witnesses extracted.

It is clear that K^* always terminates. It is also clear that whenever K^* outputs an index set, it also outputs the corresponding witnesses. We must now show that the following two claims hold:

Claim 1 (Running time) K^* runs in expected polynomial time.

Claim 2 (Correctness) The values X and I output by K^* are distributed correctly.

From these two claims, it follows that V is concurrently composable, by Definition 2.5.2.

Let us first show Claim 1. Suppose that the adversary's random coins are fixed. The first stage of K^* is just running several independent copies of the verifier V. It takes a polynomial number of steps. Suppose that the output of V^* is (X, I). By $\mathcal{A}'_i$, let us denote an adversarial algorithm defined as follows: Run $V^*(1^k)$ with adversary $\mathcal{A}$ until you receive the i'th "Initiate x_i" message. Upon receiving this message, instead of making a new copy V_i of the verifier, start interacting with $V(x_i)$, as directed by adversary $\mathcal{A}$. Let $p_i(\mathcal{A}) = \Pr[(\mathcal{A}'_i \leftrightarrow V(x_i)) \rightarrow b : b = 1]$. Note that because V is a knowledge verifier for relation R, it follows that $K(x_i)$ will extract w_i with in $\frac{(x+u(x))^c}{p(x)-2^{-u(x)}}$ steps, if $p_i(\mathcal{A}) > 2^{-u(x)}$, by Definition 2.5.1.

Therefore, if $p_i(\mathcal{A}) > 2^{-u(x_i)}$, then $K'(x_i)$ will run for at most $\frac{(x+u(x))^c}{p(x)-2^{-u(x)}}$ steps in expectation. Otherwise, $K'(x_i)$ will run for at most $O(2^{u(x)})$ steps since that's how long it takes to find w_i by brute force or discover that it does not exist.

Moreover, once $\mathcal{A}$ is fixed, all the values $p_i(\mathcal{A})$ are fixed as well. Even though the event that V^* accepts x_i is not necessarily independent of the event that it accepts x_j, our calculation of expected running time will go through because of linearity of expectation.

For each $i \leq \ell$, the extractor K^* will do no work with probability $1 - p_i(\mathcal{A})$, corresponding to the event that V^* rejected x_i; and with the remaining probability $p_i(\mathcal{A})$, it will run for at most $\frac{(|x_i|+u(x_i))^c}{p_i(\mathcal{A})-2^{-u(x_i)}} \leq \frac{2(|x_i|+u(x_i))}{p_i(\mathcal{A})}$ steps if $p_i(\mathcal{A}) > 2^{-u(x)+1}$, and at for at most $2^{u(k)}$ steps otherwise. Therefore, in expectation, the extractor K^* will spend at most $O((|x_i| + u(x_i))^c)$ steps extracting witness w_i. By linearity of expectation, the running time of K^* is $O(\ell(|x| + u(x))^c)$, where $x \in X$ is the value for which $|x| + u(x)$ is maximized. This quantity is polynomial in k since all x_i's are of length that is polynomial in k. Thus we have proved Claim 1.

Now let us prove Claim 2. The only case when K^* does not output the value obtained by running V^* is when V^* accepted some x_i for which no witness exists.

Suppose this happens with non-negligible probability. Then we show that there is no knowledge extractor for verifier V. Namely, let us construct an adversarial prover P who will induce V to accept with probability greater than $2^{-u(x)}$ a string x for which no witness exists. This in itself will imply that no extractor exists, since, no matter what K is or how long an extractor will run, it will fail to find a witness, contradicting the given. The adversarial prover will be $\mathcal{A}'_i$ described above. Since it fully emulates the conditions of the interaction between V^* and $\mathcal{A}$, it induces V to accept x_i with the same probability as the probability that V^* accepts x_i. □

Remark. Note that both the definition of concurrent composition, and the theorem can be formulated for an *on-line* composed verifier, i.e., the verifier that does not wait until the end of the computation to output x_i together with its acceptance/rejection decision. The modification to the definition is that the simulator should also produce its output on-line, without waiting for the end of the computation, and its output should be indistinguishable from that of the on-line composed verifier.

2.6 Protocols for Models with Public Parameters

In the public parameter model, we assume that some set-up phase (possibly run by a trusted third party, or possibly by a secure distributed protocol among all parties), was run before the protocol begins. We treat the adversary as fixed *before* this set-up phase took place. An example of the public-parameter model is the public-key model, where we assume that in the initial set-up phase each participant has obtained a key pair: a public key that is published and available to all parties, and a secret key that is only known to this party.

The reason we think of the non-uniform adversary as being fixed before the set-up phase, is that, in the first place, this is a more realistic scenario since an adversary will attack an algorithm as opposed to its specific incarnation. In addition, if the adversary is allowed to be fixed *after* the set-up phase, then this set-up phase becomes obsolete since the adversary may be designed especially for this specific setting of the parameters, and for example in the public-key setting, such an adversary will know all the secret keys.

Definitions of zero-knowledge proofs and proofs of knowledge in these models are changed as follows: The adversary is fixed before the set-up phase, the players take public parameters as input, and the simulator and knowledge extractor are allowed auxiliary inputs that depend on the public parameters, and are computable from the randomness that was used to generate these public parameters. Here, we omit the formalism, but refer the reader to Reyzin's work [Rey01] on secure protocols in various such models.

2.6.1 Concurrent Zero-knowledge Using Trapdoor Commitments

Damgård [Dam00] showed that, assuming trapdoor commitments and the public-parameter setting, it is easy to achieve concurrent zero-knowledge protocols using Σ-protocols.

First, let us define what it means for a zero-knowledge proof to compose concurrently. It is similar to concurrently composable proof of knowledge, except that here we are concerned about both the composed Prover P^* and the composed Verifier V^*. They are both similar to what we described in the previous section for the proof of knowledge case. The composed verifier V^* is, in fact, identical. For the presentation of these composed protocols, we will assume that the instructions to initiate a protocol come from an adversary. The composed prover P^*.

1. Upon receiving an input "Initiate x, w, *ID*" from the outside, create a copy of the Prover P and run $P(x, w)$. If this is the i'th "Initiate" message, then we denote this copy of the prover by P_i, and this input by (x_i, w_i). The identifier *ID* is used as a label to denote that a message received from the prover is concerning the protocol invocation associated with the given identifier *ID*.

2. If P_i computes a message m to be sent to its verifier, the composed prover sends (m, ID) to the composed verifier.

3. If the composed verifier sends the message (m, ID) to the composed prover, direct that message to the prover P_i corresponding to the right *ID*.

4. Upon receiving the "Halt" message from the outside, halt.

Definition 2.6.1. *A proof system (P, V) is concurrently composable if*

Completeness*: The composed verifier V^* will always accept everything that the composed prover P^* proves.*

Soundness s*: The probability that the composed verifier V^* accepts a false statement is $\sum s(|x_i|)$.*

Theorem 2.6.1. *Any proof system with negligible soundness is concurrently composable with negligible soundness.*

Proof. (Sketch.) Completeness follows from the perfect completeness of the original proof system, because all that the composed prover and verifier do is create copies of P and V and forward messages between them.

Soundness is shown by contrapositive. Suppose that the soundness of the concurrent proof system is ϵ. Then we show that the soundness of the underlying proof system (P, V) is at least ϵ. This is because we can (non-uniformly) see which false x_i the Verifier accepts with this probability, and then create a prover that would emulate the concurrent conditions and force the verifier to behave in the same was as in the concurrent conditions. □

Definition 2.6.2 (Concurrent ZK). *A proof system (resp., argument) (P, V) for language L is concurrently zero knowledge if for all probabilistic polynomial-time adversaries $\mathcal{A}$, for all ℓ, there is a simulator S for $\mathcal{A}$'s interaction with the composed prover P^* on input $x_1, \dots, x_\ell \in L$ (resp., where the composed prover P^* also receives as auxiliary input the witnesses $w_1, \dots, w_\ell$).*

Definition 2.6.3 (Black-box concurrent ZK). *A proof system (resp., argument) (P, V) for language L is concurrently zero knowledge if there is a simulator S for all probabilistic polynomial-time adversaries $\mathcal{A}$, for all ℓ, for $\mathcal{A}$'s interaction with the composed prover P^* on input $x_1, \dots, x_\ell \in L$ (resp., where the composed prover P^* also receives as auxiliary input the witnesses $w_1, \dots, w_\ell$).*

It has been shown that if these two notions are non-trivial (i.e., if not all languages for which zero-knowledge proofs exist are in BPP), then these two notions are distinct. This follows from the fact that, Canetti et al. [CKPR01] showed that at least a logarithmic number of rounds is required to achieve black-box concurrent zero knowledge, while Barak [Bar01] showed that non-black-box concurrent ZK can be achieved in constant rounds.

In the common random string (CRS) model, *non-interactive* zero knowledge proofs are possible for any language in NP [BDMP91, FLS99] (assuming certified trapdoor permutations; this assumption was later reduced to trapdoor permutations [BY96]). Non-interactive zero-knowledge proofs are concurrently composable simply because they are sequentially composable. However, general non-interactive zero knowledge proofs can be quite inefficient, since the best know techniques to carry one out require going through the Cook-Levin theorem. Therefore, interactive zero-knowledge protocols are also of interest in the CRS and common setup models, so long as they are more efficient.

Definition 2.6.4 (Black-box concurrent ZK in the common setup model). *A proof system (P, V) is black-box concurrently composable in the common setup model if there exist simulators S_1, S_2 such that*

- *S_1 is the simulator for the generation of the common setup. It takes as input the security parameter 1^k, and outputs the setup string s, and some auxiliary information a. Let $S_1(1^k)_s$ denote the s-part of the output of S_1.*
- *For all adversaries $\mathcal{A}$ that act as both a verifier and as the "outside" that supplies the instances to the proof system, $(s \leftarrow \mathcal{G}(1^k), VIEW_{\mathcal{A}}(\mathcal{A}(r) \leftrightarrow P^*)) \overset{c}{\equiv} (S(1^k)_s, VIEW_{\mathcal{A}}(\mathcal{A}(r) \overset{\blacksquare}{\leftarrow} S_2(a)))$, where $\mathcal{G}(1^k)$ denotes the algorithm that, on input 1^k, outputs the setup parameters for the proof system (P, V).*

Damgård showed how to transform a special flavor of a zero-knowledge proof of knowledge, namely, the Σ-protocol into a black-box concurrently composable zero-knowledge argument of knowledge in this model using trapdoor commitments. Here, we recall his result. In the sequel, we will give Σ-protocols for the languages we will need a concurrent zero-knowledge proofs for.

A Σ-protocol is a three-round proof of knowledge protocol where in the second round, all that the verifier needs to do is reveal its random coins; its security is only against a verifier who chooses his coins adversarially. Although examples of such protocols were known prior to his work, the term Σ-protocol was introduced by Cramer [Cra97]. The reason that he called these protocols Σ-protocol is the shape of the letter Σ. The definition we give here is slightly less restrictive than the one given by Cramer and by Damgård [Dam02] in that we do not require knowledge extraction to be as efficient as they do.

Definition 2.6.5 (Σ-Protocol). *Let (P, V) be a three-round proof system for language L, where V is a knowledge verifier with respect to language L. Suppose (P, V) starts with a message from P to V, and where V's only message to P consists of its random coins. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary that on input x, chooses the random coins for verifier V. By $V_{\mathcal{A}}$, let us denote the resulting verifier. (P, V) is a Σ-protocol if there exists a simulator S such that for all $\mathcal{A}$, and for all $x \in L$, S simulates the view of $V_{\mathcal{A}}$ for the interaction with prover P on input x, where the Prover receives the auxiliary input w that is the witness for $x \in L$.*

Damgård technique is as follows: the common setup produces a trapdoor commitment key PK. The Σ-protocol is then modified as follows:

1. Let m_1 be the prover P's first message. Instead of sending m_1, the prover will send $M = \texttt{Commit}(PK, m_1, r)$ for random r.

2. The verifier sends R that is the contents of its random tape, as is prescribed by the Σ-protocol.

3. The prover sends r, thereby revealing his first message m_1. He also computes and sends the prover P's response to the message R from the verifier.

4. The verifier checks that $M = \texttt{Commit}(PK, m_1, r)$ and uses verifier V on randomness R to verify prover's messages (m_1, m_2).

Let us called the proof system thus modified (P', V').

Lemma 2.6.2. *If (P, V) is a proof of knowledge for language L, then so is (P', V').*

Proof. (Sketch) The completeness property is clear. The knowledge extraction property holds because suppose that the soundness of the proof system got worse by ϵ. Let us show that this contradicts either the knowledge extraction property of the proof system (P, V), or the binding property of the commitment. Let us try to first contradict the knowledge extraction property. We will interact with the adversary as follows: In the first step, the adversary sends us the message M. In Step 2, we will send him randomness, and obtain the response. We will repeat this step until the prover induces us to accept (each time this will happen with probability $s + \epsilon$, so this step must be repeated $1/\epsilon$ times.) So we have created an accepting transcript with initial message M, with the verifier's randomness, let us call it R_1. Now we rewind him to Step 2 and obtain another accepting transcript with message M, this time with different randomness R_2. Suppose that the probability that in this sort of experiment, both times, the same value m_1 corresponding to M is revealed, is $1/2$. Since R_2 was chosen independently of R_1, and it took $1/\epsilon$ tries in expectation to find it, and the probability that it was found is $1/2$, it follows that for $\epsilon/2$ fraction of possible R's, the prover would send a message in Step 3 inducing the verifier to accept. Then the knowledge error of the original proof system is at least $\epsilon/2$: this is because if the initial message of Prover P is m_1, then it turns out that for $\epsilon/2$-fraction of the possible R's, there is a message m_2 that induces V to accept. So suppose that with probability more that $1/2$, the adversary gives a different m_1 as corresponding to commitment M. Then we have broken the commitment scheme. □

Lemma 2.6.3. (P', V') *is concurrent black-box zero-knowledge proof of knowledge in the auxiliary string model.*

Proof. (Sketch) As for the proof of knowledge part, it is taken care of by theorem 2.5.1.

Now let us show the zero-knowledge property. The simulator will act as follows: In the common setup phase, it will run $\mathcal{G}'$ of the trapdoor commitment scheme and obtain not only a commitment public key PK, but also the trapdoor TK.

Then it will run the following code for each interaction with verifier V^*:

1. Compute an arbitrary m_1' and random r, and let $M = \texttt{Commit}(PK, m_1', r)$.
2. Receive the value R.
3. Run S, the simulator of the Σ-protocol (P, V), to obtain a transcript (m_1, m_2) for randomness R. Alter M to be a commitment to m_1 and send this to the verifier.

Note that this simulation can be carried out concurrently because it does not need any rewinding of the verifier.

Due to the trapdoor property of the commitment scheme, the resulting simulation will be indistinguishable from the interaction with the prover. □

Chapter 3

Anonymous Credential System

As information becomes increasingly accessible, protecting the privacy of individuals becomes more challenging. To solve this problem, an application that allows the individual to control the dissemination of personal information is needed. An anonymous credential system (also called pseudonym system), introduced by Chaum [Cha85], is the best known idea for such a system.

An anonymous credential system [Cha85, CE87, Che95, Dam90, LRSW99] consists of users and organizations. Organizations know the users only by pseudonyms. Different pseudonyms of the same user cannot be linked even if all of the organizations cooperate. Yet, an organization can issue a credential to a pseudonym, and the corresponding user can prove possession of this credential to another organization (who knows her by a different pseudonym), without revealing anything more than the fact that she owns such a credential. Credentials can be for unlimited use (these are called *multiple-show* credentials) and for one-time use (these are called *one-show* credentials). Possession of a multi-show credential can be demonstrated an arbitrary number of times; these demonstrations cannot be linked to each other.

Basic desirable properties. It should be impossible to forge a credential for a user, even if users and other organizations team up and launch an adaptive attack on the organization. Each pseudonym and credential must belong to some well-defined user [LRSW99]. In particular, it should not be possible for different users to team up and show some of their credentials to an organization and obtain a credential for one of them that this user alone would not have gotten. Systems where this is not possible are said to have *consistency of credentials.* As organizations are autonomous entities, it is desirable that they be separable, i.e., be able to choose their keys themselves and independently of other entities, so as to ensure security of these keys and facilitate the system's key management.

The scheme should also provide user privacy. An organization cannot find out anything about a user, apart from the fact of the user's ownership of some set of credentials, even if it cooperates with other organizations. In particular, two pseudonyms belonging to the same user cannot be linked [Bra99, Cha85, CE87, Che95, Dam90, LRSW99].

Finally, it is desirable that the system be efficient. Besides requiring that it be based on efficient protocols, we also require that each interaction involve as few entities as possible, and the rounds and amount of communication be minimal. In particular, if a user has a multiple-show credential from some organization, she ought to be able to demonstrate it without getting the organization to reissue credentials each time.

Additional desirable properties. It is an important additional requirement that the users should be discouraged from sharing their pseudonyms and credentials with other users. One way of doing it is by ensuring that sharing a credential implies also sharing a particular, valuable secret key from *outside* the system (e.g., the secret key that gives access to the user's bank account) [DLN96, GPR98, LRSW99].

In addition, it may be desirable to have a mechanism for discovering the identity of a user whose transactions are illegal (this feature, called *global anonymity revocation*, is optional); or reveal a user's pseudonym with an issuing organization in case the user misuses her credential (this feature, called *local anonymity revocation*, is also optional). It can also be beneficial to allow *one-show* credentials, i.e., credentials that should only be usable once and should incorporate an *off-line* double-spending test. It should be possible to encode attributes, such as expiration dates, into a credential.

Related work. The scenario with multiple users who, while remaining anonymous to the organizations, manage to transfer credentials from one organization to another, was first introduced by Chaum [Cha85]. Subsequently, Chaum and Evertse [CE87] proposed a solution that is based on the existence of a semi-trusted third party who is involved in all transactions. However, the involvement of a semi-trusted third party is undesirable.

The scheme later proposed by Damgård [Dam90] employs general complexity-theoretic primitives (one-way functions and zero-knowledge proofs) and its complexity renders it not applicable for practical use. Moreover, it does not protect organizations against colluding users. The scheme proposed by Chen [Che95] is based on discrete-logarithm-based blind signatures. It is efficient but does not address the problem of colluding users. Another drawback of her scheme and the other practical schemes previously proposed is that to use a credential several times, a user needs to obtain several signatures from the issuing organization.

In an earlier work [LRSW99, Lys99], we proposed a general credential system. While this general solution captured many of the desirable properties, it is not usable in practice because the constructions are based on one-way functions and general zero-knowledge proofs of unrealistic complexity. The other, more practical construction we gave, based on a non-standard discrete-logarithm-based assumption, has the same problem as the one due to Chen [Che95]: a user needs to obtain several signatures from the issuing organization in order to use unlinkably a credential several times.

Other related work is that of Brands [Bra99] who provides a certificate system in which a user has control over what is known about the attributes of a pseudonym. Although a credential system with one-show credentials can be inferred from his framework, obtaining a credential system with multi-show credentials is not immediate and may in fact be impossible in practice. Another inconvenience of these and the other discrete-logarithm-based schemes mentioned above is that all the users and the certification authorities in these schemes need to share the same discrete logarithm group.

The concept of revocable anonymity is found in literature on electronic payment systems [BGK95, SPC95] and group signature and identity escrow [ACJT00, CS97, CvH91, KP98] schemes.

Prior to our work, the problem of constructing a practical system with multiple-use credentials eluded researchers for some time [Bra99, Che95, Dam90, LRSW99]. We solve it by extending ideas found in the constructions of strong-RSA-based signature schemes [CS99, GHR99] and group signature schemes [ACJT00].

Our contribution. In Section 3.1 we present our definitions for a credential system with the basic properties. Although not conceptually new and inspired by the literature on multi-party computation [Can95, Can00] and reactive systems [PW00], these definitions are of interest, as our treatment is more formal than the one usually encountered in the literature on credential and electronic cash systems.

Our basic credential system, presented in Section 3.2.2, provably satisfies the basic properties listed above given a signature scheme and a commitment scheme. Its efficiency relies on the efficiency of the underlying schemes.

Our extended credential system, presented in Section 3.3, describes how to incorporate additional desirable properties into the basic credential system.

3.1 Formal Definitions and Requirements

We will define security for a credential system in two steps. First, we will define what it means for a cryptographic system (CS) to conform to an ideal-world specification (IS). Then we will give an ideal-world specification for a credential system.

3.1.1 Definition of a Secure System Conforming to an Ideal-World Functionality

Our definition is inspired by Canetti's universally composable framework [Can01]. However, the notion of security we require is weaker because we are not concerned with a notion of composability of the same strength in this thesis.

Ideal-World Functionality

An ideal (static) functionality consists of a trusted party T through which all transactions are carried out, a set of honest ideal players IS, the adversary $\mathcal{A}$, and the environment E. A transaction is an event involving two or more parties; it can take several rounds of communication to complete; in the sequel we will give specifications for several transactions relevant for credential systems.

All parties receive as input the security parameter 1^k.

The environment E proceeds in periods. In each period, it tells one honest player which transaction to initiate. An honest party $P \in \text{IS}$, activated by the environment, participates in the specified transaction by contacting the trusted party T and telling T what transaction it intends to carry out and with whom. T then executes the trusted code for this transaction, which may involve sending messages to other players and receiving messages from them. In the end of the period, P tells E the outcome of the transaction.

Adversary $\mathcal{A}$ can also send and receive messages to and from T. The environment E has total control over all of the adversary's actions: the adversary $\mathcal{A}$ forwards all the received messages to E, and then E tells the adversary $\mathcal{A}$ which messages to send.

In Figure 3-1, we give an example of an ideal system.

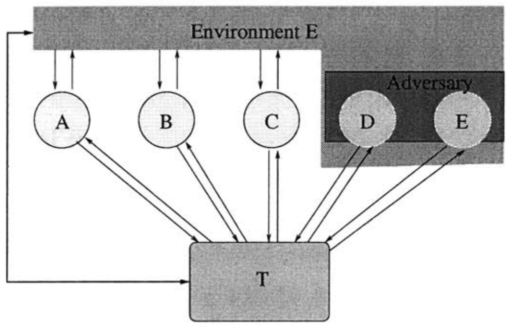

Figure 3-1: An example of an ideal system where the environment interacts with the ideal parties A, B, and C, and the adversary fully controls ideal parties D and E.

Cryptographic System

A (static) cryptographic system is modeled by a set of honest cryptographic players CS, the adversary $\mathcal{A}$, and the environment E.

Here, too, the environment E proceeds in periods. In the beginning of each period, it tells some honest player which transaction to initiate. An honest party $P \in \text{CS}$, activated by the environment, participates in the specified transaction by contacting the parties it needs to contact to carry out the given transaction, and running the corresponding cryptographic protocol with these parties. In the end of the period, P tells E the outcome of the transaction. (Note that here, these are periods of execution of the system, not rounds of communication; each protocol may require very many communication rounds; once it is completed, this is the end of this period of execution).

Adversary $\mathcal{A}$ can also send and receive messages to and from any other player in the system. The environment E has total control over all of the adversary's actions: the adversary $\mathcal{A}$ forwards all the received messages to E, and then E tells the adversary $\mathcal{A}$ which messages to send and to whom.

Let `Init` denote the algorithm for initializing the system's public parameters. A designated (honest, incorruptible) party is responsible for making these parameters known to all the parties. In case no public parameters are needed, assume that `Init` is a void algorithm.

Figure 3-2, is an example of a real system.

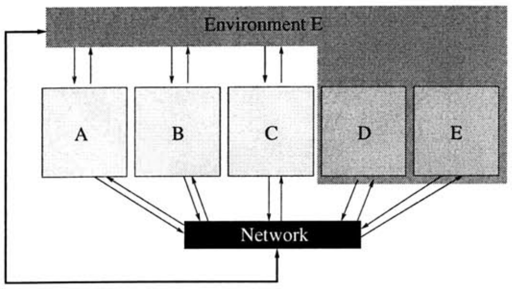

Figure 3-2: An example of a real system where the environment interacts with the parties A, B, and C, fully controls parties D and E (and so parties D and E are the adversary), and has some knowledge about the network's behavior.

Cryptographic System Satisfying an Ideal Functionality

The idea of this definition is that no environment should be able to tell whether it is talking to a cryptographic system, or to the ideal functionality. This idea originates in the multi-party computation literature [GMW86].

Definition 3.1.1. *A cryptographic system CS with initialization algorithm* `Init` *conforms to the ideal specification IS in a black-box fashion, if there exists simulator S such that for all pairs probabilistic polynomial-time families of interactive Turning machines* $\{E_k, \mathcal{A}_k\}$ *there exists a negligible function* $\nu(k)$ *such that,*

$$\begin{tikzcd}[column sep=small] \left|\Pr[PK \leftarrow \texttt{Init}(1^k);\ (\right. & CS\,(1^k, PK) \arrow[r, leftrightarrow] \arrow[rr, leftrightarrow, bend left] & \mathcal{A}_k \arrow[r, leftrightarrow] & E_k) \arrow[r] & b\ :\ b=1] \ - \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small] \Pr[\ (& IS\,(1^k) \arrow[r, leftrightarrow] \arrow[rrr, leftrightarrow, bend left] & T(1^k) \arrow[r, leftrightarrow] & S(\mathcal{A}_k) \arrow[r, "\blacksquare"] & E_k) \arrow[r] & b\ :\ b=1]| \ = \ \nu(k) \end{tikzcd}$$

Note that in the definition above, the simulator is given *black-box* access to the environment. Of course, since the simulator does not have black-box access to the ideal parties, it cannot rewind the environment beyond the beginning of a given period, and cannot read messages sent between the environment and the honest parties.

The reason that this definition is weaker than the universally composable definition, is precisely that the environment has to schedule an entire transaction to be carried out in a given period, so it cannot interleave messages of different transactions or have transactions conducted concurrently. This makes this notion of security much less appealing than universal composability, but also easier to work with.

Figure 3-3 shows how to show that the real system from Figure 3-2 satisfies the specification of the ideal system from Figure 3-1.

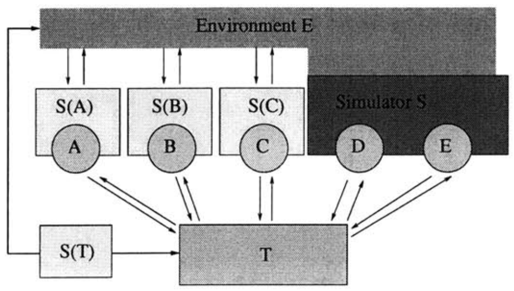

Figure 3-3: A simulator translating a real environment and adversary into an ideal environment and adversary. $S(A)$, $S(B)$ and $S(C)$ are simulators that translate the real environment E's instructions for ideal parties A, B, and C. The simulator $S(T)$ translates the messages of the real environment E into messages that the trusted party T understands, and back. The main simulator S translates the real environment E and the real adversary into ideal behavior.

3.1.2 Ideal Basic Credential System Functionality

A basic credential system has *users*, *organizations*, and *verifiers* as types of players. Users are entities that receive credentials. The set of users in the system may grow over time. Organizations are entities that grant and verify the credentials of the users. Each organization grants a unique (for simplicity of exposition) type of credential. Finally, verifiers are entities that verify credentials of the users.

The system supports the following transactions:

FormNym(U, O): This protocol is a transaction between a user U and an organization O. The user U contacts T with a request to establish a pseudonym between herself and organization O. She further specifies the login name L_U by which T knows her and the corresponding authenticating key K_U. If she does not have an account with T yet, she first establishes it by providing to T a login name L_U and obtaining K_U in return. She further specifies pseudonym $N_{(U,O)}$. Then T verifies the validity of (L_U, K_U) and, if K_U is the authenticating key corresponding to login name L_U, contacts O and tells it that some user wants to establish a pseudonym $N_{(U,O)}$. The organization either accepts or rejects. If it accepts, it generates a random k-bit tag t and stores it together with N. T notifies U of acceptance or rejection.

GrantCred(U, N, O): This protocol is a transaction between a user U and an organization O. U approaches T, and submits her login name L_U, her authenticating key K_U, the pseudonym N, and the name of organization O. If K_U is not a valid authenticating key for L_U, or if N is not U's pseudonym with O, then T replies with a "Fail" message. Otherwise, T contacts O. If O accepts, then T notifies the user that a credential has been granted, otherwise it replies with "Reject."

VerifyCred(U, V, N, O): This protocol is a transaction between a user U and a verifier V. A user approaches T and gives it her login L_U, her authenticating key K_U, a name of the verifier V, a pseudonym N and a name of a credential-granting organization O. If K_U is a valid authenticating key for L_U, and N is U's pseudonym with organization O, and a credential has been granted by O to N, then T notifies V that the user has a credential from O. Otherwise, T replies with a "Fail" message.

$\mathsf{VerifyCredOnNym}(U, V, N_V, N_O, O)$: This protocol is a transaction between a user U and an verifier V. U approaches T and gives it her login name L_U, her authenticating key K_U, a name of the verifier V, pseudonyms N_V and N_O, and a name of a credential-granting organization O. If K_U is a valid authenticating key for L_U, and N_V is U's pseudonym with V while N_O is U's pseudonym with organization O, and a credential has been granted by O to N_O, then T notifies V that the user with pseudonym N_V has a credential from O.

Output: Once a transaction is completed, a user outputs the name of the transaction just completed, and the name of the organization it was completed with, and if it was a verification transaction, then also the name of the credential-granting organization; an organization outputs just the name of the transaction and the identifying tag t of the pseudonym with which the transaction was conducted.

This ideal system captures the intuitive requirements, such as unforgeability of credentials, anonymity of users, unlinkability of credential showings, and consistency of credentials. Ideal operations that allow additional desirable features can be implemented as well.

Let us briefly illustrate the use of the credential system by a typical example. Consider a user U who wants to get a credential from organization O. Organization O requires the possession of credentials from organizations O_1 and O_2 as a prerequisite to get a credential from O. Assume that U possesses such credentials. Then U can get a credential from O as follows: she first establishes a pseudonym with O by executing $\mathsf{FormNym}(U, O)$ and then shows O her credentials from O_1 and O_2 by executing $\mathsf{VerifyCredOnNym}(U, O, N_O, N_{O_1}, O_1)$ and $\mathsf{VerifyCredOnNym}(U, O, N_O, N_{O_2}, O_2)$. Now O knows that the user it knows under N_O possesses credentials from O_1 and O_2 and will grant U a credential, i.e., U can execute $\mathsf{GrantCred}(U, N_O, O)$. We remark that the operation $\mathsf{VerifyCred}(U, V, N, O)$ exists for efficiency reasons. This operation can be used by U if she wants to show a party only a single credential, e.g., to access a subscription-based service.

3.2 Constructing a Basic Scheme using Signatures and Commitments

Suppose we are given a signature scheme $(G, \texttt{Sign}, \texttt{Verify})$ and a commitment scheme $(G_C, \texttt{Commit})$. Suppose a protocol for proving knowledge of a committed value is given. Let $(P((C, PK_C, PK_\sigma), (x, r, \sigma)), V(C, PK))$ be the protocol for proving knowledge of the values x, r, σ such that $C = \texttt{Commit}_{PK_C}(x, r)$ and $\texttt{Verify}_{PK_\sigma}(x, \sigma)$. We will also need a protocol for signing a committed value, defined below.

3.2.1 Signing a Committed Value

In this section, we will define a protocol needed as a building block for constructing an anonymous credential system. This will be a protocol between a user and a signer. As a result of the protocol, the user will obtain a signature on a value of her choice, while the signer will not learn any information about the value he has signed.

Let $(G, \texttt{Sign}, \texttt{Verify})$ be a secure signature scheme and $(G_C, \texttt{Commit})$ be a non-interactive commitment scheme. In this section, we show a two-party protocol between a user U and a signer S, with the following functional specification, where X_k and R_k are efficiently samplable domains:

Common Input Signature public key PK, commitment public key PK_C. and commitment C.

User's private input The value $x \in X_k$ that needs to be signed, and value $r \in R_k$ such that $C = \texttt{Commit}(PK_C, x, r)$.

Signer's private input Secret key SK that corresponds to PK.

User's output The value σ such that $\texttt{Verify}_{PK}(x, \sigma) = 1$.

The security specifications of the protocol $(S \leftrightarrow U)$ are:

Security for the user : The protocol must reveal (almost) no information about the user's input x. More formally, there exists a negligible function $\nu(k)$ such that for all $x, y \in X_k$, for all (unbounded) adversaries A, for all $PK \in G(1^k)$,

$$\begin{aligned} &\Pr[r \leftarrow R_k; b \leftarrow A \leftrightarrow U(PK, x, r) \ : \ b = 0] - \\ &\Pr[r \leftarrow R_k; b \leftarrow A \leftrightarrow U(PK, y, r) \ : \ b = 0]| \ \leq \ \nu(k) \end{aligned}$$

Security for the signer : A User cannot output a valid signature σ on any value unless σ was issued by the Signer. Moreover, corresponding to any σ, there is a unique value x that a user can produce, such that this value is *fixed* at the beginning of the protocol. More formally, let S denote the interactive Turing machine for the Signer's protocol. There exists an *extractor* algorithm E such that the User's view in interactions with the extractor is the same as in interactions with the Signer, even though the extractor only has a *single* oracle access to the Signer. More formally, for all probabilistic polynomial-time families of Turing machines $\{U_k\}$, there exists a negligible function $\nu(k)$ such that

$$\begin{aligned} &|\Pr[(PK, SK) \leftarrow G(1^k); S(SK) \leftrightarrow U_k(PK) \rightarrow b \ : \ b = 0] \ - \\ &\Pr[(PK, SK) \leftarrow G(1^k); E^{1\texttt{Sign}_{PK}(\cdot)}(1^k) \overset{\blacksquare}{\rightarrow} U_k(PK) \rightarrow b \ : \ b = 0]| \ = \ \nu(k) \end{aligned}$$

More strongly, let $(S(\cdot))$ denote the sequentially composed signer protocol, i.e., the protocol that repeats the signer's protocol sequentially with the adversary U until the adversary stops it. Let the sequentially composed extractor $(E^{(\cdot)}(\cdot))$ be defined analogously. We require the following: For all probabilistic polynomial-time families of Turing machines $\{U_k\}$, there exists a negligible function $\nu(k)$ such that

$$\begin{aligned} |\Pr[(PK, SK) \leftarrow G(1^k); (S(SK)) \leftrightarrow U_k(PK) \rightarrow b \ : \ b = 0] \ - \\ \Pr[(PK, SK) \leftarrow G(1^k); (E^{1\mathtt{Sign}_{PK}(\cdot)}(1^k)) \overset{\blacksquare}{\rightarrow} U_k(PK) \rightarrow b \ : \ b = 0]| \ &= \ \nu(k) \end{aligned}$$

It is clear that using general two-party computation that utilizes a mutually independent and extractable commitment scheme [LLM+01], this protocol can be implemented for any combination of a signature scheme with a commitment scheme.

Lemma 3.2.1. *No polynomial-time adversary can output a signature on a value for which it did not run a protocol with the signer S.*

Proof. This follows from the security properties of the signature scheme. Since the adversary cannot distinguish whether he is talking to the extractor E or to the actual signer S, he is just as likely to output a forgery when talking to S as when talking to E. But if he outputs a forgery when talking to E, that's a direct forgery on the signature scheme. □

3.2.2 Basic Anonymous Credential System

We propose the following basic anonymous credential system:

Init The public parameters for the system is the security parameter 1^k, and a public key PK_C of a commitment scheme.

User initialization The user U chooses a secret key SK_U.

Org. initialization The organization O chooses a signature key pair using algorithm G and publishes its public key PK_O.

FormNym(U, O) User U forms a commitment to her secret SK_U: $N = \mathtt{Commit}(x_U, r)$, sends it to the organization O and proves that she knows the value committed to. Once that is done, U and O both store N as U's pseudonym with O. O also creates a random identifying tag for the pseudonym, t.

GrantCred(U, N, O) User U and organization O run a protocol for securely signing a committed value, as defined in Section 3.2.1, on common input (PK_O, PK_C, N), and the corresponding private inputs. User stores his output σ.

VerifyCred(U, V, N, O) User U who has previously input his pseudonym N into a GrantCred(N, O) transaction, and obtained σ, forms a commitment

$$C = \texttt{Commit}(PK, x_U, r)$$

to his secret key x_U. User U and verifier V invoke the protocol (P, V) for proving knowledge of a signature on a committed value, on common inputs (PK_O, PK_C, C), and user's private input (x_U, r, σ).

VerifyCredOnNym(U, V, N_V, N_O, O) User U who has previously input his pseudonym N_O into a GrantCred(N_O, O) transaction, and obtained σ, and verifier V, with whom U has established pseudonym $N_V = \texttt{Commit}(PK_C, x_U, r)$, run the protocol (P, V) for proving knowledge of a signature on a committed value, with common inputs (PK_O, PK_C, N_V), and user's private input (x_U, r, σ).

Output The same as in the ideal system: Once a transaction is completed, a user outputs the name of the transaction just completed, and the name of the organization it was completed with, and if it was a verification transaction, then also the name of the credential-granting organization; an organization outputs just the name of the transaction and the identifying tag t of the pseudonym with which the transaction was conducted.

3.2.3 Proof of Security

In order to show that our proposed basic credential system conforms to the ideal functionality specification given in Section 3.1.2, we must exhibit a simulator S. In this section, we describe such a simulator and give a sketch of a proof (it is only a sketch because the proof is tedious and straightforward) for why the simulator is correct.

The simulator represents the adversary-controlled parties to the ideal parties, and the ideal parties to the adversary-controlled ones.

The simulator will do the following:

- Representing adversary-controlled users to the IS and ideal organizations to the adversary-controlled users:
  - The simulator generates public keys for ideal organizations.
  - When an adversary-controlled user initiates a FormNym, the simulator extracts his secret key x. If this secret key has never been seen before, then the simulator creates a new login name L and sends it to the trusted party. The trusted party sets up a password K for this user and contact the ideal organization to set up a pseudonym N. The simulator stores that $(x, L, K, (N, O)$. If x has been seen before, then the simulator uses the already existing login name L and password K, and once the pseudonym N is set up, adds (N, O) to its record on the key x.

- When an adversary-controlled user initiates a GrantCred on pseudonym N, the simulator runs the extractor of the signing on a committed values protocol: it first obtains x, then initiates a GrantCred with O through the trusted party T, and then once T notifies the simulator that the credential has been granted, the simulator computes the signature on x and finishes the code of the extractor.
- When an adversary-controlled user initiates a VerifyCred, the simulator extracts his secret key x and initiates VerifyCred in the ideal system using the values $(L, K, (N, O))$ corresponding to the key x.

- Representing adversary-controlled organizations to the IS and ideal users to the adversary-controlled organizations:
  - When T contacts S for a FormNym, create a commitment N on a random value x and run FormNym with the adversary's organization using N. Respond to the trusted party to establish the ideal-world pseudonym N' for the anonymous user.
  - When T contacts S for a $\mathsf{GrantCred}(N', O)$, where N' is the ideal-world pseudonym that corresponds to pseudonym N, run the $\mathsf{GrantCred}(N, O)$ with the adversarially-controlled organization O. If O issues a value σ such that $\mathtt{Verify}_{PK_O}(x, \sigma)$, then notify T that the credential is granted.
  - When T contacts S for a $\mathsf{VerifyCred}(V, ?, O)$, form a random commitment C and invoke the simulator for the zero-knowledge proof of knowledge of a signature on the value committed to in C. (Note that O may be an adversarially controlled organization, so it may be necessary to fake the proof.)
  - When T contacts S for a $\mathsf{VerifyCred}(V, N'_V, ?, O)$, where N'_V is the ideal-world pseudonym corresponding to the real-world pseudonym N_V, invoke the simulator for the zero-knowledge proof of knowledge of a signature on the value committed to in N_V.

If a polynomial-time E distinguishes between a simulation and CS, then it distinguishes for at least one of these simulated transactions, by a standard hybrid argument: we can run a hybrid system in which the first several transactions are real-world ones, and the transactions following them are using the ideal systems and the simulator.

There are eight cases for the transaction that is distinguishable. In all but two of them (VerifyCred or VerifyCredOnNym between a dishonest user and an honest organization, where the credential issuer is also honest), distinguishing a real transaction from the simulated one amounts to either distinguishing a prover from a simulator in a zero-knowledge proof, or distinguishing a verifier from a knowledge extractor in a proof of knowledge, or distinguishing a signer from an extractor in a protocol for signing a committed value. All seven lead to straightforward contradictions.

Suppose the adversary distinguishes whether VerifyCred is run with a real-world organization or with a simulator. But it does not distinguish an extractor from a real organization. Therefore, it must distinguish because it succeeds in proving knowledge of a signature on a value x that has not been signed. But then we set up a reduction from the signature scheme.

3.2.4 Discussion and Efficiency Analysis

The efficiency of this basic credential system depends on the efficiency of its building blocks. It is clear that this system can be constructed from any one-way function using the following well-known facts: (1) OWF $\Rightarrow$ PRG [HILL99]; (2) PRG $\Rightarrow$ secure commitment [Nao91]; (3) OWF $\Rightarrow$ secure signatures [Rom90]; and (4) NP $\in$ ZKPOK [GMW87b], where ZKPOK denotes the class of languages for which there exist zero-knowledge proofs of knowledge of a witness. However, going about it in this way, it unsuitable for practical use.

In Chapter 4, we give an efficient signature scheme, together with an efficient commitment scheme and efficient protocols for issuing a signature on a committed value and for proving knowledge of a signature on a committed value. By "efficient," we mean that the communication complexity is going to be linear in the security parameter, and the computationally we will require a constant number of k-bit arithmetic operations (such as modular exponentiation) per transaction.

3.3 Building in Additional Desirable Properties

In this section, we explain how to extend the basic credential system presented above to provide additional desirable features. The exposition in this section is high-level, because the low-level details are straightforward.

3.3.1 Credential Revocation

It is very desirable to be able to revoke a credential. In the anonymous setting, doing so is tricky and introduces an extra communication and computation overhead. We do not know how to do that from the general assumptions of commitment and signature schemes. We address this issue by introducing dynamic accumulators in Chapter 5.

3.4 Credential Attributes and Limited-Use Credentials

Suppose we have a signature scheme for signing a block of messages. Then different parts of the signed message can serve different functions. In Chapter 4, Section 4.4, we give a signature scheme of this kind.

For example, if tuples of the form (SK, d) can be signed, where d is the expiration date of a credential, or some other attribute, then in order to demonstrate that a credential has not expired at time t, it is sufficient to prove knowledge of a signature on a tuple (SK, d) where $d > t$.

In order to create single-use credentials, if tuples of the form (SK, N) can be signed, where N is a nonce, then in order to show a single-use credential, the user can reveal the nonce N and prove knowledge of a signature on (SK, N). Then if the user attempts to use the credential a second time, this can be detected.

An even better solution for single-use credentials, is as follows: the user obtains a signature on (SK, N_1, N_2), where $SK, N_1 \in \mathbb{Z}_p$ for some prime number p, and N_2 is just some random identifier for the credential. The verifying organization sends to the user a random value $a \in \mathbb{Z}_p$. The user reveals the values $b = aSK + N_1 \bmod p$ and N_2, and proves knowledge of a signature on $(SK, N_1.N_2)$ such that $b = aSK + N_1 \bmod p$. An organization that accepts the credential, will store the credential record (N_2, a, b) in a public archive. If the user ever runs the credential showing protocol again, then most likely he will receive challenge a' and will have to respond with $b' = a'SK + N_1$, and N_2. Off-line, the organization can locate the credential record by searching for N_2, and solve for the value SK by solving the system of linear equations.

This method for preventing double-spending was originally proposed by Stefan Brands. We refer the reader to his dissertation [Bra99].

3.4.1 Identity-based Properties

For this section, assume that we are within a public-key infrastructure (PKI). That is to say, every (potentially anonymous) user in this system has an identity and a public key corresponding to this identity.

Non-Transferability

Note that in our system, the identity corresponds to knowledge of a secret key. This is the same as in any public-key infrastructure. If the secret keys in the two are the same, then any user in our system will correspond to some actual identity.

In order to ensure such a correspondence, we make the certification authority (CA) for the PKI part of our system. In the PKI, the job of the CA is to ensure that the secret key underlying a given public key is known by the party to whose identity the public key is attached. In the credential system, the job of the CA is to ensure that only users with valid identities participate in the system.

In order to participate in a non-transferable credential system, a user will first contact the CA and reveal his identity and public key PK to the CA. He will then form a pseudonym with the CA, based on his secret key SK corresponding to PK. He will then prove that the pseudonym formed corresponds to this SK. The CA will issue a credential to this user, certifying that he has a valid identity. Let us call this a "root credential." Any organization that wants to talk to users with valid identities, will request that the user prove possession of a root credential.

One may note that, even though now we have a user identity attached to each transaction, we still don't have the property that the users don't share their credentials: all one needs to do in order to allow one's friend to use one's credentials is let the friend get hold of the secret key SK and some other information. This limitation is inherent, since digital data can always be duplicated. However, this can be discouraged by making the value SK too valuable to give away. For example, each user may deposit a big amount of money into the PKI, such that knowledge of the secret key SK allows one to withdraw the money.

Identity-on-Demand Protocols

The mechanism described above discourages credential sharing, but does not prevent it. Moreover, two users who completely trust each other (for example, husband and wife) are not discouraged by such a mechanism at all. Therefore, in scenarios where it is crucial that the identity of a given user be discovered, and the user is physically available and required to prove his identity, it should be possible to enable the user to prove that his digital identity (i.e., his SK) corresponds to his physical identity. This is done by having the user provide his public key PK and show that the SK underlying his pseudonym and credential corresponds to this PK. Using the PKI and external physical notions of identity verification, it is then verified that this user is indeed the owner of this PK.

Anonymity Revocation

The above is only possible when a user can be physically forced to reveal his identity. In some cases, this is not an option. To cope with such situations, we introduce the notion of anonymity revocation. Anonymity revocation works as follows: the user encrypts his public key PK using some trusted third party's encryption. He then proves that the encryption is of the PK that corresponds to the SK underlying the credential the possession of which is being proved. It is easy to see that so long as the PK is a function of the SK, this method allows the trusted party to find out who the user was.

Weaker Forms of Identity-on-Demand and Anonymity Revocation

A weaker form of identity-on-demand and anonymity revocation is when, instead of finding out the identity of a user, it is possible to find out the pseudonym of the user with the organization that granted the credential. For example, if the user misbehaved, the credential granting organization may want to find out and revoke the credential.

To achieve this, suppose every time a credential is issued, the issuing organization must output at least some part of the signature σ on the user's secret key SK. Denote this part $s(\sigma)$. The security for signature scheme must be stronger than usual: not only is is impossible to forge a signature on a message m' not explicitly queried, but it is also hard to compute a signature σ' on already queried message m such that $s(\sigma') \neq s(\sigma)$, where σ is the original signature on message m. The signature scheme we give in Chapter 4 has this property.

When a user proves possession of a credential, he encrypts the value $s(\sigma)$ under some trusted party's key. He then proves that the encrypted value corresponds to his credential. In case it becomes necessary, the trusted party will decrypt this ciphertext and obtain $s(\sigma)$ which will enable the issuing organization to identify the user's pseudonym and take appropriate action.

Chapter 4

Signature Scheme with Efficient Proof of Knowledge

4.1 Number-theoretic preliminaries

We assume that the reader has basic familiarity with modular arithmetic and/or elementary abstract algebra. (For a crash course, read either Dana Angluin's [Ang] or Goldwasser-Bellare's [GB] lecture notes.)

Here we give several relevant facts and definitions.

Definition 4.1.1 (RSA modulus). *A $2k$- or $2k-1$-bit number n is called an* RSA modulus *if $n = pq$, where p and q are k-bit prime numbers.*

Definition 4.1.2 (Euler totient function). *Let n be an integer. The Euler totient function $\phi(n)$ measures the cardinality of the group $\mathbb{Z}_n^*$.*

Fact 4.1.1. *If $n = pq$ is an RSA modulus, then $\phi(n) = (p-1)(q-1)$.*

Notation: We say that an RSA modulus $n = pq$ is chosen uniformly at random with security k if p and q are random k-bit primes. Let `RSAmodulus` denote the algorithm that, on input 1^k, outputs a random RSA modulus with security k. Let $\texttt{RSApk}(1^k)$ denote the algorithm that, on input 1^k, chooses two random k-bit primes, p and q, and outputs $n = pq$ and a random $e \in \mathbb{Z}_n^*$ such that $\gcd(e, \phi(n)) = 1$. Such a pair (n, e) is also called *RSA public key*, where the corresponding *secret key* is the value $d \in \mathbb{Z}_{\phi(n)}$ such that $ed \equiv 1 \bmod \phi(n)$.

Assumption 4.1.1 (RSA Assumption). *The RSA assumption is that given an RSA public key (n, e), and a random element $u \in \mathbb{Z}_n^*$, it is hard to compute the value v such that $v^e \equiv u \bmod n$. More formally, we assume that for all polynomial-time circuit families $\{\mathcal{A}_k\}$, there exists a negligible function $\nu(k)$ such that*

$$\Pr[(n,e) \leftarrow \texttt{RSApk}(1^k); u \leftarrow \mathbb{Z}_n^*; v \leftarrow \mathcal{A}_k(n,e,u) \ : \ v^e \equiv u \bmod n] = \nu(k)$$

Assumption 4.1.2 (Strong RSA Assumption). *The strong RSA assumption is that it is hard, on input an RSA modulus n and an element $u \in \mathbb{Z}_n^*$, to compute values $e > 1$ and v such that $v^e \equiv u \bmod n$. More formally, we assume that for all polynomial-time circuit families $\{\mathcal{A}_k\}$, there exists a negligible function $\nu(k)$ such that*

$$\Pr[n \leftarrow \texttt{RSAmodulus}(1^k); u \leftarrow \mathbb{Z}_n^*; (v,e) \leftarrow \mathcal{A}_k(n,u) \ : e > 1 \ \wedge \ v^e \equiv u \bmod n] = \nu(k)$$

The tuple (n, u) generated as above, is called a *general instance* of the *flexible* RSA problem.

Notation: By $QR_n \subseteq \mathbb{Z}_n^*$ we will denote the set of quadratic residues modulo n, i.e., elements $a \in \mathbb{Z}_n^*$ such that $\exists b \in \mathbb{Z}_n^*$ such that $b^2 \equiv a \bmod n$.

If (n, u) is a general instance of the flexible RSA problem, and $u \in QR_n$, then (n, u) is a *quadratic* instance of the flexible RSA problem.

Note that since one quarter of all the elements of $\mathbb{Z}_n^*$ are quadratic residues, the strong RSA assumption implies that it is hard to solve quadratic instances of the flexible RSA problem. In the sequel, by an *instance* of the flexible RSA problem, we will mean a *quadratic*, rather than general instance.

Corollary 4.1.1. *Under the strong RSA assumption, it is hard, on input a flexible RSA instance (n, u), to compute integers $e > 1$ and v such that $v^e \equiv u \bmod n$.*

Definition 4.1.3 (Safe primes). *A prime number p is called* safe *if $p = 2p' + 1$, such that p' is also a prime number. (The corresponding number p' is known as a* Sophie Germain *prime.)*

Definition 4.1.4 (Special RSA modulus). *An RSA modulus $n = pq$ is* special *if $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.*

Notation: We say that a special RSA modulus $n = pq$ was chosen at random with security k if p and q are random k-bit safe primes.

Theorem 4.1.2 (Chinese Remainder Theorem). *If $n = pq$, and $\gcd(p, q) = 1$, then $\Phi : \mathbb{Z}_n \mapsto \mathbb{Z}_p \times \mathbb{Z}_q$, $\Phi(x) = (x \bmod p, x \bmod q)$ is an efficiently computable and invertible ring isomorphism.*

Proof. Computing $\Phi(x)$ is straightforward. Consider the function $\Omega : \mathbb{Z}_p \times \mathbb{Z}_q \mapsto \mathbb{Z}_n$, $\Omega(x_1, x_2) = x_1 \nu q + x_2 \pi p \bmod n$, where $\nu \equiv q^{-1} \bmod p$, $\pi \equiv p^{-1} \bmod q$. Note that $\Phi(\Omega(x_1, x_2)) = (x_1, x_2)$. Since $|\mathbb{Z}_n| = |\mathbb{Z}_p \times \mathbb{Z}_q|$, it follows that $\Omega = \Phi^{-1}$, and Φ is a bijection.

Additive isomorphism:

$$\begin{aligned} \Omega(x_1 + y_1, x_2 + y_2) &= (x_1 + y_1)\nu q + (x_2 + y_2)\pi p \bmod n \\ &= (x_1 \nu q + x_2 \pi p) \bmod n + (y_1 \nu q + y_2 \pi p) \bmod n \\ &= \Omega(x_1, x_2) + \Omega(y_1, y_2) \end{aligned}$$

Multiplicative isomorphism: we wish to show that $\Phi(xy) = \Phi(x)\Phi(y)$. $\Phi(xy) = (xy \bmod p, xy \bmod q)$. Suppose $\Phi(x) = (x_1, x_2)$, $\Phi(y) = (y_1, y_2)$. Then $xy \bmod p =$

$(x_1+ap)(y_1+bp) \bmod p = x_1y_1 \bmod p$ for some integers a, b. Analogously, $xy \bmod q = x_2y_2 \bmod q$. Therefore, $\Phi(xy) = (x_1y_1, x_2y_2) = (x_1, x_2)(y_1, y_2) = \Phi(x)\Phi(y)$. □

Lemma 4.1.3. *If $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$ is a special RSA modulus, then QR_n is a cyclic group under multiplication, of size $p'q'$, where all but $p' + q'$ of the elements are generators.*

Proof. It is clear that QR_n is a group. It it easy to see (for example, by the Chinese remainder theorem), that QR_n is isomorphic to $QR_p \times QR_q$. QR_p (resp., QR_q has order p' (resp., q'), and all but one of its elements are generators. Since $\gcd(p', q') = 1$, an element that is a generator for QR_p and QR_q, is a generator for QR_n. Therefore, there are $(p' - 1)(q' - 1)$ generators in QR_n. □

Corollary 4.1.4. *If (n, u) is an instance of the flexible RSA problem, and n is a special RSA modulus, we may assume that u is a generator of QR_n.*

The following lemma originates from the analysis of the primality test due to Miller [Mil76] and Rabin [Rab80].

Lemma 4.1.5. *Let a composite integer n be given. Given any value x such that $\phi(n) \mid x$, one can find a non-trivial divisor of n in probabilistic polynomial time.*

Proof. If n is even, we are done. So suppose n is odd. Suppose there exists a value $p > 2$ such that $p^2 \mid n$. Then $p \mid \gcd(\phi(n), n) \mid \gcd(x, n)$, and so $\gcd(x, n)$ is a non-trivial divisor of n.

Otherwise, n has at least two distinct odd prime factors. Call them p and q. Let $p' = (p-1)/2^{J_p}$, $q' = (q-1)/2^{J_q}$ be odd integers. Assume without loss of generality that $J_p \leq J_q$.

Let $u \in \mathbb{Z}_n^*$ be chosen at random. With probability $1/2$, u is a square modulo p, and so

$$u^{2^{J_p-1}p'q'} = (v^2)^{2^{J_p-1}p'q'} = v^{2^{J_p}p'q'} = (v^{q'})^{p-1} \equiv 1 \bmod p$$

With non-negligible probability, u is a generator modulo q, and so $u^{2^{J_p-1}p'q'} \not\equiv 1 \bmod q$, since $q - 1 = 2^{J_q}q' \nmid 2^{J_p-1}q'p'$ because $J_q \geq J_p$. Using the Chinese Remainder theorem, it is easy to see that these events are independent, and therefore the probability that a random u is a square modulo p, and a generator modulo q, is non-negligible.

Therefore, if J_p were known, we would get a value $V = u^{2^{J_p-1}p'q'} \bmod n$ such that $V \equiv 1 \bmod p$ and $V \not\equiv 1 \bmod q$, and therefore $p \leq \gcd(V - 1, n) < n$, and so $\gcd(V - 1, n)$ is a non-trivial divisor of n.

But there are only $\log n$ possibilities for J_p, and therefore we can try them all, and we will be done. □

Corollary 4.1.6. *Let n be an integer. Let e such that $\gcd(e, \phi(n)) = 1$ be given. Given any value x such that $\phi(n) \mid x$, one can efficiently compute v such that $v^e \equiv u \bmod n$.*

For special RSA moduli, it is also true that given $x > 4$ such that $x \mid \phi(n)$, one can factor n. The following lemma and proof are straightforward:

Lemma 4.1.7. *Let $n = pq$ be a special RSA modulus, i.e., $p = 2p'+1$ and $q = 2q'+1$, and p' and q' are primes. Suppose a value $x > 4$ is given such that $x \mid \phi(n) = 4p'q'$. Then it is possible to efficiently factor n.*

Proof. Suppose $x = 2^I p'$, for $I \in \{0, 1, 2\}$. Then factoring n is immediate. The more difficult case is $x = 2^I p'q'$. This immediately gives us $\phi(n)$. By Lemma 4.1.5, we are done. $\square$

Corollary 4.1.8. *Given a special RSA modulus n, and an integer x such that*

$$\gcd(\phi(n), x) > 4$$

, one can efficiently factor n.

The following lemma is due to Shamir:

Lemma 4.1.9. *Let an integer n be given. Suppose that we are given the values $u, v \in \mathbb{Z}_n^*$ and $x, y \in \mathbb{Z}$, $\gcd(x, y) = 1$ such that $v^x \equiv u^y \bmod n$. Then there is an efficient procedure to compute the value z such that $z^x = u \bmod n$.*

Proof. Since $\gcd(x, y) = 1$, by the extended GCD algorithm, find integers a and b such that $ax + by = 1$. Let $z = u^a v^b \bmod n$. Then

$$z^x = u^{ax}(v^x)^b = u^{ax}(u^y)^b = u^{ax+by} = u \bmod n$$

$\square$

The following lemma is a generalization of Lemma 4.1.9, in that we are not restricted to the case where $\gcd(x, y) = 1$. However, we were only able to show this generalization for special RSA moduli.

Lemma 4.1.10. *Let a special RSA modulus $n = pq$, $p = 2p'+1$, $q = 2q'+1$, be given. Suppose we are given the values $u, v \in QR_n$ and $x, y \in Z$, $\gcd(x, y) < x$ such that $v^x \equiv u^y \bmod n$. Values z, $w > 1$ such that $z^w \equiv u \bmod n$ can be computed efficiently.*

Proof. Let $c = \gcd(x, y)$. If $\gcd(4c, \phi(n)) > 4$, then by Corollary 4.1.8, we factor n. Otherwise, it must be the case that $\gcd(c, p'q') = 1$. Therefore, there exists a value $d \in \mathbb{Z}_{p'q'}^*$ such that $cd \equiv 1 \bmod p'q'$.

By the extended GCD algorithm, find integers a and b such that $ax + by = c$. Note that

$$v^{x/c} \equiv (v^x)^d \equiv (u^y)^d \equiv u^{y/c} \bmod n$$

Then, by Lemma 4.1.9, we find an element z such that $z^{x/c} = u$ and we obtain $e = x/c$ and z. $\square$

4.2 The Basic Scheme

Key generation. On input 1^k, choose a special RSA modulus $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$. Choose, uniformly at random, $a, b, c \in QR_n$. Output $PK = (n, a, b, c)$, and $SK = p$. Let ℓ_n denote the length of the modulus n. $\ell_n = 2k$.

Message space. Let ℓ_m be a parameter. The message space consists of all binary strings of length ℓ_m. Equivalently, it can be thought of as consisting of integers in the range $[0, 2^{\ell_m})$.

Signing algorithm. On input m, choose a random prime number $e > 2^{\ell_m+1}$ of length $\ell_e = \ell_m + 2$, and a random number s of length $\ell_s = \ell_n + \ell_m + l$, where l is a security parameter. Compute the value v such that

$$v^e \equiv a^m b^s c \bmod n$$

Output (e, s, v).

Verification algorithm. To verify that the tuple (e, s, v) is a signature on message m in the message space, check that $v^e \equiv a^m b^s c \bmod n$, and check that $2^{\ell_e - 1} < e < 2^{\ell_e}$.

Efficiency analysis. In practice, $\ell_n = 1024$ is considered secure. As for the message space, if the signature scheme is intended to be used directly for signing messages, then $\ell_m = 160$ is good enough, since, given a suitable collision-resistant hash function, such as SHA-1, one can first hash a message to 160 bits, and then sign the resulting value. Then $\ell_e = 162$. The parameter l guides the statistical closeness of the simulated distribution to the actual distribution, hence in practice, $l = 160$ is sufficient, and therefore $\ell_s = 1024 + 160 + 160 = 1346$.

Therefore, this signature scheme requires one short (160-bit) and two long (1024 and 1346) exponentiations for signing, while verification requires two short (162 and 160 bits) and one long (1346-bit) exponentiations. This is not as efficient as the Cramer-Shoup signature, which requires three short (i.e., 160-bit) and one long (i.e., 1024-bit) exponentiations for signing, and four short exponentiations for verifying. However, our signature scheme has other redeeming qualities.

4.3 Proof of Security

We will show that the signature scheme in Section 4.2 is secure. Note that a forgery (m, s, e, v) may have value e of length ℓ_e not necessary a prime number.

Suppose that a forger's algorithm $\mathcal{F}$ is given. Suppose $\mathcal{F}$ makes K signature queries in expectation before it outputs a successful forgery. By the Markov inequality, half the time, $\mathcal{F}$ needs $2K$ or fewer queries. Without loss of generality, we may assume that K is known (since if it is not, then it can be estimated experimentally).

Suppose that the output of $\mathcal{F}$ is (m, s, e, v). Let us distinguish three types of outputs.

- *Type 1:* The forger's exponent e is relatively prime to all the exponents output by the signature oracle.
- *Type 2:* The forger's value e is not relatively prime to some previously seen exponent e_i and the root v is different from the corresponding root v_i.
- *Type 3:* The forger's values e is not relatively prime to some previously seen exponent e_i, $v = v_i$, but the tuple (m, s) is new.

By $\mathcal{F}_1$ (resp., $\mathcal{F}_2$, $\mathcal{F}_3$) let us denote the forger who runs $\mathcal{F}$ but then only outputs the forgery if it is of Type 1 (resp., Type 2, Type 3). The following lemma is clear by the standard hybrid argument:

Lemma 4.3.1. *If the forger $\mathcal{F}$ success probability is ϵ, then one of $\mathcal{F}_1$, $\mathcal{F}_2$, or $\mathcal{F}_3$ succeeds with probability at least $\epsilon/3$.*

Next, we will show that under the strong RSA assumption, success probabilities for each of $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ are negligible.

Lemma 4.3.2. *Under the Strong RSA assumption, the success probability of $\mathcal{F}_1$ is negligible. More strongly, if $\mathcal{F}_1$ succeeds with probability ϵ_1 in expected time $O(p_1(k))$, using expected number $Q(k)$ queries, then there is an algorithm that runs in time $O(p_1(k))$ and solves the flexible RSA problem with probability $\epsilon/4$.*

Proof. Suppose $\mathcal{F}_1$ succeeds with non-negligible probability. Let is show that this contradicts the Strong RSA assumption.

Recall that Q is the expected number of queries, and by the Markov inequality, half the time, $\mathcal{F}_1$ will need at most $2Q = O(p_1(k))$ queries.

Let us construct an algorithm $\mathcal{A}$ such that, on input a strong RSA instance (n, u), and with access to $\mathcal{F}_1$, $\mathcal{A}$ will output (v, e) such that $e > 1$ and $v^e \equiv u \bmod n$.

Now $\mathcal{A}$ plays the role of the signer, as follows:

Key generation: Choose $2Q$ prime numbers $e_1, \ldots, e_{2Q}$. Choose, at random, value $r_1 \in \mathbb{Z}_{n^2}$ and $r_2 \in \mathbb{Z}_{n^2}$. Let $a = u^E$, where $E = \prod_{i=1}^{2Q} e_i$, $b = a^{r_1}$, $c = a^{r_2}$. Let (n, a, b, c) be the public key.

Signing: Upon receiving the i^{th} signature query m_i, choose a value s_i of length ℓ_s uniformly at random. Output (e_i, s_i, v_i), where $v_i = a_i^{m_i} b_i^{s_i} c_i$, where $a_i = u^{E/e_i}$, $b_i = a_i^{r_1}$, $c_i = a_i^{r^2}$. (In other words, a_i, b_i and c_i are values such that $a_i^{e_i} = a$, $b_i^{e_i} = b$, and $c_i^{e_i} = c$). Note that (e_i, s_i, v_i) constitute a valid signature:

$$\begin{aligned} v_i^{e_i} &= (a_i^{m_i} b_i^{s_i} c_i)^{e_i} \\ &= a^{m_i} b^{s_i} c \ . \end{aligned}$$

Note that the public key and all the signatures are distributed correctly.

Suppose the forgery is (e, s, v) on message m, and $e > 4$. This gives us $v^e = a^m b^s c = u^{E(m+r_1 s+r_2)}$. Observe that if it is the case that e and $E(m + r_1 s + r_2)$ are relatively prime, then by Shamir's trick (Lemma 4.1.9) we are done. We will not necessarily have this nice case, but we will show that things will be good enough.

First, by assumption on $\mathcal{F}_1$ we have $\gcd(e, E) = 1$. Second, we will use the following claim:

Claim. *With probability at least* $1/2$ *over the random choices made by the reduction, it is the case that either* $\gcd(e, m + r_1 s + r_2) < e$*, or, on input* e*, one can efficiently factor* n*.*

The claim immediately gives us the conditions of Lemma 4.1.10, which gives us values v, $w > 1$ such that $v^w \equiv u \bmod n$.

We must now prove the claim. Suppose that it is false. Then the probability that $\gcd(e, m + r_1 s + r_2) < e$ is less than $1/2$. Suppose $r_2 = x + \phi(n)z$. Note that z is independent of the view of the adversary and it is chosen statistically indistinguishable from random over $\mathbb{Z}_n$. Therefore, if with probability greater than $1/2$, $e \mid m+r_1 s+r_2$, this holds for greater than $1/2$ of all the possible choices for z. Then there exists a value $z_0 \in \mathbb{Z}_n$ such that $e \mid (m+r_1 s+x+\phi(n)z_0)$ and $e \mid (m+r_1 s+x+\phi(m)(z_0+1))$. Then it holds that $e \mid \phi(n)$ with probability at least $1/2$. Therefore, with probability $1/2$, we either factor n (by Lemma 4.1.5) or $\gamma = \gcd(e, m + r_1 s + r_2) < e$.

Therefore, $\mathcal{A}$ succeeds in breaking the strong RSA assumption whenever the number of queries does not exceed $2Q$ (which happens with probability $1/2$ by Markov inequality) and whenever the conditions of the claim are satisfied (which happens with probability $1/2$, independently on the number of queries), and so the success probability of $\mathcal{A}$ is $\epsilon_1/4$, while its running time is $O(p_1(k))$. $\square$

Lemma 4.3.3. *Under the Strong RSA assumption, the success probability of* $\mathcal{F}_3$ *is negligible. More strongly, if* $\mathcal{F}_3$ *succeeds with probability* ϵ_1 *in expected time* $O(p_1(k))$*, using expected number* $Q(k)$ *queries, then there is an algorithm that runs in time* $O(p_3(k))$ *and succeeds with probability* $\epsilon_3/2$*.*

Proof. The input to the reduction is (n, e), an instance of the flexible RSA problem.

The key generation and signatures in this case are as in the proof of Lemma 4.3.2. The signatures (m, s, e, v) and (m_i, s_i, e_i, v) give us the following: Note that from $\gcd(e_i, e) \neq 1$ and the fact that $2^{\ell_e} > e, e_i > 2^{\ell_e}$ it follows that $e_i = e$ and thus $a^m b^s \equiv a^{m_i} b^{s_i}$. This implies that $m + r_1 s \equiv m_i + r_1 s_i \bmod \phi(n)$. Since $(m, s) \neq (m_i, s_i)$, and $r_1 > m$, $r_1 > m_i$, $m + r_1 s \neq m_i + r_1 s_i$. Therefore, $\phi(n) \mid m + r_1 s - m_i + r_1 s_i \neq 0$, and so by Lemma 4.1.6, we are done.

The probability that $\mathcal{A}$ succeeds in breaking the Strong RSA assumption is at least $\epsilon/2$ because with probability at least $1/2$, the forger $\mathcal{F}_3$ will need at most $2Q$ queries. $\square$

Lemma 4.3.4. *Under the Strong RSA assumption, the success probability of* $\mathcal{F}_2$ *is negligible. More strongly, if* $\mathcal{F}_2$ *succeeds with probability* ϵ_2 *in expected time* $O(p_2(k))$*,* *using expected number $Q(k)$ queries, then there is an algorithm that runs in time $O(p_2(k))$ and succeeds with probability $\epsilon(k)/8Q(k)$.*

Proof. The input to the reduction is (n, u), an instance of the flexible RSA problem.

Choose a value $1 \leq i \leq 2Q(k)$ uniformly at random. With probability $1/2Q(k)$, the exponent e will be the same as in the i'th query to the signature oracle.

The idea of this reduction is that we will set up the public key in such a way that, without knowledge of the factorization of n, it is possible to issue correctly distributed signatures on query $j \neq i$, in almost the same manner as in the proof of Lemma 4.3.2. For query i, the answer will be possible only for a specific, precomputed value of $t = m\alpha + s$, where $\alpha = \log_b a$.

Key Generation: Choose $2Q(k)$ random primes $\{e_j\}$. Let $E = (\prod_{j=1}^{2Q} e_j)/e_i$. Choose $\alpha, \beta \in \mathbb{Z}_{2n}$ uniformly at random. Choose a random value t of length ℓ_s. Let $b = u^E \bmod n$. Let $a = b^\alpha \bmod n$, and $c = b^{e_i\beta - t}$. Let (n, a, b, c) be the public key.

Signing: Upon receiving the j^{th} signature query m_j, $j \neq i$, choose a value s_j of length ℓ_s uniformly at random. Output (e_j, s_j, v_j), where $v_j = a_j^{m_j} b_j^{s_j} c_j$, where $b_j = u^{E/e_j}$, $a_j = b_j^\alpha$, $c_j = b_j^{e_i\beta - t}$. (In other words, a_j, b_j and c_j are values such that $a_j^{e_j} = a$, $b_j^{e_j} = b$, and $c_j^{e_j} = c$). Note that (e_j, s_j, v_j) constitute a valid signature.

Upon receiving the i^{th} signature query m_i, compute the value $s_i = t - \alpha m_i$. $v_i = b^\beta$. Note that $v_i^{e_i} = b^{e_i\beta - t + t} = b^t c = a^{m_i} b^{s_i} c$. Note that if ℓ_s is sufficiently long (e.g., $\ell_n + \ell_m + l$, where ℓ_n is the length of the modulus n, ℓ_m is the length of a message, and l is a security parameter), s_i is distributed statistically close to uniform over all strings of length ℓ_s.

Suppose the forgery gives (m, s, e, v) such that e is not relatively prime to e_i. Then $e = e_i$ because e is of length ℓ_e. Also, as the verification is successful for the forgery as well as for the i'th query, $v^{e_i} = a^m b^s c = b^{m\alpha + s + e_i\beta - t} = b^{(m-m_i)\alpha + (s-s_i) + e_i\beta}$. Let $\alpha = \alpha_1\phi(n) + \alpha_2$. Note that with probability very close to 1, the value $\alpha_1 \in \{0, 1\}$. Also, note that α_1 is independent on the adversary's view. Therefore, if the probability that $e_i \mid (m - m_i)\alpha + (s - s_i) + e_i\beta$ is non-negligibly higher than $1/2$, then it is the case that $e_i \mid (m - m_i)\alpha_2 + (s - s_i) + e_i\beta$ and $e_i \mid (m - m_i)(\phi(n) + \alpha_2) + (s - s_i) + e_i\beta$, and therefore $e_i \mid (m - m_i)$ (note that by definition of a forgery $m \neq m_i$). If the length ℓ_e is two bits longer than the length of a valid message ℓ_m, this is impossible.

Therefore, with probability at least $1/2$, the following condition is met: $e_i \nmid (m\alpha + s + e_i\beta - t) = \gamma$. Therefore, we have the following: $v^{e_i} \equiv u^{E\gamma}$, where $\gcd(e_i, \gamma) = 1$, and so by Lemma 4.1.9, we compute e_i'th root of u.

Therefore, $\mathcal{A}$ succeeds in solving an instance of the flexible RSA problem when $2Q$ is a sufficient number of queries (with probability at least $1/2$ this is the case by the Markov inequality), when i is chosen correctly (this is the case with probability $1/2Q$) and when the condition is met (with probability $1/2$ independently of everything else). Therefore, $\mathcal{A}$ succeeds with probability $\epsilon_2/8Q$ in time $p_2(k)$. $\square$

Since the number of signature queries is at most the running time, we have shown the following theorem:

Theorem 4.3.5. *Our signature scheme is secure under the Strong RSA assumption. More precisely, if a forger breaks our signature scheme in time $p(k)$ with probability $\epsilon(k)$, then the Strong RSA assumption can be broken in time $O(p(k))$ with probability $\Omega(\epsilon(k)/p(k))$.*

4.4 The Scheme for Blocks of Messages

Suppose, instead of signing a single message m, one wants to sign a block of L messages $m_1, \dots, m_L$. For most applications, these two problems are equivalent, since in order to sign such a block, it is sufficient to hash it to a small message m using a collision-resistant hash function, and then to sign m using a one-message signature scheme.

However, if the application we have in mind involves proving knowledge of a signature and proving relations among certain components of $m_1, \dots, m_L$, it may be helpful to have a signature scheme especially designed to handle blocks of L messages. Here, we propose one such scheme. Note that in our proposed scheme, the value L is fixed; i.e., the public key is designed to work for a specific value L.

Key generation. On input 1^k, choose a special RSA modulus $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$. Choose, uniformly at random, $a_1, \dots, a_L, b, c \in QR_n$. Output $PK = (n, a_1, \dots, a_L, b, c)$, and $SK = p$. Let ℓ_n denote the length of the modulus n. $\ell_n = 2k$.

Message space. Let ℓ_m be a parameter. The message space consists of all binary strings of length ℓ_m. Equivalently, it can be thought of as consisting of integers in the range $[0, 2^{\ell_m})$.

Signing algorithm. On input $m_1, \dots, m_L$, choose a random prime number $e > 2^{\ell_m+1}$ of length $\ell_e = \ell_m + 2$, and a random number s of length $\ell_s = \ell_n + \ell_m + l$, where l is a security parameter. Compute the value v such that

$$v^e \equiv a_1^{m_1} \dots a_L^{m_L} b^s c \bmod n$$

Output the tuple (e, s, v).

Verification algorithm. To verify that the tuple (e, s, v) is a signature on message m, check that $v^e \equiv a_1^{m_1} \dots a_L^{m_L} b^s c \bmod n$, and check that $e > 2^{\ell_e - 1}$.

Theorem 4.4.1. *The signature scheme above is secure for blocks of messages, under the Strong RSA assumption.*

Proof. We will show a reduction from a forger $\mathcal{F}$ for this signature scheme, to an algorithm that breaks the basic signature scheme from Section 4.2. The reduction receives as input the public key of the basic scheme: (n, a, b, c), and has oracle access to the basic signature scheme.

Key Generation. Choose an index $I \in [1, L]$ at random. Let Let $a_I = a$. For $j \in [1, L]$, $j \neq I$, choose and integer r_j from $\mathbb{Z}_{2n}$ uniformly at random, and let $a_j = b^{r_j}$. Output the public key $(n, a_1, \ldots, a_L, b, c)$.

Signature generation. On input signature query $(m_1, \ldots, m_L)$, ask the signature oracle for a signature on the value m_I. Upon receiving the value (s, v, e), let $s' = s - \sum_{j \neq I} r_j m_j$. The resulting signature is (s', v, e). It is easy to see that it is distributed statistically close to the right distribution provided that $\ell_s = \ell_n + \ell_m + l$, where l is sufficiently long.

Suppose the adversary's forgery is $(m_1, \ldots, m_L, s, v, e)$. Then $(s + \sum_{j=2}^{L} r_j m_j, v, e)$ is a valid signature in the basic scheme on message m_I. Suppose the signature's e is different from any previously seen e_i. Then we are done: we have created forger $\mathcal{F}_\infty$ and contradicted Lemma 4.3.2. So suppose this $e = e_i$. But $(m_1, \ldots, m_L) \neq (m_1^i, \ldots, m_L^i)$. With probability at least $1/L$, it is the case that $m_I \leq m_I^i$, and we have created a forged signature on message m_I.

Overall, if the adversary $\mathcal{F}$ succeeds in time $p(k)$ with probability ϵ using $Q(k)$ queries, the reduction succeeds against the basic scheme in time $p(k)$ with probability at least $\epsilon/2L$ also using $Q(k)$ queries. □

4.5 Protocol Preliminaries

In this section, we will show how to construct a secure protocol for signing a committed protocol as defined in Section 3.2.1, under our basic signature scheme described in Section 4.2.

4.5.1 Commitment Scheme

The following commitment scheme is due to Fujisaki and Okamoto [FO98] and elaborated on by Damgård and Fujisaki [DF01]. Its security relies on the hardness of factoring.

Key generation The public key consists of a special RSA modulus n of length ℓ_n, and $h \leftarrow QR_n$, $g \leftarrow \langle h \rangle$, where $\langle h \rangle$ is the group generated by h.

Commitment The commitment $\texttt{Commit}(PK, x, r)$ for inputs of length ℓ_x and randomness $r \in \mathbb{Z}_n$, is defined as follows: $\texttt{Commit}((n, g, h), x, r) = g^x h^r \bmod n$.

Lemma 4.5.1. *The commitment scheme described above is statistically hiding and computationally binding if factoring is hard.*

Proof. Let us first show the hiding properties of the scheme. Let x be a string of length ℓ_x. Let us show that for all correct keys $PK = (n, g, h)$ the distribution $(r \leftarrow \mathbb{Z}_n; c = \texttt{Commit}(PK, x, r) \ : \ c)$ is statistically close to $(c \leftarrow < h > \ : \ c)$. Since x is fixed, and $g \in < h >$, we can equivalently show that these two experiments induce statistically close distributions on $c' = c/g^x$, i.e., $(r \leftarrow \mathbb{Z}_n; c' = h^r \bmod n \ : \ c)$ is statistically close to $(c' \leftarrow< h > \ : \ c)$. This follows because r was chosen from $\mathbb{Z}_n$, $r = r_1\phi(n) + r_2$, where r_2 is statistically close to uniform to $\mathbb{Z}_{\phi(n)}$.

Now let us show the binding property. Let us show that an adversary breaking the binding property allows us to factor. Suppose our input is a special RSA modulus n. Let α be a random number of length $\ell_n + 1$. Note that the resulting public key is distributed statistically close to the right distribution. Suppose that we are given a commitment c, and two ways of opening it: $g^x h^r = c = g^{x'} h^{y'} \bmod n$, such that $x \neq x'$. Let $X = \alpha x + r$, $X' = \alpha x' + r'$. It follows $h^{X-X'} = 1 \bmod n$. Note that since r and r' are only of length ℓ_n, while α is of length $\ell_n + 1$, it must be the case that $X \neq X'$. Since $h^{X-X'} = 1 \bmod n$, it follows that $|h|$ divides $X - X'$, and so $\gcd(X - X', \phi(n)) > 4$, and so we are done by Corollary 4.1.8, as long as we know α. □

Remark. Note that for the hiding property, the only requirement from the public key is that $g \in \langle h \rangle$, and so the public key can be generated adversarially provided that this requirement holds.

We want to guarantee the security of this commitment scheme even when the key is provided by the adversary. To ensure that, we add the following key certification procedure: the party that generated the commitment keys (n, g, h) must prove existence of the discrete logarithm $\log_h g \bmod n$. This can be done using well-known techniques that we elaborate on in the sequel (cf. [DF01]).

4.5.2 Summary of the Protocols Used

All protocols assume that the public parameters were fixed after the adversary was fixed (otherwise a non-uniform adversary may know secret keys).

The following known protocols are secure under the strong RSA assumption:

- Proof of knowledge of discrete logarithm representation modulo a composite [FO97]. That is to say, a protocol with common inputs (1) a $(n, g_1, \dots, g_m)$, where n is a special RSA modulus and $g_i \in QR_n$ for all $1 \leq i \leq m$; and (2) a value $C \in QR_n$, the prover proves knowledge of values $\alpha_1, \dots, \alpha_m$ such that $C = \prod_{i=1}^m g_i^{\alpha_i} \bmod n$.

- Proof of knowledge of equality of representation modulo two (possibly different) composite moduli [CM99b]. That is to say, a protocol with common inputs $(n_1, g_1, \dots, g_m)$ and $(n_2, h_1, \dots, h_m)$ and the values $C_1 \in QR_{n_1}$ and $C_2 \in QR_{n_2}$, the prover proves knowledge of values $\alpha_1, \dots, \alpha_m$ such that $C_1 = \prod_{i=1}^m g_i^{\alpha_i} \bmod n_1$. and $C_2 = \prod_{i=1}^m h_i^{\alpha_i} \bmod n_2$.

- Proof that a committed value is the product of two other committed values [CM99a]. That is to say, a protocol with common inputs (1) a commitment key (n, g, h) as described in Section 4.5.1; and (2) values C_a, C_b, C_{ab} in QR_n, where the prover proves knowledge of the integers α, β, ρ_1, ρ_2, ρ_3 such that $C_a = g^\alpha h^{\rho_1} \bmod n$, $C_b = g^\beta h^{\rho_2} \bmod n$, and $C_{ab} = g^{\alpha\beta} h^{\rho_3} \bmod n$.

- Proof that a committed value lies in a given integer interval [Lip01]. That is to say, a protocol with common inputs (1) a commitment key (n, g, h) as described in Section 4.5.1; (2) a value $C \in QR_n$; (3) integers a and b, where the prover proves knowledge of the integers α and ρ such that $C = g^\alpha h^\rho \bmod n$ and $a \le \alpha \le b$.

Notation: In the sequel, we will sometimes use the notation introduced by Camenisch and Stadler[CS97] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance,

$$PK\{(\alpha,\beta,\gamma) : y = g^\alpha h^\beta \ \wedge\ \tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma \ \wedge\ (u \le \alpha \le v)\}$$

denotes a "*zero-knowledge* P*roof of* K*nowledge of integers* α*,* β*, and* γ *such that* $y = g^\alpha h^\beta$ *and* $\tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma$ *holds, where* $v < \alpha < u$*,*" where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. The convention is that Greek letters denote quantities the knowledge of which is being proved, while all other parameters are known to the verifier. Using this notation, a proof-protocol can be described by just pointing out its aim while hiding all details.

We will now present the protocols themselves.

4.5.3 Proof of Knowledge of Commitment Opening

We now show a proof of knowledge protocol for proving knowledge of a committed value. We give a Σ-protocol proof of knowledge of representation; it can be converted into a general zero-knowledge proof using standard techniques based on trapdoor commitment schemes, described in Section 2.6.1. All protocols are in the public-parameter model.

Common inputs: security parameters ℓ_r and ℓ_k;
(n, g, h): a public key of the commitment;
a commitment $X = g^x h^y \bmod n$.
Prover's inputs: x and y.

$P \longrightarrow V$ Choose ℓ_r-bit r_1, r_2 at random. Send $R = g^{r_1} h^{r_2}$ to the Verifier.

$P \longleftarrow V$ Generate an ℓ_c-bit challenge c.

$P \longrightarrow V$ Compute the response $s_1 = r_1 + cx$, $s_2 = r_2 + cy$.

Acceptance: The verifier accepts if $RX^c = g^{s_1} h^{s_2}$.

Figure 4-1: Σ-Protocol Proof of Representation Modulo a Composite

Lemma 4.5.2. *Under the Strong RSA assumption, the protocol described in Figure 4-1 is a Σ-protocol zero-knowledge proof of knowledge of witness (x, y) for the relation*

$$R_{n,g,h} = \{X, (x, y) \ : \ X = g^x h^y \bmod n\}$$

.

Proof. The completeness is obvious: if the prover follows the protocol, then $RX^c = g^{r_1} h^{r_2} (g^x h^y)^c = g^{r_1+cx} h^{r_2+cy} = g^{s_1} h^{s_2}$.

The Σ zero knowledge property follows because suppose the challenge c is fixed in advance. Then the simulator chooses s_1 and s_2 at random, and sets $R = g^{s_1} h^{s_2} / X^c$. If $\ell_r = \ell_c + \max(\ell_x, \ell_y, \ell_n) + l$ where l is a security parameter, then choosing s_1 and s_2 at random is statistically indistinguishable.

The extraction follows because suppose that with non-negligible probability over the choice of the challenge c, the prover causes the verifier to accept. Then the verifier can get two accepting transcripts based on the same R, with challenges c and c'. This results in the equations $RX^c = g^{s_1} h^{s_2}$ and $RX^{c'} = g^{s'_1} h^{s'_2}$.

Claim. *Let $\delta = c - c'$. Then $\delta \mid (s_1 - s'_1)$ and $\delta \mid (s_2 - s'_2)$, under the Strong RSA assumption.*

Proof of claim. Suppose that (n, g) are an instance of the flexible RSA problem. The extractor sets up the public key of the commitment by choosing a random α of length $\ell_s + 3$ and letting $PK = (n, g, h)$, where $h = g^\alpha \bmod n$.

Now the extractor obtains equations $RX^c = g^{s_1} h^{s_2}$ and $RX^{c'} = g^{s'_1} h^{s'_2}$. Let $S = (s_1 - s'_1) + \alpha(s_2 - s'_2)$, $\delta = c - c'$, and suppose for contradiction that $\delta \nmid (s_2 - s'_2)$ or $\delta \nmid S$.

The equations give us that $X^\delta \equiv g^S \bmod n$. Therefore, if $\delta \nmid S$, we break the Strong RSA assumption by Lemma 4.1.10 and we are done. So we wish to show that this happens with non-negligible probability, say, at least 1/2.

Suppose, then, that, $\delta \mid S$ and $\delta \nmid (s_2 - s'_2)$. Let $\alpha = \alpha_1 \phi(n) + \alpha_2$. Since α_1 is independent of the adversary's view, for some choice of α_1, $\delta \mid S = (s_1 - s'_1) + (\alpha_1 \phi(n) + \alpha_2)(s_2 - s'_2)$ and $\delta \mid S' = (s_1 - s'_1) + ((\alpha_1 + 1)\phi(n) + \alpha_2)(s_2 - s'_2)$. Therefore, δ divides $(S' - S)$, i.e., $\delta \mid \phi(n)(s_2 - s'_2)$. Since $\delta \nmid (s_2 - s'_2)$, it follows that $\gcd(\delta, \phi(n)) > 4$ and we are done by Corollary 4.1.8.

Thus, it must be that $\delta \mid (s_2 - s'_2)$ and $\delta \mid S$. It follows that $\delta \mid (s_1 - s'_1)$. □

From the claim, it follows that, if we let $x = (s_1 - s'_1)/\delta$, and $y = (s_2 - s'_2)/\delta$, then (x, y) are a representation of X in bases g and h, modulo n. Note that the extracted values are not necessarily positive. □

Remark. It is easy to see how to generalize this protocol and the corresponding extractor to the case where instead of g, there are several bases $(g_1, \ldots, g_u)$.

4.5.4 Proof of Equality of Committed Values

It will sometimes be necessary to prove equality of two committed values, possibly committed to under different commitment keys. For that, we exhibit the following Σ-protocol:

Common inputs: security parameters ℓ_r and ℓ_k;
(n_1, g_1, h_1) and (n_2, g_2, h_2): public keys of the commitment;
a commitment $X = g_1^x h_1^y \bmod n_1$ and $Y = g_2^x h_2^z \bmod n_2$.
Prover's inputs: x, y, z.

$P \longrightarrow V$ Choose ℓ_r-bit r_1, r_2, r_3 at random. Send $R_1 = g_1^{r_1} h_1^{r_2} \bmod n_1$, $R_2 = g_2^{r_2} h_2^{r_3} \bmod n_2$ to the Verifier.

$P \longleftarrow V$ Generate an ℓ_k-bit challenge c.

$P \longrightarrow V$ Compute the response $s_1 = r_1 + cx$, $s_2 = r_2 + cy$, $s_3 = r_3 + cz$.

Acceptance: The verifier accepts if $R_1 X^c = g_1^{s_1} h_1^{s_2} \bmod n_1$ and $R_2 Y^c = g_1^{s_1} h_2^{s_3} \bmod n_2$.

Figure 4-2: Σ-Protocol Proof of Equality of Representation Modulo a Composite

Lemma 4.5.3. *Under the Strong RSA assumption, the protocol described in Figure 4-2 is a Σ-protocol zero-knowledge proof of knowledge of witness (x, y, z) for the relation $R_{n,g_1,h_1,g_2,h_2} = \{X, Y, (x, y, z) \ : \ X = g_1^x h_1^y \bmod n \wedge Y = g_2^x h_2^z\}$.*

Proof. The completeness and zero-knowledge properties follow in the same way as in the proof of Lemma 4.5.2. The extraction property follows because, as in the proof of Lemma 4.5.2, it has to be the case that, if we obtain two accepting transcripts both based on (R_1, R_2), with challenges c and $c' = c + \delta$, we get $x = (s_1 - s_1')/\delta$, $y = (s_2 - s_2')/\delta$, and $z = (s_3 - s_3')/\delta$, all over the integers, under the Strong RSA assumption. $\square$

Remark. Note that, while the protocol in Figure 4-1 only worked for a commitment key chosen in a trusted setup phase uniformly at random, the proof of security for the protocol in Figure 4-2 goes through so long as just *one* of the commitment keys was chosen in this fashion. The other one may be chosen in an arbitrary way, with various parameters known to the adversary. This observation is due to Camenisch and Michels [CM99a].

4.5.5 Proof That a Committed Value Lies in an Interval

The last proof of knowledge protocol we will need in this section is a proof that a committed value lies in an interval. This is due to Lipmaa [Lip01], using earlier work of Boudot[Bou00] and of Chan, Frankel and Tsiounis [CFT98].

Lipmaa's is a simple and elegant result that gives statistical zero-knowledge protocols that a committed integer x lies within a given interval (a, b). The idea is as follows:

1. The problem of showing that $x \in (a, b)$ is reduced to the problem to showing that a number is positive: $x \in (a, b) \Leftrightarrow x - a > 0 \wedge b - x > 0$. If $C = g^x h^r \bmod n$, then let $C_a = C/g^a \bmod n$, and $C_b = g^b/C \bmod n$. Now we have to show that C_a and C_b are both commitment to positive integers.

2. It is a well-known fact due to Lagrange that any non-negative integer can be represented as a sum of four squares. Rabin and Shallit [RS86] gave an efficient algorithm for computing this representation. Thus the task is to show that a committed number is a sum of squares.

3. Showing that a committed number is a sum of several other committed values is straightforward. Let X and Y be two commitments. To show that a commitment Z is to the value $z = x + y$, it is sufficient to show that Z is a commitment to the same value as XY.

4. Prove that a committed number is a square, using the protocol for showing that a committed number is a product of two other committed values, below.

The only remaining protocol to give is for proving that a committed number is the product of two other committed numbers.

Common inputs: security parameters ℓ_r and ℓ_k;
(n, g, h): public key of a commitment scheme;
commitments $C_x = g^x h^{r_x} \bmod n$, $C_y = g^y h^{r_y} \bmod n$
and $C_z = g^{xy} h^{r_z}$.
Prover's inputs: x, y, r_x, r_y, r_z.

$P \longleftrightarrow V$ Using the protocol given in Figure 4-2, carry out the following proofs, using security parameters ℓ_r and ℓ_k:

1. $PK\{(\alpha, \rho_x) : C_x = g^\alpha h^{\rho_x}\}$.
2. $PK\{(\beta, \rho_y, \rho') : C_y = g^\beta h^{\rho_y} \wedge C_z = C_x^\beta h^{\rho'}\}$

Figure 4-3: Σ-Protocol for Proving That a Committed Number Is the Product of Two Other Committed Numbers

Lemma 4.5.4. *Under the Strong RSA assumption, the protocol described in Figure 4-2 is a Σ-protocol zero-knowledge proof of knowledge of witness* $(x, y, r_x, r_y, r_z))$ *for the relation* $R_{n,g,h} = \{X, Y, Z, (x, y, r_x, r_y, r_z) \; : \; X = g^x h^{r_x} \bmod n \wedge Y = g^y h^{r_y} \bmod n \wedge Z = g^{xy} h^{r_z}\}$.

Proof. First, let us give the knowledge extractor. Since both of the steps of the proof are proofs of knowledge of equality of representation, by Lemma 4.5.3, we can extract the values $(\alpha, \beta, \rho_x, \rho_y, \rho')$ such that $C_x = g^\alpha h^{\rho_x}$, $C_y = g^\beta h^{\rho_y}$, and $C_z = C_x^\beta h^{\rho'} = g^{\alpha\beta} h^{\rho_x \beta + \rho'}$. Output the values $(\alpha, \beta, \rho_x, \rho_y, \rho_x\beta + \rho')$.

As for the zero-knowledge property, it follows from the fact that by Lemma 4.5.3, the protocol for proving knowledge and equality of representation is a Σ-protocol. □

The following lemma follows from the discussion in this section:

Lemma 4.5.5. *Under the Strong RSA assumption, Lipmaa's protocol is a* Σ*-protocol zero-knowledge proof of knowledge of witness* (x, r) *for the relation*

$$R_{n,g,h} = \{C, a, b, (x, r) \ : \ C = g^x h^r \wedge x \in (a, b)\}$$

.

4.6 Protocols for the Signature Scheme

In the sequel, we rely on Σ-protocol zero-knowledge proofs of knowledge of representation protocols; they can be converted into a general zero-knowledge proof using standard techniques based on trapdoor commitment schemes (cf. [Dam00]).

4.6.1 Protocol for Signing a Committed Value

Informally, in a protocol for signing a committed value, we have a signer with public key *PK*, and the corresponding secret key *SK*, and a user who queries the signer for a signature. The common input to the protocol is a commitment C for which the user knows the opening (m, r). In the end of the protocol, the user obtains a signature $\sigma_{PK}(m)$, and the signer learns nothing about m.

Lemma 4.6.1. *Figure 4-4 describes a secure protocol for signing a committed value.*

Proof. Let the extractor E be as follows: it runs the extractor for step 4.6.1 of the protocol, and discovers x and r that are of lengths ℓ_x and ℓ_r respectively. It then queries the signature oracle for a signature on x. It obtains the signature (s, e, v). Note that the value s is a random integer of length ℓ_s. The extractor then sets $r' = s - r$. If $\ell_s \geq \ell_n + \ell$, where ℓ is a security parameter guiding the statistical closeness, then the value r' obtained this way will be statistically close to the value r' generated by S, and so statistically, the adversary's view is the same whether talking to the real signer S, or to the extractor E. □

4.6.2 Proof of Knowledge of a Signature

In this section, we give a Σ-protocol zero-knowledge proof of knowledge protocol for showing that a committed values is a signature on another committed value.

Common inputs: Public key (n, a, b, c) of the signature scheme with parameters ℓ_m, ℓ_s, ℓ_e.

Commitment public key (n_C, g_C, h_C), commitment C.

User's input: x and r_C such that $C = g_C^x h_C^{r_C} \bmod n_C$.

Signer's input: Factorization of n.

$U \longleftrightarrow S$ Form commitment $C_x = a^x b^r \bmod n$, and prove that C_x is a commitment to the same value as C.

Prove knowledge of x, r. Prove that $x \in (0, 2^{\ell_m})$, and $r \in (0, 2^{\ell_n})$.

$U \longleftarrow S$ Choose a random r' of length ℓ_s, and a prime number of length ℓ_e. Compute $v = (C_x b^{r'} c)^{1/e}$. Send (r', e, v) to the user.

User's output: Let $s = r + r'$. The user outputs a signature (s, e, v) on x.

Figure 4-4: Signature on a Committed Value

Lemma 4.6.2. *The protocol in Figure 4-5 is a Σ-protocol for proving knowledge of the values (x, r_x, s, e, v) such that $C_x = g^x h^{r_x} \bmod n$ and $\mathtt{Verify}_{PK}(x, s, e, v)$.*

Proof. The completeness property is clear. The zero-knowledge property follows because the simulator can form the commitments at random (since they reveal nothing statistically), and then invoke the simulator for the zero-knowledge proofs of knowledge of representation and belonging to an interval.

We must now show that there exists an extractor that computes a valid signature. Our extractor will invoke the extractor for the proof protocols we use as a building block. If our extractor fails, then we can set up a reduction that breaks the Strong RSA assumption.

Suppose the extractor succeeds and computes $(x, s, e, w, z, r_x, r_s, r_e, r_w, r_z, r)$ such that $C = (C_v)^e h^r$ and $C = a^x b^s c g^z h^r$, and $C_z = g^{ew} h^{r_z}$. Note that this implies that $C_v^e = a^x b^s c g^z$. Let $v = C_v / g^w$. Note that $v^e = C_v^e / g^z = a^x b^s c$. Since we also know that x is of length ℓ_m, s is of length ℓ_s and e is of length ℓ_e, we are done: we output (s, e, v) which is a signature on the message x. □

4.6.3 Protocols for Signatures on Blocks of Messages

We note that straightforward modifications to the protocols above give us protocols for signing blocks of committed values, and to prove knowledge of a signature on blocks of committed values. We also note that, using any protocol for proving relations among components of a discrete-logarithm representations of a group element [Cam98, CM99a, Bra99], can be used to demonstrate relations among components of a signed block of messages. We highlight this point by showing a protocol that allows an off-line double-spending test described in Section 3.4.

Recall that in order to enable an off-line double-spending test, a credential is a signature on a tuple (SK, N_1, N_2), and in a spending protocol (i.e., credential showing) a user reveals the value $b = aSK + N_1 \bmod q$, for some prime number q, and N_2, for a value $a \in \mathbb{Z}_q$ chosen by the verifier. The user must then prove that the value revealed corresponds to the signed block of messages (SK, N_1, N_2).

Here, we describe a mechanism that, combined with our signature scheme for blocks of messages, achieves this. We note that these types of techniques are similar to those due to Brands [Bra99].

We assume that we are given, as public parameters, public values $(p = 2q + 1, g_p)$ where p and q are primes, and $g_p\mathbb{Z}_p$ has order q. We require that $q > \max(2^{\ell_m}, 2^{\ell_N})$ where ℓ_N is the length of the nonce N_1. We also assume that we are given another commitment public key, (n_C, g_C, h_C), same as in the rest of this Chapter.

In order to prove that the values (b, N_2) correspond to the signature on his secret key committed to in commitment C_{SK} the user forms commitments and C_1 to N_1 and C_2 to N_2. He then shows that he has a signature on the block of values committed to by this block of commitments, and that the value committed to in C_1 is of length ℓ_N. He reveals N_2 and shows that C_2 is a valid commitment to N_2. He also carries out the following proof of knowledge protocol:

$$\begin{aligned} PK\{(\alpha, \beta, r_\alpha, r_\beta) \ : \ & g_p^b = (g_p^a)^\alpha g^\beta \ \wedge \\ & C_{SK} = g_C^\alpha h_C^{r_\alpha} \ \wedge \\ & C_1 = g_C^\beta h_C^{r_\beta} \ \} \end{aligned}$$

Lemma 4.6.3. *The protocol described above is a proof of knowledge of a block of values (SK, N_1) such that (SK, N_1, N_2) is signed, and C_{SK} is a commitment to SK, and $b = aSK + N_1 \bmod q$.*

Proof. (Sketch) The completeness and zero-knowledge properties follow in the usual way. We must now address the question of extraction. The fact that we extract (SK, N_1, N_2) and a signature on them follows similarly as for the protocol in Figure 4-5. The only thing we worry about is that $b = aSK + N_1 \bmod q$. Using a knowledge extractor from the proofs of knowledge of equal representations, we extract values α and β of lengths ℓ_m and ℓ_N such that $\log_{g_p}(g_p^b) = b = \alpha a + \beta \bmod q$, (since the order of g_p is q) and α is the value committed to in C_{SK} while β is the value committed to in C_1. Under the strong RSA assumption, it follows that $\alpha = SK$ and $\beta = N_1$, and so $b = SKa + N_1 \bmod q$ as we require. $\square$

Common inputs: Public key (n, a, b, c) of the signature scheme with parameters ℓ_m, ℓ_s, ℓ_e

Public key (g, h) of a commitment scheme modulo n

Commitment public key (n_C, g_C, h_C), commitment C_x

User's input: The values (x, r_x) such that $C_x = g_C^x h_C^{r_x} \bmod n_C$

and x is of length ℓ_m and r_x is of length ℓ_n.

Values (s, e, v), a valid signature on message x.

$U \longleftrightarrow V$ Choose, uniformly at random of length ℓ_n, values w, r_s, r_e, r_w, r_z and r. Compute the following quantities: $C_s = g^s h^{r_s} \bmod n$, $C_e = g^e h^{r_e} \bmod n$, $C_v = v g^w \bmod n$, $C_w = g^w h^{r_w}$, $z = ew$, $C_z = g^z h^{r_z}$. $C = (C_v)^e h^r \bmod n$.

It is important to note that $C = (C_v)^e h^r = v^e g^{we} h^r = (a^x b^s c) g^z h^r \bmod n$.

Send the values $(C_s, C_e, C_v, C_w, C_z, C)$ to the verifier and carry out the following Σ-protocol zero-knowledge proofs of knowledge:

1. Show that C is a commitment in bases (C_v, h) to the value committed to in the commitment C_e:

$$PK\{(\epsilon, \rho, \rho_e) : C = C_v^{\epsilon} h^{\rho} \ \wedge \ C_e = g^{\epsilon} h^{\rho_e}\}$$

2. Show that C/c is also a commitment in bases $((a, b, g), h)$, to the values committed to by commitments C_x,C_s,C_z, and that the power of h here is the same as in C:

$$\begin{aligned} PK\{(\xi, \sigma, \zeta, \epsilon, \rho_x, \rho_s, \rho_z, \rho) \ : \ & C/c = a^{\xi} b^{\sigma} g^{\zeta} h^{\rho} \ \wedge \ C_x = g^{\xi} h^{\rho_x} \ \wedge \\ & C_s = g^{\sigma} h^{\rho_s} \ \wedge \ C_z = g^{\zeta} h^{\rho_z} \ \wedge \\ & C = (C_v)^{\epsilon} h^{\rho}\} \end{aligned}$$

3. Show that C_z is a commitment to the product of the values committed to by C_w and C_e:

$$\begin{aligned} PK\{(\zeta, \omega, \epsilon, \rho_z, \rho_w, \rho_e, \rho') \ : \ & C_z = g^{\zeta} h^{\rho_z} \ \wedge \ C_w = g^{\omega} h^{\rho_w} \ \wedge \\ & C_e = g^{\epsilon} h^{\rho_e} \ \wedge \ C_z = (C_w)^{\epsilon} h^{\rho'}\} \end{aligned}$$

4. Show that C_x is a commitment to an integer of length ℓ_m, C_s is a commitment to an integer of length ℓ_s, and C_e is a commitment to an integer in the interval $(2^{\ell_e - 1}, 2^{\ell_e})$.

Figure 4-5: Proof of Knowledge of a Signature

Chapter 5

Dynamic Accumulators and Revocation

5.1 Introduction

Suppose a set of users is granted access to a resource. This set changes over time: some users are added, and for some, the access to the resource is revoked. When a user is trying to access the resource, some verifier must check that the user is in this set. The immediate solution is to have the verifier look up the user in some database to make sure that the user is still allowed access to the resource in question. This solution is expensive in terms of communication if the database is not local to the verifier. Another approach is of certificate revocation chains, where every day eligible users get a fresh certificate of eligibility. This is somewhat better because the communication burden is now shifted from the verifier to the user, but still suffers the drawback of high communication costs, as well as the computation costs needed to reissue certificates. Moreover, it disallows revocation at arbitrary time as need arises. A satisfactory solution to this problem has been an interesting question for some time, especially in a situation where the users in the system are anonymous.

Accumulators were introduced by Benaloh and de Mare [BdM94] as a way to combine a set of values into one short accumulator, such that there is a short witness that a given value was incorporated into the accumulator. At the same time, it is infeasible to find a witness for a value that was not accumulated. Extending the ideas due to Benaloh and de Mare [BdM94], Barić and Pfitzmann [BP97] give an efficient construction of so-called collision-resistant accumulators, based on the strong RSA assumption.

We propose a variant of the cited construction with the additional advantage that, using additional trapdoor information, the work of deleting a value from an accumulator can be made independent of the number of accumulated values, at unit cost. Better still, once the accumulator is updated, updating the witness that a given value is in the accumulator (provided that this value has not been revoked, of course!) can be done without the trapdoor information at unit cost. Accumulators with these properties are called *dynamic*. Dynamic accumulators are attractive for the application of granting and revoking privileges.

In the anonymous access setting, where a user can prove eligibility without revealing his identity, revocation appeared impossible to achieve, because if a verifier can tell whether a user is eligible or ineligible, he seems to gain some information about the user's identity. However, it turns out that this intuition was wrong! Indeed, using accumulators in combination with zero-knowledge proofs allows one to prove that a committed value is in the accumulator. We show that this can be done efficiently (i.e., *not* by reducing to an NP-complete problem and then using the fact that $NP \subseteq ZK$ [GMW87b] and *not* by using cut-and-choose for the Barić and Pfitzmann's [BP97] construction).

From the above, we obtain an efficient mechanism for revoking group membership for the Ateniese et al. identity escrow/group signature scheme [ACJT00] (the most efficient secure identity escrow/group signature scheme known to date) and a credential revocation mechanism for our credential system. The construction can be applied to other such schemes as well. The idea is to incorporate the public key for an accumulator scheme into the group manager's (resp., organization's) public key, and the secret trapdoor of the accumulator scheme into the corresponding secret key. Each time a user joins the group (resp., obtains a credential), the group manager (resp., organization) gives her a membership certificate (resp., credential certificate). An integral part of this certificate is a prime number e. This will be the value added to the accumulator when the user is added, and deleted from the accumulator if the user's privileges have to be revoked. This provably secure mechanism does not add any significant communication or computation overhead to the underlying schemes (at most a factor of 2). We note that both our dynamic accumulator scheme and the ACJT identity escrow/group signature scheme rely on the strong RSA assumption.

5.1.1 Related Work

For the class of group signature schemes [CP95, Cam97] where the group's public key contains a list of the public keys of all the group members, excluding a member is straightforward: the group manager only needs to remove the affected member's key from the list. These schemes, however, have the drawback that the complexity of proving and verifying membership is linear in the number of current members and therefore becomes inefficient for large groups. This drawback is overcome by schemes where the size of the group's public key as well as the complexity of proving and verifying membership is independent of the number of members [CS97, KP98, CM99b, ACJT00]. The idea underlying these schemes is that the group public key contains the group manager's public key of a suitable signature scheme. To become a group member, a user chooses a membership public key which the group manager signs. Thus, to prove membership, a user has to prove possession of membership public key, of the corresponding secret key and of a group manager's signature on a membership public key.

The problem of excluding group members within such a framework without incurring big costs has been considered, but until now no solution was satisfactory. One approach is to change the group's public key and reissue all the membership certificates (cf. [AT01]). Clearly, this puts quite a burden on the group manager, especially for large groups. Another approach is to incorporate a list of revoked certificates and their corresponding membership keys into the group's public key [BS01]. In this solution, when proving membership, a user has to prove that his or her membership public key does not appear on the list. Hence, the size of the public key as well as the complexity of proving and verifying signatures are linear in the number of excluded members. In particular, this means that the size of a group signature grows with the number of excluded members.

Song [Son01] presents an alternative approach in conjunction with a construction that yields forward secure group signature schemes based on the ACJT group signature scheme [ACJT00]. While here the size of a group signature is independent of the number of excluded members, the verification task remains computationally intensive, and is linear in the number of excluded group members. Moreover, her approach does not work for ordinary group signature schemes as it relies heavily on the different time periods peculiar to forward secure signatures. Ateniese and Tsudik [AT01] adapt this approach to the ACJT group signature/identity escrow scheme. Their solution retains the property that the verification task is linear in the number of excluded group members. Moreover, it uses so-called double discrete logarithms which results in the complexity of proving/signing and verifying to be rather high compared to underlying scheme (about a factor of 90 for reasonable security parameters).

Finally, we point out that the proposal by Kim et al. [KLL01] is broken, i.e., excluded group members can still prove membership (after the group manager changed the group's key, excluded members can update their membership information in the very same way as non-excluded members).

Thus, until now, all schemes have a linear dependency either on the number of current members, or on the total number of deleted members. As we have noted above, this linear dependency comes in three flavors: (1) the burden being on the group manager to re-issue certificates in every time period; (2) the burden being on the group member to prove that his certificate is different from any of those that have been revoked and on the verifier to check this; or (3) the burden being on the verifier to perform a computational test on the message received from the user for each item in the list of revoked certificates.

In contrast, in our solution no operation is linearly dependent on the number of current or total deleted members. Its only overhead over a scheme without revocation is the following: We require some public archive that stores information on added and deleted users. Then, the public key (whose size depends only on the security parameter) needs to be updated each time a user is added or deleted. Each user must read the public key from time to time (e.g., prior to proving his membership), and if the public key has changed since the last time he looked, he must read the news in the public archive and then perform a local computation. The amount of data to read and the local computation are linear in the number of changes that have taken place in the meantime, but *not* in the total number of changes. The additional burden on the verifier is simply that he should look at the public key frequently (which seems unavoidable); the verifier need not read the archive.

5.2 Definition of Dynamic Accumulators

Definition 5.2.1. *A secure accumulator for a family of inputs $\{\mathbb{X}_k\}$ is a family of families of functions $\mathcal{G} = \{\mathbb{F}_k\}$ with the following properties:*

Efficient generation: *There is an efficient probabilistic algorithm G that on input 1^k produces a random element f of $\mathbb{F}_k$. Moreover, along with f, G also outputs some auxiliary information about f, denoted aux_f.*

Efficient evaluation: *$f \in \mathbb{F}_k$ is a polynomial-size circuit that, on input $(u, x) \in \mathcal{U}_f \times \mathbb{X}_k$, outputs a value $v \in \mathcal{U}_f$, where $\mathcal{U}_f$ is an efficiently samplable input domain for the function f; and $\mathbb{X}_k$ is the intended input domain whose elements are to be accumulated.*

Quasi-commutative: *For all k, for all $f \in \mathbb{F}_k$, for all $u \in \mathcal{U}_f$, for all $x_1, x_2 \in \mathbb{X}_k$, $f(f(u, x_1), x_2) = f(f(u, x_2), x_1)$. If $X = \{x_1, \ldots, x_m\}$ is a list of elements of $\mathbb{X}_k$, possibly with repetitions, then by $f(u, X)$ we denote $f(f(\ldots(u, x_1), \ldots), x_m)$.*

Witnesses: *Let $v \in \mathcal{U}_f$ and $x \in \mathbb{X}_k$. A value $w \in \mathcal{U}_f$ is called a* witness *for x in v under f if $v = f(w, x)$.*

Security: *Let $\mathcal{U}'_f \times \mathbb{X}'_k$ denote the domains for which the computational procedure for function $f \in \mathbb{F}_k$ is defined (thus $\mathcal{U}_f \subseteq \mathcal{U}'_f$, $\mathbb{X}_k \subseteq \mathbb{X}'_k$). For all probabilistic polynomial-time adversaries $\mathcal{A}_k$, there exists a negligible function $\nu(k)$ such that*

$$\Pr[f \leftarrow G(1^k); u \leftarrow \mathcal{U}_f; (x, w, X) \leftarrow \mathcal{A}_k(f, \mathcal{U}_f, u) : \\ X \subset \mathbb{X}_k; w \in \mathcal{U}'_f; x \in \mathbb{X}'_k; x \notin X; f(w, x) = f(u, X)] = \nu(k)$$

Note that only the legitimate accumulated values, $(x_1, \ldots, x_m)$, must belong to $\mathbb{X}_k$; the forged value x can belong to a possibly larger set $\mathbb{X}'_k$.

(This definition is essentially the one of Barić and Pfitzmann, with the difference that they do not require that the accumulator be quasi-commutative; as a consequence they need to introduce two further algorithms, one for generation and one for verification of a witness value.)

The above definition is seemingly tailored for a static use of the accumulator. In this paper, however, we are interested in a dynamic use where there is a manager controlling the accumulator, and several users. First, let us show that dynamic addition of a value is done at unit cost in this setting.

Lemma 5.2.1. *Let $f \in \mathbb{F}_k$. Let $v = f(u, X)$ be the accumulator so far. Let $v' = f(v, x') = f(u, X')$ be the value of the accumulator when x' is added to the accumulated set, $X' = X \cup \{x'\}$. Let $x \in X$ and w be the witness for x in v. The computation of w' which is the witness for x in v', is independent on the size of X.*

Proof. w' is computed as follows: $w' = f(w, x')$. Let us show correctness using the quasi-commutative property: $f(w', x) = f(f(w, x'), x) = f(f(w, x), x') = f(v, x') = v'$. □

We must also be able to handle dynamic deletions of a value from the accumulator. It is clear how to do that using computations that are linear in the size of the accumulated set X. Here, we restrict the definition so as to make the complexity of this operation independent of the size of X:

Definition 5.2.2. *A secure accumulator is* dynamic *if it has the following property:*

Efficient deletion *there exist efficient algorithms D, W such that, if $v = f(u, X)$, $x, x' \in X$, and $f(w, x) = v$, then*

1. *$D(aux_f, v, x') = v'$ such that $v' = f(u, X \setminus \{x'\})$;*
2. *$W(f, v, v', x, x') = w'$ such that $f(w', x) = v'$.*

Now, we show that a dynamic accumulator is secure against an adaptive adversary, in the following scenario: An accumulator manager sets up the function f and the value u and hides the trapdoor information aux_f. The adversary adaptively modifies the set X. When a value is added to it, the manager updates the accumulator value accordingly. When a value $x \in X$ is deleted, the manager runs algorithm D and publishes the result. In the end, the adversary attempts to produce a witness that $x' \notin X$ is in the current accumulator v.

Theorem 5.2.2. *Let $\mathcal{G}$ be a dynamic accumulator algorithm. Let M be an interactive Turing machine set up as follows:*

Input: *(f, aux_f, u), where $f \in \mathbb{F}_k$ and $u \in \mathcal{U}_f$.*

Initialization: *$X := \emptyset$, $v := u$.*

Response to messages:

("ADD", x) *In response to this message, set $v := f(v, x)$, and $X := X \cup \{x\}$.*

("DELETE", x) *In response to this message, check that $x \in X$, and if so, set $v := D(aux_f, v, x)$, and $X := X \setminus \{x\}$.*

Supply the current value of v to the adversary.

Let $\mathcal{U}'_f \times \mathbb{X}'_k$ denote the domains for which the computational procedure for function $f \in \mathbb{F}_k$ is defined. For all probabilistic polynomial-time adversaries $\mathcal{A}_k$, there exists a negligible function $\nu(k)$ such that

$$\Pr[((f, aux_f) \leftarrow G(1^k); u \leftarrow \mathcal{U}_f; (x, w) \leftarrow \mathcal{A}_k(f, \mathcal{U}_f, u) \leftrightarrow M(f, aux_f, u) \rightarrow (X, v) : \\ w \in \mathcal{U}'_f; x \in \mathbb{X}'_k; x \notin X; f(w, x) = f(u, X)] = \nu(k)$$

Proof. Let us exhibit a reduction from the adversary that violates the theorem to the adversary that breaks the security property of a secure accumulator. The reduction will proceed in the following (straightforward) manner: On input $(f, \mathcal{U}_f, u)$, feed these values to the adversary. To respond to an ("ADD", x) query, simply update X and compute $v = f(u, X)$. To respond to a ("DELETE",x) query, compute $v = f(u, X \setminus \{x\})$, and then update X. The success of the adversary directly corresponds to the success of our reduction. □

Finally, in the application we have in mind we require that the accumulator allows for an efficient proof that a secret value given by some commitment is contained in a given accumulator value. That is, we require that the accumulator be *efficiently provable* with respect to some commitment scheme (*Commit*).

Zero-knowledge proof of knowledge of an accumulated value There exists an efficient zero-knowledge proof of knowledge system where the common inputs are c (where $c = \mathit{Commit}(x, r)$ with a r being a randomly chosen string), the accumulating function f and $v \in \mathcal{U}_f$, and the prover's inputs are $(r, x \in \mathbb{X}_k, u \in \mathcal{U}_f)$ for proving knowledge of x, w, r such that $c = \mathit{Commit}(x, r)$ and $v = f(w, x)$.

If by "efficient" we mean "polynomial-time," then any accumulator satisfies this property. However we consider a proof system efficient if it compares well with, for example, a proof of knowledge of a discrete logarithm.

5.3 A Candidate Construction

The construction due to Barić and Pfitzmann [BP97] is the basis for our construction below. The differences from the cited construction are that (1) the domain of the accumulated values consists of prime numbers only; (2) we give a method for deleting values from the accumulator, i.e., we construct a *dynamic* accumulator; (3) we give efficient algorithms for deleting a user and updating a witness; and (4) we provide an efficient zero-knowledge proof of membership knowledge.

- $\mathbb{F}_k$ is the family of functions that correspond to exponentiating modulo safe-prime products drawn from the integers of length k. Choosing $f \in \mathbb{F}_k$ amounts to choosing a random modulus $n = pq$ of length k, where $p = 2p'+1$, $q = 2q'+1$, and p,p',q,q' are all prime. We will denote f corresponding to modulus n and domain $\mathbb{X}_{A,B}$ by $f_{n,A,B}$. We denote $f_{n,A,B}$ by f_n and by f when it does not cause confusion.
- $\mathbb{X}_{A,B}$ is the set $\{e \in \texttt{primes} : e \neq p', q' \ \wedge \ A \leq e \leq B\}$, where A and B can be chosen with arbitrary polynomial dependence on the security parameter k, as long as $2 < A$ and $B < A^2$. $\mathbb{X}'_{A,B}$ is (any subset of) of the set of integers from $[2, A^2 - 1]$ such that $\mathbb{X}_{A,B} \subseteq \mathbb{X}'_{A,B}$.
- For $f = f_n$, the auxiliary information aux_f is the factorization of n.

- For $f = f_n$, $\mathcal{U}_f = \{u \in QR_n : u \neq 1\}$ and $\mathcal{U}'_f = \mathbb{Z}_n^*$.
- For $f = f_n$, $f(u,x) = u^x \bmod n$. Note that $f(f(u,x_1),x_2) = f(u,\{x_1,x_2\}) = u^{x_1x_2} \bmod n$
- Update of the accumulator value. As mentioned earlier, adding a value $\tilde{x}$ to the accumulator value v can be done as $v' = f(v,\tilde{x}) = v^{\tilde{x}} \bmod n$. Deleting a value $\tilde{x}$ from the accumulator is as follows. $D((p,q),v,\tilde{x}) = v^{\tilde{x}^{-1} \bmod (p-1)(q-1)} \bmod n$.
- Update of witness: As mentioned, updating the witness u after $\tilde{x}$ has been added can be done by $u' = f(u,\tilde{x}) = u^{\tilde{x}}$. In case, $\tilde{x} \neq x \in \mathbb{X}_k$ has be deleted from the accumulator, the witness u can be updated as follows. By the extended GCD algorithm, one can compute the integers a,b such that $ax + b\tilde{x} = 1$ and then $u' = W(u,x,\tilde{x},v,v') = u^b v'^a$. Let us verify that $f(u',x) = u'^x \bmod n = v'$:

$$\begin{aligned}
(u^b v'^a)^x &= & (5.1)\\
((u^b v'^a)^{x\tilde{x}})^{1/\tilde{x}} &= & (5.2)\\
((u^x)^{b\tilde{x}}(v'^{\tilde{x}})^{ax})^{1/\tilde{x}} &= & (5.3)\\
(v^{b\tilde{x}}v^{ax})^{1/\tilde{x}} &= v^{1/\tilde{x}} = v' & (5.4)
\end{aligned}$$

Equation (5.2) is correct because $\tilde{x}$ is relatively prime to $\varphi(n)$.

We note that adding or deleting several values at once can be done simply by letting x' be the product of the added or deleted values. This also holds with respect to updating the witness. More precisely, let π_a be the product the x's to add to and π_d be the ones to delete from the accumulator value v. Then, the new accumulator value $v' := v^{\pi_a \pi_d^{-1} \bmod (p-1)(q-1)} \bmod n$. If u was the witness that x was contained in v and x was not removed from the accumulator, i.e., $x \nmid \pi_d$, then $u' u^{a\pi_a} v'^b \bmod n$ is a witness that x is contained in v', where a and b satisfy $ax + b\pi_d = 1$ and are computed using the extended GCD algorithm.

Theorem 5.3.1. *Under the strong RSA assumption, the above construction is a secure dynamic accumulator.*

Proof. Everything besides security is immediate. By Theorem 5.2.2, it is sufficient to show that the construction satisfies security as defined in Definition 5.2.1. Our proof is similar to the one given by Barić-Pfitzmann for their construction (the difference being that we do not require x' to be prime). The proof by Barić-Pfitzmann is actually the same as one given by Shamir [Sha83].

Note that the definition of security requires that for any domain $\mathbb{X}'$ for which the accumulator function f is well-defined, it should be impossible to compute a value u' such that $u = f(u',x)$; therefore we consider the set of possible forgery to be $\mathbb{X}'_{A,B}$ rather than $\mathbb{X}_{A,B}$.

Suppose we are given an adversary $\mathcal{A}$ that, on input n and $u \in_R QR_n$, outputs m primes $x_1, \ldots, x_m \in \mathbb{X}_{A,B}$ and $u' \in \mathbb{Z}_n^*$, $x' \in \mathbb{X}'_{A,B}$ such that $(u')^{x'} = u^{\prod x_i}$ while $x' \neq x_i$ for all i. Let us use $\mathcal{A}$ to break the strong RSA assumption.

Suppose $n = pq$ that is a product of two safe primes, $p = 2p' + 1$ and $q = 2q' + 1$, is given. Suppose the value $u \in QR_n$ is given as well. To break the strong RSA assumption, we must output a value $e > 1$, y such that $y^e = u$.

We shall proceed as follows: Give (n, u) to the adversary. Suppose the adversary comes up with a forgery $(u', x', (x_1, \dots, x_m))$, where $x \neq x_i$ for any i. Let $x = \prod_{i=1}^{m} x_i$. Thus we have $u'^{x'} = u^x$.

Claim. *Let $d = \gcd(x', x)$. Then either $d = 1$ or $d = x_j$ for some $1 \leq j \leq m$.*

Proof of claim: Suppose $d|x$ and $d \neq 1$. Then, as $x_1, \dots, x_m$ are primes, it follows that d is the product of a subset of primes. Suppose for some x_i and x_j we have $x_i x_j | d$. Then $x_i x_j | x'$. But this is a contradiction as $x_i x_j > x'$ must hold due to the definitions of $X_{A,B}$ and $X'_{A,B}$: Because $x' \in X'_{A,B}$ we have $x' < A^2$. For any $x_i, x_j \in X_{A,B}$, $x_i x_j \geq A^2 > x'$, as $x_1, x_2 \geq A$.

From the claim, it follows that $d = \gcd(x', x) < x$: either (1) $d = 1$, or (2) $d = x_j$, while $x_j \neq x'$, so $d = x_j < x'$. By Lemma 4.1.10, the values z and $w > 1$ can be computed efficiently, such that $z^w = u \bmod n$. This contradicts the strong RSA assumption. □

5.3.1 Efficient Proof That a Committed Value Was Accumulated

Here we show that the accumulator exhibited above is efficiently provable with respect to the commitment scheme described in Section 4.5.1. Suppose that the public key of the commitment scheme is (n_C, g_C, h_C), and that it was picked in an initial trusted setup phase.

We also need to extend the accumulator public key (n, v) by random elements $(g, h) \in QR_n$ such that $\log_g h$ is not known to the prover.

To prove that a given commitment C_e and a given accumulator v contain the same value e, the following protocol is carried out.

Lemma 5.3.2. *The protocol in Figure 5-1 is a zero-knowledge proof of knowledge of the values (e, u, r) such that $u^e = v \bmod n$ and $C = g_C^e h_C^r$.*

Proof. The completeness property is clear. The zero-knowledge property follows because the simulator can form the commitments at random and fake the proofs. For the extraction, note that we extract values $(\epsilon, \omega, \rho, \alpha)$ such that (1) $C = vg^\alpha h^\rho = C_u^\epsilon h^\rho$, (2) $\alpha = \epsilon\omega$, and (3) $\epsilon \in \mathbb{X}$. From (1), we also get (4) $C_u^\epsilon = vg^\alpha$. Let $u = C_u / g^\omega$. Note that $u^\epsilon = (C_u/g^\omega)^\epsilon = C_u^\epsilon / g^{\epsilon\omega} = vg^\alpha / g^\alpha = v$, and so we output the value $\epsilon \in \mathbb{X}'$ and proof of membership u. □

5.4 Application to the Credential System

Here, we show how to use the dynamic accumulator presented in Section 5.3 in order to revoke credentials in the credential system presented in Chapter 3 where the underlying signature scheme is as in Chapter 4.

Common inputs: Public key (n, v, g, h) of the accumulator, range $[A, B]$
Commitment public key (n_C, g_C, h_C), commitment C_e
User's input: The values (u, e, r_C) such that $u^e \equiv v \bmod n$ and $C_e = g_C^e h_C^{r_C} \bmod n_C$.

$U \longleftrightarrow V$ Choose, uniformly at random of length ℓ_n, values w, r_w, r and compute the following: $C_u = ug^w \bmod n$, $C_w = g^w h^{r_w} \bmod n$, $C = (C_u)^e h^r \bmod n$.

It is important to note that $C = (C_u)^e h^r = u^e g^{we} h^r (\bmod n)$.

Send the values (C_u, C_w, C) to the verifier and carry out the following zero-knowledge proofs of knowledge:

1. Show that C is a commitment in bases (C_u, h) to the value committed to in the commitment C_e, and that C/v has the right representation in (g, h), using the protocol described in Section 4.5.4.

$$PK\{(\epsilon, \rho, \rho_e, \alpha) : C = C_u^\epsilon h^\rho \bmod n \quad \wedge \quad C_e = g_C^\epsilon h_C^{\rho_e} \bmod n_C \quad \wedge \quad C/v = g^\alpha h^\rho \bmod n\}$$

2. Show that C/v is also a commitment in bases (g, h), to the product of the values committed to by C and C_w, as done in Section 4.6.2.
3. Show that the value committed to in commitment C, in bases (C_u, h), is in the range $\mathbb{X}'_{A,B}$, as described in Section 4.5.5.

Figure 5-1: Proof That a Committed Value Is in the Accumulator

In general, each organization will have, in addition to its usual credential public key, a public key (f, u) of a dynamic accumulator. Every time a credential σ is issued under the credential public key, the organization receives the value $x(\sigma)$ where $x(\cdot)$ is an efficiently computable function such that $x(\sigma) \in \mathbb{X}$. The value $x(\sigma)$ is added to the accumulator when the credential σ is granted, and deleted from the accumulator when the credential is revoked. Proof of knowledge of a credential involves proving knowledge of (SK, σ, w) such that σ is a signature on SK, and $f(x(\sigma), w) = u$, where u is the current value of the accumulator. What is essential here is that no two credentials have the same value $x(\sigma)$.

For the signature scheme described in Chapter 4, where signature σ on message m is a tuple (s, e, v) such that $a^m b^s c = v^e \bmod n$, let $x(\sigma) = e$. No two credentials will have the same value e.

Chapter 6

Unique Signature Scheme

Since the seminal work of Goldwasser, Micali and Rivest [GMR88] on defining signature schemes, it has become clear that the first requirement from a signature scheme is that it should satisfy the GMR definition of security. However, to be most useful, two additional properties of signature schemes are desirable: (1) that the scheme be secure in the plain model, i.e., without a random oracle or common parameters; and (2) that the scheme be stateless, i.e., not require the signer to update the secret key after each invocation.

The only signature schemes satisfying both of these additional properties are the Strong-RSA-based schemes of Gennaro et al. [GHR99] and Cramer and Shoup [CS99], and the scheme implied by the verifiable random function due to Micali et al. [MRV99], based on RSA. An open question was to come up with a signature scheme satisfying the two additional properties, such that it would be secure under a different type of assumption. Here, we give such a signature scheme, based on a generalization of the Diffie-Hellman assumption for groups where decisional Diffie-Hellman is easy.

Unique signatures. The signature scheme we propose is a unique signature scheme. Unique signature schemes are GMR-secure signature schemes where the signature is a hard-to-compute function of the public key and the message. They were introduced by Goldwasser and Ostrovsky [GO92][1].

In the random-oracle model, a realization of unique signatures is well-known. For example, some RSA signatures are unique: $\sigma_{n,e,H}(m) = (H(m))^{1/e} \bmod n$ is a function of (n, e, H, m) so long as e is a prime number greater than n (this is because for a prime e, $e > n$ implies that e is relatively prime to $\phi(n)$). Goldwasser and Ostrovsky give a solution in the common-random-string model. However, in the standard model, the only known construction of unique signatures was the one due to Micali, Rabin, Vadhan [MRV99].

Goldwasser and Ostrovsky have also shown how to construct unique signatures in the common random string model using non-interactive zero-knowledge proofs. They

[1]They call it an *invariant* signature. In fact, invariant signatures are not exactly unique signatures as defined here. But the concepts are sufficiently related not to worry about the subtleties of the definitions.

also gave the converse construction.

Goldwasser and Ostrovsky's construction of a non-interactive zero-knowledge proof (NIZK) for any language in NP for polynomial-time provers[2], is the alternative to the construction due to Feige, Lapidot and Shamir [FLS99]. Feige, Lapidot and Shamir give a construction based on trapdoor permutations. In contrast, Goldwasser and Ostrovsky show that the trapdoor property is not necessarily crucial, because unique signatures are sufficient as well. Until the results in this thesis, there was no application to the Goldwasser and Ostrovsky construction, since the only known constructions of unique signatures, even in the common-random-string model, were either based on RSA (which is conjectured to be a family of trapdoor permutations), or on non-interactive zero knowledge. Our result therefore implies that trapdoor permutations might not be necessary for NIZK with efficient provers. Specifically, our result, combined with the result of Goldwasser and Ostrovsky, shows that our Many-DH Assumption stated in Section 6.2 implies such NIZK proofs. At the same time, this assumption is not known to imply existence of trapdoor permutations.

Verifiable random functions. Another reason why unique signatures are valuable is that they are closely related to verifiable random functions. Verifiable random functions (VRFs) were introduced by Micali, Rabin, and Vadhan [MRV99]. They are similar to pseudorandom functions [GGM86], except that they are also verifiable. That is to say, associated with a secret seed SK, there is a public key PK and a function $F_{PK}(\cdot) : \{0,1\}^k \mapsto \{0,1\}^u$ such that (1) $y = F_{PK}(x)$ is efficiently computable given the corresponding SK; (2) a proof $\pi_{PK}(x)$ that this value y corresponds to the public key PK is also efficiently computable given SK; (3) based purely on PK and oracle calls to $F_{PK}(\cdot)$ and the corresponding proof oracle, no adversary can distinguish the value $F_{PK}(x)$ from a random value without explicitly querying for the value x.

VRFs [MRV99] are useful for protocol design. They can be viewed as a commitment to an exponential number of random-looking bits, which can be of use in protocols. For example, using verifiable random functions, one can reduce the number of rounds for resettable zero knowledge proofs to 3 in the bare model [MR01]. Another example application, due to Micali and Rivest [MR02], is a non-interactive lottery system used in micropayments. Here, the lottery organizer holds a public key PK of a VRF. A participant creates his lottery ticket t himself and sends it to the organizer. The organizer computes the value $y = F_{PK}(t)$ on the lottery ticket, and the corresponding proof $\pi = \pi_{PK}(t)$. The value y determines whether the user wins, while the proof π guarantees that the organizer cannot cheat. Since a VRF is hard to predict, the user has no idea how to bias the lottery in his favor.

These objects are not well-studied; in fact, only one construction, based on the RSA assumption, was previously known [MRV99]. Micali, Rabin and Vadhan showed that, for the purposes of constructing a VRF, it is sufficient to construct a unique signature scheme. More precisely, from a unique signature scheme with small mes-

[2]In fact, their construction is using *invariant* signatures; however, using later techniques of Micali, Rabin, and Vadhan, their construction can use unique signatures.

sage space, they constructed a VRF with an arbitrary input size that tolerates a polynomial-time adversary. They then gave an RSA-based unique signature scheme for a small message space.

Constructing VRFs from pseudorandom functions (PRFs) is a good problem that we do not know how to solve in the standard model. This is because by its definition, the output of a PRF should be indistinguishable from random. However, if we extend the model to allow interaction, it is possible to commit to the secret seed of a PRF and let that serve as a public key, and then prove that a given output corresponds to the committed seed using a zero-knowledge proof. This solution is unattractive because of the expense of communication rounds. In the so-called *common random string* model where non-interactive zero-knowledge proofs are possible, a construction is possible using commitments and non-interactive zero knowledge [BFM88, BDMP91, FLS99]. However, this is unattractive as well because this model is unusual and non-interactive ZK is expensive. Thus, construction of verifiable random functions from general assumptions remains an interesting open problem.

DH–DDH separation. Recently, Joux and Nguyen [JN01] demonstrated that one can encounter groups in which decisional Diffie-Hellman is easy, and yet computational Diffie-Hellman seems hard. This is an elegant result that, among other things, sheds light on how reasonable it is to assume decisional Diffie-Hellman. (The statement of these assumptions is given in Section 6.1.)

Joux [Jou00] proposed using the DH-DDH separation to a good end, by exhibiting a one-round key exchange protocol for three parties. Subsequently, insight into such groups has proved relevant for the recent construction of identity-based encryption due to Boneh and Franklin [BF01] which resolved a long standing open problem [Sha85]. Other interesting consequences of the study of these groups are a construction of a short signature scheme in the random-oracle model [BLS01] and of a simple credential system [Ver01].

Our results. We give a simple construction of a unique signature scheme based on groups where DH is conjectured hard and DDH is easy. Ours is a tree-like construction. The message space consists of codewords of an error-correcting code that can correct a constant fraction of errors. For n-bit codewords, the depth of the tree is n. The root of the tree is labelled with g, a generator of a group where DH is hard and DDH is easy. The 2^n leaves of the tree correspond all the possible n-bit strings. The 2^i nodes of depth i, $1 \leq i < n$ correspond to all the possible i-bit prefixes of an n-bit string.

A pair of group elements $(A_{i,0} = g^{a_{i,0}}, A_{i,1} = g^{a_{i,1}})$ is associated with each depth i of the tree as part of the public key. The label of a node of depth i is derived from the label of its parent by raising the parent's label to the exponent $a_{i,0}$ if this node is its parent's left child, or $a_{i,1}$ if it is the parent's right child.

Computing the signature on each codeword m amounts to computing the labels of the nodes on the path from the root of the tree all the way down to the leaf corresponding to m. The signature is this sequence of labels.

At first glance, this may seem very similar to the Naor-Reingold [NR97] pseudorandom function. Indeed, their construction was an inspiration to mine. However the proof is not immediate. The major difference from the cited result is that here we must give a proof that the function was evaluated correctly. That makes it harder to prove security. For example, proof by a hybrid argument, as done by Naor and Reingold [NR97] is ruled out immediately: there is no way we can answer some of the adversary's queries with truly random bits, since for such bits there will be no proof.

Our proof of security for the unique signature relies on a generalization of the Diffie-Hellman assumption. We call it the "Many-DH" assumption. However, for directly converting this simple US to VRF, we need to make a very strong assumption; however the resulting VRF is very simple. We also suggest a more involved but also more secure construction of a VRF based on the Many-DH assumption. This last construction is closely related to that due to Micali et al. [MRV99].

Outline of the rest of this chapter. In Section 6.1 we give definitions of verifiable random functions and unique signatures. In Section 6.2 we state our complexity assumptions for unique signatures. In Section 6.3 we give our unique signature scheme and prove it secure in Section 6.4. We then provide a simple construction for a VRF under a somewhat stronger assumption, in Section 6.5. We conclude in Section 6.6 with a construction of a VRF based on the weaker assumption alone, but whose complexity (both in terms of computation and in terms of conceptual simplicity) is the same as that of Micali et al. [MRV99].

6.1 Definitions

6.1.1 Unique Signatures

Unique signatures are secure signatures where the signature is a function as opposed to a distribution.

Definition 6.1.1. *A function family $\sigma_{(\cdot)}(\cdot) : \{0,1\}^k \mapsto \{0,1\}^{\ell(k)}$ is a unique signature scheme (US) if there exists probabilistic algorithm G, efficient deterministic algorithm* `Sign`*, and deterministic algorithm* `Verify` *such that $G(1^k)$ generates the key pair PK, SK,* `Sign`*(SK, x) computes the value $\sigma = \sigma_{PK}(x)$ and* `Verify`*(PK, x, σ) verifies that $\sigma = \sigma_{PK}(x)$.*

1. *(Validity) If $\sigma =$ `Sign`(SK, x), and there exists a random string and a value k such that (PK, SK) is obtained by running $G(1^k)$ on this random string, then* `Verify`*$(PK, x, \sigma) = 1$.*

2. *(Uniqueness of $\sigma_{PK}(m)$) There do not exist values $(PK, m, \sigma_1, \sigma_2)$ such that $\sigma_1 \neq \sigma_2$ and* `Verify`*$(PK, m, \sigma_1) =$* `Verify`*$(PK, m, \sigma_2) = 1$.*

3. *(Security) For all families of probabilistic polynomial-time oracle Turing machines $\{A_k^{(\cdot)}\}$, there exists a negligible function $\nu(k)$ such that*

$$\begin{array}{ll}\Pr[(PK, SK) \leftarrow G(1^k); & \\ (Q, x, \sigma) \leftarrow A_k^{\texttt{Sign}(SK,\cdot)}(1^k); & : \ \texttt{Verify}(PK, x, \sigma) = 1 \wedge (x, \sigma) \notin Q] \leq \nu(k)\end{array}$$

On a relaxed definition. Goldwasser and Ostrovsky [GO92] give a relaxed definition. In their definition, even though the signature is unique, the verification procedure may require, as an additional input, a proof that the signature is correct. This proof is output by the signing algorithm together with the signature, and it might not be unique. Here we give the stronger definition because we can satisfy it (Goldwasser and Ostrovsky do not, even though they work in the common random string model). However, note that the relaxed definition is sufficient for constructing VRFs [MRV99].

Unique signatures are stateless. Since a unique signature is a function of the public key and the message, a signature on a given message will be the same whether this was the first message signed by the signer, or the n'th message. As a result, it is easy to see that the signer does not need to remember anything about past transactions, i.e., a unique signature must be stateless.

6.1.2 Verifiable Random Functions

The definition below is due to Micali et al. [MRV99]. We use somewhat different and more compact notation, however.

The intuition of this definition is that a function is a verifiable random function if it is like a pseudorandom function with a public key and proofs.

Definition 6.1.2. *A function family $F_{(\cdot)}(\cdot)\ :\ \{0,1\}^{\ell(k)} \mapsto \{0,1\}^{m(k)}$ is a verifiable random function (VRF) if there exist probabilistic algorithm G, and deterministic algorithms* `Eval`*,* `Prove`*, and* `Verify` *such that: $G(1^k)$ generates the key pair PK, SK;* `Eval`(SK, x) *computes the value $y = F_{PK}(x)$;* `Prove`(SK, x) *computes the proof π that $y = F_{PK}(x)$; and* `Verify`(PK, x, y, π) *verifies that $y = F_{PK}(x)$ using the proof π. More formally:*

1. *(Computability of $F_{PK}(x)$) $F_{PK}(x) =$* `Eval`(SK, x).

2. *(Provability of $F_{PK}(x)$) If $(y, \pi) =$* `Prove`(SK, x)*, then*

$$\texttt{Verify}(PK, x, y, \pi) = 1$$

.

3. *(Uniqueness of $F_{PK}(x)$) There do not exist values $(PK, SK, x, y_1, y_2, \pi_1, \pi_2)$ such that $y_1 \neq y_2$ and* `Verify`$(PK, x, y_1, \pi_1) =$ `Verify`$(PK, x, y_2, \pi_2) = 1$.

4. *(Pseudorandomness of $F_{PK}(x)$) For all families of probabilistic polynomial-time Turing machines $\{A_k^{(\cdot)}, B_k\}$, there exists a negligible function $\nu(k)$ such that*

$$\begin{aligned}
\Pr[(PK, SK) &\leftarrow G(1^k);\\
(Q_A, x, \mathtt{state}) &\leftarrow A_k^{\mathtt{Prove}(SK,\cdot)}(1^k);\\
y_0 &= \mathtt{Eval}(SK, x);\\
y_1 &\leftarrow \{0,1\}^{m(k)};\\
b &\leftarrow \{0,1\};\\
(Q_B, b') &\leftarrow B_k^{\mathtt{Prove}(SK,\cdot)}(\mathtt{state}, y_b) \ : \ b = b' \wedge (x, \mathtt{Prove}(SK, x)) \notin Q_A \cup Q_B]\\
&\leq 1/2 + \nu(k)
\end{aligned}$$

(The purpose of the state *random variable is so that A_k can save some useful information that B_k will then need.)*

In other words, the only way that an adversary could tell $F_{PK}(x)$ from a random value, for x of its own choice, is by querying it directly.

It is easy to see [MRV99] that given a VRF $F_{(PK)} : \{0,1\}^{\ell(k)} \mapsto \{0,1\}$, one can construct a VRF $F'_{(\cdot)} : \{0,1\}^{\ell'(k)} \mapsto \{0,1\}^{m(k)}$, where $\ell'(k) = \ell(k) - \lceil \log m(k) \rceil$, as follows: $F'_{PK}(x_1 \circ \ldots \circ x_{\ell'(k)}) = F_{PK}(x_1 \circ \ldots \circ x_{\ell'(k)} \circ u_0) \circ F_{PK}(x_1 \circ \ldots \circ x_{\ell'(k)} \circ u_1) \circ \ldots \circ F_{PK}(x_1 \circ \ldots \circ x_{\ell'(k)} \circ u_{m(k)})$ where u_i denotes the $\lceil \log m(k) \rceil$-bit representation of the integer i, and "$\circ$" denotes concatenation.

Thus in the sequel we will focus on constructing VRFs with binary outputs.

Unique signatures vs. VRFs. Micali et al. showed how to construct VRFs from unique signatures. The converse, namely construction of a unique signature from a VRF, holds immediately if the proofs in the VRFs are unique. If the proofs are not unique, then no construction satisfying our strong definition of unique signatures in known. However, constructing relaxed unique signatures in the sense of Goldwasser and Ostrovsky (see the end of Section 6.1.1) is immediate.

6.2 Assumptions

Let S be an efficient algorithm that, on input 1^k, generates a group $G = (*, q, g)$ with efficiently computable group operation $*$, of prime order q, with generator g. We require that g is written down in binary using $O(\log q)$ bits, and that every element of the group has a unique binary representation.

We require that the decisional Diffie-Hellman problem be easy in G. More precisely, we require that there is an efficient algorithm D for deciding the following language $L_{DDH}(G)$:

$$L_{DDH}(G) = \{(*, q, g, X, Y, Z) \mid \exists x, y \in \mathbb{Z}_q \text{ such that } X = g^x, Y = g^y, Z = g^{xy}\}$$

One the other hand, we will need to make the following assumption which is somewhat stronger than computational Diffie-Hellman. It is essentially like computational Diffie-Hellman, except that instead of taking two inputs, g^x and g^y, and the challenge is computing g^{xy}, we have a logarithmic number of bases $g^{y_1}, \dots, g^{y_\ell}$, as well as all the products $g^{\prod_{j \in J} y_j}$ for all proper subsets J of the naturals up to and including ℓ, and the challenge is to come up with the value $g^{\prod_{j=1}^{\ell} y_j}$.

Assumption 6.2.1 (Many-DH Assumption). *For all $\ell = O(\log k)$, for all probabilistic polynomial-time families of Turing machines $\{A_k\}$,*

$$\begin{aligned}\Pr[(*, q, g) \leftarrow S(1^k); \{y_i \leftarrow \mathbb{Z}_q \ : \ 1 \le i \le \ell\};\\ \{z_J = \prod_{j \in J} y_j; Z_J = g^{z_J} \ : \ J \subset [\ell]\};\\ Z \leftarrow A_k(*, q, g, \{Z_J \ : \ J \subset [\ell]\}) \ : \ Z = g^{\prod y_i}] \le \nu(k)\end{aligned}$$

where by $[\ell]$ we denote the set of integers from 1 *to* ℓ.

Groups of this flavor, where the decisional Diffie-Hellman problem is easy, and yet generalizations of the computational Diffie-Hellman problem are hard, are said to have "DH-DDH separation." Examples of groups that are conjectured to have DH-DDH separation are elliptic curve groups that allow the Weil pairing [Sil86]. For a more thorough exposition of such groups, we refer the reader, for example, to the recent papers by Joux and Nguyen [JN01], Joux [Jou00] and Boneh and Franklin [BF01]. In the sequel, we will not address the number-theoretic aspects of the subject.

6.3 Construction of a Unique Signature

Suppose the algorithms S, D, as in Section 6.2, are given. Let k be the security parameter. Let the message space consist of strings of length n_0. Our only assumption on the size of the message space is that $n_0 = \omega(\log k)$.

Let $C_{n_0} : \{0,1\}^{n_0} \mapsto \{0,1\}^{n(n_0)}$ be a family of error-correcting codes where $n(n_0)$ is polynomial, and the distance of the code is $cn(n_0)$, where $c > 0$ is a constant. In other words, C is a function such that if $M \neq M'$ are strings of length n_0, then $C(M)$ differs from $C(M')$ in at least $cn(n_0)$ places. For ease of notation, in the sequel we will write n instead of $n(n_0)$. For an overview of error-correcting codes, see the lecture notes of Madhu Sudan [Sud]. Here, we note that since we will not need the decoding operation, only the encoding operation, we can easily achieve $n = O(n_0)$. We will give an example of such a code in Section 6.3.1.

We will now show how to construct algorithms G, `Sign`, `Verify` as specified in Definition 6.1.1.

Algorithm G Run $S(1^k)$ to obtain$G = (*, q, g)$. Choose n pairs of random elements in $\mathbb{Z}_q$: $(a_{1,0}, a_{1,1}), \dots, (a_{n,0}, a_{n,1})$. Let $A_{i,b} = g^{a_{i,b}}$, for $1 \le i \le k$, $b \in \{0,1\}$. Output the following key pair:

$$SK = \begin{array}{|c|c|c|c|}\hline a_{1,0} & a_{2,0} & \cdots & a_{n,0} \\ \hline a_{1,1} & a_{2,1} & \cdots & a_{n,1} \\ \hline \end{array} \qquad PK = \begin{array}{|c|c|c|c|}\hline A_{1,0} & A_{2,0} & \cdots & A_{n,0} \\ \hline A_{1,1} & A_{2,1} & \cdots & A_{n,1} \\ \hline \end{array}$$

Algorithm `Sign` On input a message M of length n_0, compute the encoding of M using code C: $m = C(M)$. To sign the n-bit codeword $m = m_1 \circ \ldots \circ m_n$, output $\sigma_{PK}(m) = (s_1, \ldots, s_n)$, where $s_0 = g$ and $s_i = (s_{i-1})^{a_{i,m_i}}$ for $1 \leq i \leq n$.

Algorithm `Verify` Let $s_{m,0} = g$. Verify that, for all $1 \leq i \leq n$,

$$D(*, q, g, s_{m,i-1}, A_{i,m_i}, s_{m,i}) = ACCEPT$$

Graphically, we view the message space as the leaves of a balanced binary tree of depth n. Each internal node of the tree is assigned a label, as follows: the label of the root is g. The label of a child, denoted l_c is obtained from the label of its parent, denoted l_p, as follows: if the depth of the child is i, and it is the left child, then its label is $l_c = l_p^{a_{i,0}}$, while if it is the right child, its label will be $l_c = l_p^{a_{i,1}}$. The signature on an n-bit message consists of all the labels on the path from the leaf corresponding to this message all the way to the root. To verify correctness of a signature, one uses the algorithm D that solves the DDH for the group over which this is done.

Efficiency. In terms of efficiency, this signature scheme is a factor of $O(n)$ worse than the Cramer-Shoup scheme, in all parameters such as the key and signature lengths and the complexity of relevant operations. This means that it is still rather practical, and yet has two benefits: uniqueness, and security based on a different assumption. In comparison to the unique signature of Micali et al., our construction is preferable as far as efficiency is concerned. This is because our construction is direct, while they first give a unique signature for short messages, then show how to construct a VRF and a unique signature of arbitrary length from that.

Reducing the length of signatures. Boneh and Silverberg [BS02] point out that, if the language $L(*, q, g) = \{g^{y_1}, \ldots, g^{y_n}, g^{\prod_{i=1}^{n} y_i}\}$ is efficiently decidable, then the signature does not need to contain the labels of the intermediate nodes. I.e., to sign a message M, $m = C(M)$, it is sufficient to give $s = g^{\prod_{i=1}^{n} a_{i,m_i}}$. This reduces the length of a signature by a factor of n. However, finding groups where $L(*, q, g)$ is efficiently decidable, and yet the Many-DH Assumption is still reasonable, is an open question [BS02].

On the need for an error-correcting code. The reason that the construction uses an error-correcting code is purely a technical tweak that allows the signature scheme to be provable under the Many-DH assumption. In order to contradict the assumption, we reduce an instance of the Many-DH problem to an instance of the signature scheme. We embed the values g^{y_i} of the Many-DH problem into the public key of the signature scheme. In order for the reduction to go through, we need the forged message to differ from all the previously queried messages on the bits where $\{g^{y_i}\}$ are embedded. To guarantee that, we use the error-correcting code, which ensures that the message on which the adversary computes a forgery differs from any previous message on a substantial fraction of bits. The step of the proof that requires error-correction is Lemma 6.4.4.

6.3.1 An Example of a Suitable Code

Consider a random code constructed as follows: $n = 24n_0$ random strings of length n_0 are chosen, let us denote them by $R^1, \ldots, R^n$, where each $R^i = r_1^i \circ \ldots \circ r_{n_1}^i$. The encoding of message $M = m_1 \circ \ldots \circ m_{n_0}$ is $C(M) = c_1 \circ \ldots \circ c_n$, where the bit $c_i = \bigoplus_{j=1}^{n_0} m_j r_j^i$.

Suppose M_1 and M_2 of length n_0 are fixed. Let $d(C(M_1), C(M_2))$ denote the Hamming distance between the codewords for M_1 and M_2. Let $X = C(M_1) \oplus C(M_2)$, and let X_i denote the i'th bit of X. Since R_i is a random n_0-bit string, and $M_1 \neq M_2$, $X_i = M_1.R_i \oplus M_2.R_i = (M_1 \oplus M_2).R_i$ is a 0-1 random variable such that $\Pr_{R_i}[X_i = 0] = \Pr_{R_i}[X_i = 1] = 1/2 = p$. Since R_i is chosen independently of R_j, $i \neq j$, X_i is independent of X_j. It is easy to see that $E_C[d(C(M_1), C(M_2))] = \sum_{i=1}^{n} E_{R_i}[X_i]$. Therefore, by the Chernoff bound, and using $e > 2$,

$$\begin{aligned}
\Pr_C[d(C(M_1), C(M_2)) > n_0] &\leq \Pr_C[|d(C(M_1), C(M_2))/n - 1/2| > 1/4] \\
&= \Pr_C[|\sum X_i/n - p| > \epsilon] \\
&< 2e^{-\frac{\epsilon^2}{2p(1-p)}n} \\
&= 2e^{-\frac{(1/4)^2}{1/2}n} \\
&= 2e^{-n/8} < 2^{-n/8} < 2^{-3n_0}
\end{aligned}$$

Consequently, by the union bound, since there are less than 2^{2n_0} possible choices for $M_1 \neq M_2$, $\Pr_C[\exists M_1 \neq M_2 \text{ s.t. } d(C(M_1), C(M_2)) \leq n_0] < 2^{-n_0}$.

6.4 Proof of Security for the Unique Signature

In this section, we show how to reduce breaking the Many-DH problem to forging a signature of the construction in Section 6.3.

First, we show the following lemma:

Lemma 6.4.1. *Suppose* $\texttt{Verify}(((*, q, g), \{A_{i,b}\}_{1 \leq i \leq n, b \in \{0,1\}}), m, (s_1, \ldots, s_n)) = 1$. *Then* $s_j = g^{\prod_{i=1}^{j} a_{i,m_i}}$, *where* a_{i,m_i} *denotes the unique value* $\mathbb{Z}_q$ *such that* $g^{a_{i,m_i}} = A_{i,m_i}$.

Proof. We show the lemma by induction on j. For $j = 1$, the verification algorithm will accept only if $s_1 = A_{1,m_i}$.

$PK =$

$g^{a_{1,0}}$	$g^{a_{2,0}}$	$g^{a_{3,0}}$	Y_2	$g^{a_{5,0}}$	$g^{a_{6,0}}$	$g^{a_{7,0}}$	$g^{a_{8,0}}$	$g^{a_{9,0}}$	$g^{a_{10,0}}$
$g^{a_{1,1}}$	$g^{a_{2,1}}$	Y_1	$g^{a_{4,1}}$	$g^{a_{5,1}}$	$g^{a_{6,1}}$	$g^{a_{7,1}}$	$g^{a_{8,1}}$	Y_3	$g^{a_{10,1}}$

Figure 6-1: Toy example of a public key produced by the reduction. In this example, $n = 10$, $\ell = 3$. The reduction randomly picked $J = \{3, 4, 9\}$, $B = 101$.

Let the lemma hold for $j - 1$. The verification algorithm will accept s_j only if $D(*, q, g, s_{j-1}, A_{j,m_j}, s_j) = 1$. But by definition of D, this is the case only if $s_j = g^{\sigma a_{j,m_j}}$, where σ is the unique value in $\mathbb{Z}_q$ such that $g^\sigma = s_{j-1}$. By the induction hypothesis, $\sigma = \prod_{i=1}^{j-1} a_{i,m_i}$. Therefore, $s_j = g^{\sigma a_{j,m_j}} = g^{\prod_{i=1}^{j} a_{i,m_i}}$. $\square$

6.4.1 Description of the Reduction

In the following, "we" refers to the reduction, and the forger for the signature scheme is "our adversary."

Recall that k is the security parameter, and that $n_0 = \omega(\log k)$, $n = O(n_0)$.

Suppose that our adversary's running time is $t(k)$. Recall that c is the distance of the error-correcting code we use. Let $\ell = \frac{\lceil \log t(k) \rceil + 2}{\log(1/(1-.99c))} + 1$. (Note that $\ell = O(\log k)$). Assume G is as in Section 6.2.

Input to the reduction: As in Assumption 6.2.1, we are given a group $G = (*, q, g)$ and ℓ elements $Y_1, Y_2, \ldots, Y_\ell$ of G, where $Y_u = g^{y_u}$. We are also given the values $Z_I = g^{\prod_{u \in I} y_u}$, for $I \subset [\ell]$. The goal is to break Assumption 6.2.1, i.e., compute $g^{\prod_{u=1}^{\ell} y_u}$.

Key generation: Pick a random ℓ-bit string $B = b_1 \circ \ldots \circ b_\ell$ and a random ℓ-bit subset $J \subset [n]$. (For simpler notation, assume that the indices are ordered such that $j_u < j_{u+1}$ for all $j_u \in J$.)

Set up the public key as follows: $A_{j_u,b_u} = Y_u$. To set up $A_{i,b}$ where $i \notin J$ or $i = j_u \in J$ but $b \neq b_u$, choose value $a_{i,b} \leftarrow \mathbb{Z}_q$ and set $A_{i,b} = g^{a_{i,b}}$. Figure 6-1 gives an example of what the reduction does in this step.

Responding to signature queries: We will respond to at most $2t$ signature queries from the adversary, as follows.

Suppose the adversary's query is M where $C(M) = m = m_1 \circ \ldots \circ m_n$. Let $J(m)$ denote the string $m_{j_1} \circ \ldots \circ m_{j_\ell}$.

Check if $J(m) = B$. If so, terminate with "Fail." Otherwise, compute the signature as follows: Let $Z_0 = g$. Let $b' = b'(J)$ be an n-bit string such that $b_{j_u} = b_u$

for all $j_u \in J$. By I_u, we denote the ℓ-bit string which has a 1 in position u, and 0's everywhere else.

Compute $(s_1, \dots, s_n)$ using the following loop:

```
Initialize $c := 0^\ell$
  For $i = 1$ to $n$
    (1) If $i = j_u \in J$ and $m_i = b_u$, then $c := c \oplus I_u$
    (2) $s_i = Z_c^{\prod_{j \notin J, j \leq i} a_{j,m_j} \prod_{j \in J, j \leq i, m_j \neq b'_j} a_{j,m_j}}$
    (3) Increment $i$
  (end of loop)
```

Processing the forgery: Now suppose that the adversary comes back with a forged signature on message M' for which it has not queried R. Let $m' = C(M')$. This forgery is $\sigma_{PK}(m') = \{s_{m',i}\}$. This forgery is *good* if $J(m') = B$. If the forgery is good, we obtain the value $g^{\prod_{u=1}^{\ell} y_u}$ by Lemma 6.4.1, by simply computing $(s_{m',i})^{1/\prod_{i \notin J} a_i^{m'_i}}$.

6.4.2 Analysis of the Reduction

Let $t = t(k)$ be the expected running time of the adversary. For the purposes of the analysis, consider the following algorithms:

- Algorithm 1 runs the signing algorithm and responds to the adversary's queries as the true signing oracle would. If the adversary outputs a valid forgery, this algorithm outputs "Success."

- Algorithm 2 runs the signing algorithm but only responds to $2t$ queries. If the adversary issues more queries, output "Fail." Otherwise, if the adversary outputs a valid forgery, this algorithm outputs "Success."

- Algorithm 3 is the same as Algorithm 2, except in one case. Namely, it chooses a random $J \subset [n]$, $|J| = \ell$, $J = \{j_1, \dots, j_\ell\}$, and outputs "Fail" if it so happens that $J(C(M')) = J(C(M))$ where M' is the adversary's forgery, and M is any previously queried message, and notation $J(m)$ denotes $m_{j_1} \circ m_{j_2} \circ \dots \circ m_{j_\ell}$.

- Algorithm 4 is just like Algorithm 3 except in one case. Namely, it chooses a random ℓ-bit string B and outputs "Fail" whenever the forged message M' is such that $C(M') = m'$ where $J(m') \neq B$.

- Algorithm 5 is just like Algorithm 4 except in one case. Namely, it outputs "Fail" whenever the queried message M is such that $C(M) = m$ where $J(m) = B$.

- Algorithm 6 runs our reduction. It outputs "Success" whenever the reduction succeeds in computing its goal.

By p_i, let us denote the success probability of algorithm i.

Lemma 6.4.2. *The success probability of Algorithm 1 is the same as the success probability of the forger.*

Proof. By construction, this algorithm outputs "Success" iff the forger succeeds. □

Lemma 6.4.3. $p_2 \geq p_1/2$.

Proof. Suppose a successful forgery requires t queries on the average. Then by the Markov inequality, $2t$ queries are sufficient at least half the time. □

Lemma 6.4.4. $p_3 \geq p_2/2$.

Proof. Recall that $m' = C(M')$ is a codeword of an error-correcting code of distance cn. That implies that for all M, m' differs from $m = C(M)$ on at least cn locations. So, if we picked ℓ locations at random with replacement, $\Pr_J[J(m) = J(m')] \leq (1-c)^{-\ell}$. Since we are picking without replacement, $\Pr_J[J(m) = J(m')] \leq \prod_{i=1}^{\ell}(n - cn - i)/n \leq (1 - c + \epsilon)^{\ell}$ for any constant $\epsilon > 0$. Let $\epsilon = .01c$ for simplicity. Let Q denote the set of messages queried by the adversary. By the union bound

$$\Pr_J[\exists M \in Q \text{ such that } J(C(M)) = J(m')] \leq 2t(1 - c + \epsilon)^{\ell} < 1/2$$

if $4t < (1/(1 - c + \epsilon))^{\ell}$. Taking the logarithm on both sides of the equation, we get the condition $\log t + 2 < \ell \log(1/(1 - .99c))$, and so since we have set $\ell > \frac{\log t + 2}{\log(1/(1-.99c))}$, this is satisfied. □

Lemma 6.4.5. $p_4 \geq p_3/2^{\ell}$.

Proof. Note that Algorithm 3 and Algorithm 4 can be run with the same adversary, but Algorithm 4 may output "Fail" while Algorithm 3 outputs "Success." Consider the case when both of them output "Success." In this case, (1) $J(C(M')) \neq J(C(M))$ for all previously queried M, and (2) $J(C(M)) = B$. Note that, given that (1) is true, the probability of (2) is exactly $2^{-\ell}$. □

Lemma 6.4.6. $p_5 = p_4$.

Proof. Note that the only difference between the two algorithms is that Algorithm 5 will sometimes output "Fail" sooner. Algorithm 4 will continue answering queries, but, if Algorithm 5 has output "Fail," then $J(C(M')) \neq J(C(M)) = B$, and so Algorithm 4 will output "Fail" as well. □

Lemma 6.4.7. $p_6 = p_5$.

Proof. First, note that whether we are running Algorithm 5 or Algorithm 6, the view of the adversary is the same. Namely: (1) the public key is identically distributed; (2) both algorithms respond to at most $2t$ signature queries; (3) both algorithms pick J and B uniformly at random and refuse to respond to a signature query M if $J(C(M)) = B$. Therefore, the probability that the adversary comes up with a forgery in the two cases is the same. Now, note that the probability that the forgery is good is also the same: in both cases, the forgery is good if $J(C(M')) = B$. □

Putting these together, we have:

Lemma 6.4.8. *If the forger's success probability is p, and expected running time is t, then the success probability of the reduction is $p/2^{\ell+2} = p/O(t)$.*

In turn, this implies the following theorem:

Theorem 6.4.9. *Under Assumption 6.2.1, the construction presented in Section 6.3 is a unique signature.*

6.5 A Simple Construction of a VRF

Consider the following, rather strong, complexity assumption:

Assumption 6.5.1. *(**Very-Many-DH-Very-Hard Assumption**) There exists a constant $\epsilon > 0$ such that for all probabilistic polynomial-time families of Turing machines $\{A_k\}$ with running time $O(2^{k^\epsilon})$,*

$$\begin{aligned}&\Pr[(*, q, g) \leftarrow S(1^k); \{y_i \leftarrow \mathbb{Z}_q \ : \ 1 \leq i \leq k^\epsilon\};\\ &Z \leftarrow A_k^{\mathcal{O}[*,q,g,\{y_i\}](\cdot)}(*, q, g, \{g^{y_i} \ : \ 1 \leq i \leq k^\epsilon\}) \ : \ Z = g^{\prod y_i}] \leq poly(k) * 2^{-2k^\epsilon}\end{aligned}$$

where $\mathcal{O}[*, q, g, \{y_i\}](\cdot)$, on input an k^ϵ-bit string I, outputs $g^{\prod_{i=1}^{k^\epsilon} y_i^{I_i}}$ iff I is not an all-1 string.

Definition 6.5.1 (VUF [MRV99]). *A verifiable unpredictable function (VUF) family $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ with input length $a(k)$, output length $b(k)$, and security $s(k)$ is defined as a VRF except that requirement 4 of Definition 6.1.2 is replaced with the following: Let $T(\cdot, \cdot)$ be any oracle algorithm that runs in time $s(k)$ when its first input is 1^k. Then:*

$$\begin{aligned}&\Pr[(PK, SK) \leftarrow G(1^k);\\ &(Q, x, y) \leftarrow T^{\texttt{Prove}(SK,\cdot)}(1^k) \ : \ y = \texttt{Eval}(SK, x) \wedge\\ &\qquad (x, \texttt{Prove}(SK, x)) \notin Q] \leq 1/s(k)\end{aligned}$$

Lemma 6.5.1. *The unique signature in Section 6.3 is a VUF with security 2^{k^ϵ} under Assumption 6.5.1.*

Proof. (Sketch) Following the proof in Section 6.4, let $\ell = k^\epsilon$. Then, by Lemma 6.4.8, using an adversary that succeeds in breaking the unique signature in 2^{k^ϵ} steps with probability 2^{-k^ϵ} corresponds to computing $g^{\prod y_i}$ in time 2^{k^ϵ} with probability $\Omega(2^{-2k\epsilon})$, which contradicts the assumption. □

The following proposition completes the picture:

Proposition 6.5.2 ([MRV99]). *If there is a VUF $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ with input length $a(k)$, output length $b(k)$, and security $s(k)$, then, for any $a'(k) \leq a(k)$,* *there is a VRF* $(G', \texttt{Eval}', \texttt{Prove}', \texttt{Verify}')$ *with input length* $a'(k)$*, output length* $b'(k) = 1$*, and security* $s'(k) = s(k)^{1/3}/(poly(k) \cdot 2^{a'(k)})$*. Namely, the following is a VRF with security* $s'(k)$*:*

- $G'(1^k) = (G(1^k), r)$*, where* r *is a* $b(k)$*-bit string chosen at random.*
- $\texttt{Eval}'(SK, x) = \texttt{Eval}(SK, x).r$*, where "." denotes the inner product.*
- $\texttt{Verify}'(SK, x) = (\texttt{Eval}(SK, x), \texttt{Verify}(SK, x))$.

Corollary 6.5.3. *We have obtained a VRF with input length* k*, output length* 1*, and security* $poly(k)$*.*

A natural open problem is to give better security guarantee to a VRF obtained from a unique signature in this fashion, or to show evidence of impossibility.

6.6 Complicated but More Secure VRF

Here, we construct a VRF and a unique signature based on the weaker assumption alone. This is the same as the Micali et al. [MRV99] construction, except that the underlying verifiable unpredictable function is different.

Proposition 6.6.1 ([MRV99]). *If there is a VRF with input length* $a(k)$*, output length* 1*, and security* $s(k)$*, then there is a VRF with unrestricted input length, output length* 1*, and security at least* $\min(s(k)^{1/5}, 2^{a(k)/5})$*.*

The proof of the proposition gives the VRF construction, which we omit here.

Now let us restate Assumption 6.2.1 to make explicit use of security:

Assumption 6.6.1 (s(k)-Many-DH assumption). *For some* $s(k) > poly(k)$*, for all* $\{A_k\}$*, if* $\{A_k\}$ *is a family of probabilistic Turing machines with running time* $t(k) = O(s(k))$*, then for all* $\ell > \log t(k)$*,*

$$\begin{aligned}\Pr[(*, q, g) \leftarrow S(1^k); \{y_i \leftarrow \mathbb{Z}_q \ : \ 1 \le i \le \ell\};\\ \{z_J = \prod_{j \in J} y_j; Z_J = g^{z_J} \ : \ J \subset [\ell]\};\\ Z \leftarrow A(*, q, g, \{Z_J \ : \ J \subset [\ell]\}) \ : \ Z = g^{\prod y_i}] < 1/s^2(k)\end{aligned}$$

We will construct a VUF with input length $\Omega(\log s'(k))$, and security $s'(k) = 2^{\omega(\log k)}$ based on this assumption. By Propositions 6.5.2 and 6.6.1, this implies a VRF with unrestricted input length and security $\min(s'(k)^{1/5}, 2^{a(k)/5}) = 2^{\omega(\log k)}$.

This is the same as our unique signature construction, only here the public key and the depth of the tree are smaller. The proof works in exactly the same way as in the previous construction.

Theorem 6.6.2. *The following construction is a VUF: set $a_{i,b}$ at random from $\mathbb{Z}_q$, for $1 \leq i \leq \ell$, $\ell > \log s(k)$, $b \in \{0,1\}$. Let $PK = \{g^{a_{i,b}} : 1 \leq i \leq \ell, b \in \{0,1\}\}$, $SK = \{a_{i,b}\}$. If J is a binary string of length ℓ, then let $F_{PK}(g^{\prod_{i=1}^{\ell} a_{i,J_i}})$. The verification is as in the construction described in Section 6.3. Its security is at least $s(k)$ under Assumption 6.6.1.*

Proof. (Sketch) This proof is essentially the same as in Section 6.4, with $n = \ell = \log s(k)$; except this case is simpler as there is no code. Here, too, we hope that the adversary's forgery will be good, and by Lemma 6.4.8, if $s(k)$ is the size of the message space, then $1/s(k)$ of the forgeries will be good. By contrapositive, in $O(s(k))$ running time, the probability of a forgery is $1/s(k)$. Therefore, the probability of a good forgery is $1/s^2(k)$. This contradicts Assumption 6.6.1. $\square$

Combining the above, we get a VRF with unrestricted input length, output length 1, and security

$$\min(s'(k)^{1/5}, 2^{a(k)/5}) = \min(s(k)^{1/10}, 2^{\log(s(k))/5}) = \Theta(s(k)^{1/10}) = 2^{\omega(\log k)}$$

.

Chapter 7

Signatures and Composition of Authenticated Byzantine Agreement

7.1 Introduction

The Byzantine Generals (Byzantine Agreement[1]) problem is one of the most researched areas in distributed computing. Numerous variations of the problem have been considered under different communication models, and both positive results, i.e. protocols, and negative results, i.e. impossibility and lower bounds on efficiency and resources, have been established. The reason for this vast interest is the fact that the Byzantine Generals problem is the algorithmic implementation of a broadcast channel within a point-to-point network. In addition to its importance as a stand-alone primitive, broadcast is a key tool in the design of secure protocols for multiparty computation.

Despite the importance of this basic functionality and the vast amount of research that has been directed towards it, our understanding of the algorithmic issues is far from complete. As is evident from our results, there are still key questions that have not yet been addressed. In this thesis, we provide solutions to some of these questions.

The problem of Byzantine Generals is (informally) defined as follows: There are n parties, one of which is the General who holds an input x. In addition, there is an adversary who controls up to t of the parties and can cause them to arbitrarily deviate from the designated protocol specification. The (honest) parties need to agree on a common value. Furthermore, if the General is not faulty, then this common value must be his original input x.

Pease *et al.* [PSL80, LSP82] provided a solution to the Byzantine Generals problem in the standard model, i.e. the information-theoretic model with point-to-point communication lines (and no setup assumptions). For their solution, the number of faulty parties, t, must be less than $n/3$. Furthermore, they complemented this result

[1]These two problems are essentially equivalent.

by showing that the requirement for $t < n/3$ is in fact inherent. That is, no protocol which solves the Byzantine Generals problem in the standard model can tolerate a third or more faulty parties.

The above bound on the number of faulty parties in the standard model is a severe limitation. It is therefore of great importance to introduce a different (and realistic) model in which it is possible to achieve higher fault tolerance. One possibility involves augmenting the standard model such that messages sent can be authenticated. By authentication, we mean the ability to ascertain that a message was in fact sent by a specific party, even when not directly received from that party. This can be achieved using a trusted preprocessing phase in which a public-key infrastructure for digital signatures (e.g. [RSA78, GMR88]) is set up. (We note that this requires that the adversary be computationally bounded. However, there exist preprocessing phases which do not require any computational assumptions; see Pfitzmann and Waidner [PW96].) Indeed, Pease *et al.* [PSL80, LSP82] use such an augmentation and obtain a protocol for the Byzantine Generals problem which can tolerate *any* number of faulty parties (this is very dramatic considering the limitation to 1/3 faulty in the standard model). The Byzantine Generals problem in this model is referred to as *authenticated* Byzantine Generals.

A common use of Byzantine Generals is to substitute for a broadcast channel. Therefore, it is clear that the settings in which we would want and need to run it involve many invocations of the Byzantine Generals protocol. The question of whether these protocols remain secure when executed concurrently, in parallel or sequentially is thus an important one. However, existing work on this problem (in both the standard and authenticated models) focused on the security and correctness of protocols in a single execution only.

It is easy to see that the unauthenticated protocol of Pease *et al.* [PSL80], and other protocols in the standard model, do compose concurrently (and hence in parallel and sequentially). However, this is not the case with respect to *authenticated* Byzantine Generals. The first to notice that composition in this model is problematic were Gong, Lincoln, and Rushby [GLR95], who also suggest methods for overcoming the problem. Our work shows that these suggestions and any others are futile; in fact composition in this model is impossible (as long as 1/3 or more of the parties are faulty). (We note that by composition, we refer to stateless composition; see Section 7.2.3 for a formal discussion.)

Our Results. Our first theorem, stated below, shows that authenticated Byzantine Generals protocols, both deterministic and randomized, cannot be composed in parallel (and thus concurrently). This is a surprising and powerful statement with respect to the issue of enhancing the standard model by the addition of authentication. The theorem shows that this enhancement *does not* provide the ability to overcome the impossibility result when composition is required. That is, if there is a need for parallel composition, then the number of faulty players cannot be $n/3$ or more, and hence the authenticated model provides no advantage over the standard model.

Theorem 7.1.1. *No protocol for authenticated Byzantine Agreement that composes in parallel (even twice) can tolerate $n/3$ or more faulty parties.*

Regarding the question of sequential composition, we show different results. We first prove another (weaker) lower bound for *deterministic* protocols:

Theorem 7.1.2. *Let Π be a* deterministic *protocol for authenticated Byzantine Agreement that terminates within r rounds of communication. Then, Π can be sequentially composed at most $2r-1$ times.*

In contrast, for *randomized* protocols we obtain positive results and present a protocol which can be composed sequentially (any polynomial number of times), and which tolerates $t < n/2$ faulty parties. The protocol which we present is based on a protocol of Fitzi and Maurer [FM00] that tolerates $t < n/2$ faulty parties, and is in the standard model augmented with an ideal three-party broadcast primitive. We show that this primitive can be replaced by an authenticated protocol for three parties that can be composed sequentially (and the resulting protocol also composes sequentially). Thus, we prove:

Theorem 7.1.3. *Assume that there exists a signature scheme that is existentially secure against chosen message attacks. Then there exists a randomized protocol for authenticated Byzantine Generals with a bounded number of rounds, that tolerates $t < n/2$ faulty parties and composes sequentially any polynomial number of times.*

We also present a randomized Byzantine Generals protocol that tolerates *any* number of faulty parties, and composes sequentially any polynomial number of times. However, the number of messages sent in this protocol is exponential in the number of parties. Therefore, it can only be used when the overall number of parties is logarithmic in the security parameter of the signature scheme.

On the Use of Unique Session Identifiers. As will be apparent from the proofs of the lower bounds (Theorems 7.1.1 and 7.1.2), what prevents agreement in this setting is the fact that honest parties cannot tell in which execution of the protocol a given message was authenticated. This allows the adversary to "borrow" messages from one execution to another, and by that attack the system. In Section 7.5, we show that if we further augment the authenticated model so that unique and common indices are assigned to each execution, then security under many concurrent executions can be achieved (for any number of faulty parties).

Thus, on the one hand, our results strengthen the common belief that session identifiers are necessary for achieving authenticated Byzantine Generals. On the other hand, we show that such identifiers *cannot be generated within the system.* Typical suggestions for generating session identifiers in practice include having the General choose one, or having the parties exchange random strings and set the identifier to be the concatenation of all these strings. However, Theorem 7.1.1 rules out all such solutions (notice that just coming up with a common identifier involves reaching agreement). Rather, one must assume the existence of some *trusted external means* for coming up with unique and common indices. This seems to be a very difficult, if not impossible, assumption to realize in many natural settings.

A natural question to ask here relates to the fact that unique and common session identifiers are anyway needed in order to correctly route incoming messages to the right copy of the protocol, when several copies of the protocol are running concurrently. Indeed, global session identifiers solve this problem. However, it also suffices for each party to allocate *local* identifiers for itself. That is, when a party begins a new execution, it chooses a unique identifier *sid* and informs all parties to concatenate *sid* to any message they send him within this execution. It is then guaranteed that any message sent by an honest party to another honest party will be directed to the execution it belongs to. We thus conclude that for the purposes of message routing in concurrent executions, global identifiers are not needed.

Our impossibility results are in the model that assumes some initialization phase before the execution of any protocol starts. Thus, we show that no initialization phase can set up the protocol in such a way that global identifiers will be readily available or easily established for concurrent, or even parallel, execution of protocols. We do not rule out that some external auxiliary inputs, such as a global clock, may give us appropriate session identifiers. Indeed, this is a subject of further study.

Implications for Secure Multiparty Computations. As we have stated above, one important use for Byzantine Generals protocols is to substitute the broadcast channel in a multiparty protocol. In fact, most known solutions for multiparty computations assume a broadcast channel, claiming that it can be substituted by a Byzantine Generals protocol without any complications. Our results therefore imply that multiparty protocols that rely on authenticated Byzantine Generals to replace the broadcast channel, cannot be composed in parallel or concurrently.

Another important implication of our result is due to the fact that any secure protocol for solving *general* multiparty tasks can be used to solve Byzantine Generals. Therefore, none of these protocols can be composed in parallel or concurrently, unless more than 2/3 of the parties are honest or a physical broadcast channel is available.

Our Work vs. Composition of Secure Multiparty Protocols. There has been much work on the topic of protocol composition in the context of multiparty computation [Bea92, MR92, Can00, DM00, Can01]. Much of this work has focused on zero-knowledge and concurrent zero-knowledge protocols [GK96, DNS98, RK99, CKPR01]. For example, Goldreich and Krawczyk [GK96] show that there exist protocols that are zero-knowledge when executed stand-alone, and yet do not compose in parallel (even twice). However, protocols that compose do exist (see, for example, Goldreich [Gol02] and references therein). In contrast, we show that it is impossible to obtain *any protocol* that will compose twice in parallel.

7.2 Definitions

7.2.1 Computational Model

We consider a setting involving n parties, $P_1, \ldots, P_n$, that interact in a *synchronous* point-to-point network. In such a network, each pair of parties is directly connected, and it is assumed that the adversary cannot modify messages sent between honest parties. In this setting, each party is formally modeled by an interactive Turing machine with $n-1$ pairs of communication tapes. The communication of the network proceeds in synchronized rounds, where each round consists of a *send* phase followed by a *receive* phase. In the *send* phase of each round, the parties write messages onto their output tapes, and in the *receive* phase, the parties read the contents of their input tapes.

This paper refers to the *authenticated* model, where some type of *trusted preprocessing phase* is assumed. This is modeled by all parties also having an additional setup-tape that is generated during the preprocessing phase. Typically, in such a preprocessing phase, a public-key infrastructure of signature keys is generated. That is, each party receives its own secret signing key, and in addition, public verification keys associated with all other parties. (This enables parties to use the signature scheme to authenticate messages that they receive, and is thus the source of the name "authenticated".) However, we stress that our lower bound holds for all preprocessing phases (even those that cannot be efficiently generated).

In this model, a t-adversary is a party that controls $t < n$ of the parties $P_1, \ldots, P_n$, where the corruption strategy depends on the adversary's view (i.e., the adversary is adaptive). Since the adversary controls these parties, it receives their entire views and determines the messages that they send. In particular, these messages need not be according to the protocol execution, but rather can be computed by the adversary as an arbitrary function of its view. We note that our impossibility results hold even against a static adversary (for whom the set of faulty parties is fixed before the execution begins).

Note that our protocols for authenticated Byzantine Agreement that compose sequentially rely on the security of signature schemes, and thus assume probabilistic polynomial-time adversaries only. On the other hand, our impossibility results hold for adversaries (and honest parties) whose running time is of any complexity. In fact, the adversary that we construct to prove our lower bounds is of the same complexity as the *honest* parties.

7.2.2 Byzantine Generals/Agreement

The existing literature defines two related problems: Byzantine Generals and Byzantine Agreement. In the first problem, there is one designated party, the General, who wishes to broadcast its value to all the other parties. In the second problem, each party has an input and the parties wish to agree on a value, with a validity condition that if a majority of honest parties begin with the same value, then they must terminate with that value. These problems are equivalent in the sense that any protocol solving one can be used to construct a protocol solving the other, while tolerating the same number of faulty parties. We relax the standard requirements on protocols for the above Byzantine problems in that we allow a protocol to fail with probability that is negligible in some security parameter. This relaxation is needed for the case of authenticated Byzantine protocols where signature schemes are used (and can always be forged with some negligible probability). Formally,

Definition 7.2.1 (Byzantine Generals). *Let $P_1, \dots, P_{n-1}$ and $G = P_n$ be n parties and let G be the designated party with input x. In addition there is an adversary who may corrupt up to t of the parties including the special party G. A protocol solves the Byzantine Generals problem if the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*

2. Validity: *If G is honest, then all honest parties output x.*

We denote such a protocol by $BG_{n,t}$.

In the setting of Byzantine Agreement it is not straightforward to formulate the validity property. Intuitively, it should capture that if enough honest parties begin with the same input value then they will output that value. By "honest," we mean the parties that follow the prescribed protocol exactly, ignoring the issue that the first step of the party might be to change its local input.

Definition 7.2.2 (Byzantine Agreement). *Let $P_1, \dots, P_n$ be n parties, with associated inputs $x_1, \dots, x_n$. In addition there is an adversary who may corrupt up to t of the parties. Then, a protocol solves the Byzantine Agreement problem if the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*

2. Validity: *If $\max(n-t, \lfloor n/2 \rfloor + 1)$ of the parties have the same input value x and follow the protocol specification, then all honest parties output x.*

We note that for the information-theoretic setting, the validity requirement is usually stated so that it must hold only when more than *two thirds* of the parties have the same input value, because in the information-theoretic setting, $n - t > 2n/3$.

Authenticated Byzantine Protocols: In the model for authenticated Byzantine Generals/Agreement, some trusted preprocessing phase is run before any executions begin. In this phase, a trusted party distributes keys to every participating party. Formally,

Definition 7.2.3 (Authenticated Byzantine Generals/Agreement). *A protocol for authenticated Byzantine Generals/Agreement is a Byzantine Generals/Agreement protocol with the following augmentation:*

- *Each party has an additional* setup-tape.

- *Prior to any protocol execution, an ideal (trusted) party chooses a series of strings $s_1, \ldots, s_n$ according to some distribution, and sets party P_i's* setup-tape *to equal s_i (for every $i = 1, \ldots, n$).*

Following the above preprocessing stage, the protocol is run in the standard communication model for Byzantine Generals/Agreement protocols.

As we have mentioned, a natural example of such a preprocessing phase is one where the strings $s_1, \ldots, s_n$ constitute a public-key infrastructure. That is, the trusted party chooses key-pairs $(pk_1, sk_1), \ldots, (pk_n, sk_n)$ from a secure signature scheme, and sets the contents of party P_i's tape to equal $s_i = (pk_1, \ldots, pk_{i-1}, pk_i, sk_i, pk_{i+1}, \ldots, pk_n)$. That is, all parties are given their own signing key and the verification keys of all the other parties.

We remark that the above-defined preprocessing phase is very strong. First, it is assumed that it is run completely by a trusted party. Furthermore, there is no computational bound on the power of the trusted party generating the keys. Nevertheless, our impossibility results hold even for such a preprocessing phase.

7.2.3 Composition of Protocols

This paper deals with the security of authenticated Byzantine Agreement protocols, when the protocol is executed many times (rather than just once). We define the composition of protocols to be *stateless.* This means that the honest parties act upon their view in a single execution only. Furthermore, in stateless composition, there is no unique session identifier that is common to all participating parties. (See the Introduction for a discussion on session identifiers and their role.)

The reason that stateless composition is of interest, is that if it is the easiest kind of composition. In particular, this means that the honest parties do not store in memory their views from previous executions or coordinate between different executions occurring at the current time. Whenever possible, stateless composition simplifies protocol design and reduces assumptions on the model.

We note that although the parties are stateless, the adversary is allowed to maliciously coordinate between executions and record its view from previous executions. Formally, composition is captured by the following process:

Definition 7.2.4 (sequential and parallel composition). *Let $P_1, \ldots, P_n$ be parties for an authenticated Byzantine Generals/Agreement protocol Π. Let $I \subseteq [n]$ be an index set such that for every $i \in I$, the adversary $\mathcal{A}$ controls the party P_i. Over time, indices are added to I as the adversary chooses to corrupt additional parties, with the restriction that $|I| \leq t$. Then, the* sequential (resp., parallel) composition *of Π involves the following process:*

- *Run the preprocessing phase associated with Π and obtain the strings $s_1, \ldots, s_n$. Then, for every j, set the* setup-tape *of P_j to equal s_j.*

- *Repeat the following process a polynomial number of times sequentially (resp., in parallel).*

1. *The adversary $\mathcal{A}$ chooses an input vector $x_1, \ldots, x_n$.*
2. *Fix the input tape of every honest P_j to be x_j and its random tape to be a uniformly (and independently) chosen random string.*
3. *Invoke all parties for an execution of Π (using the strings generated in the preprocessing phase above). The execution is such that for $i \in I$, the messages sent by party P_i are determined by $\mathcal{A}$ (who also sees P_i's view). On the other hand, all other parties follow the instructions as defined in Π.*

We stress that the preprocessing phase is executed only once and all executions use the strings distributed in this phase. Furthermore, we note that Definition 7.2.4 implies that all honest parties are oblivious of the other executions that have taken place (or that are taking place in parallel). This is implicit in the fact that in each execution the parties are invoked with no additional state information, beyond the contents of their input, random and key tapes. On the other hand, the adversary $\mathcal{A}$ can coordinate between the executions, and its view at any given time includes all the messages received in all other executions.[2]

Before proceeding, we show that any Byzantine Generals (or Agreement) protocol *in the standard model* composes concurrently.

Proposition 7.2.1. *Any protocol Π for Byzantine Generals (or Agreement) in the* standard model, *remains secure under concurrent composition.*

Proof. We reduce the security of Π under concurrent composition to its security for a single execution. Assume by contradiction that there exists an adversary $\mathcal{A}$ who runs N concurrent executions of Π, such that with non-negligible probability, in one of the executions the outputs of the parties are not according to the requirements of the Byzantine Generals. We construct an adversary $\mathcal{A}'$ who internally incorporates $\mathcal{A}$ and attacks a single execution of Π. Intuitively, $\mathcal{A}'$ simulates all executions apart from the one in which $\mathcal{A}$ succeeds in its attack. Formally, $\mathcal{A}'$ begins by choosing an index $i \in_R \{1, \ldots, N\}$. Then, for all but the i^{th} execution of the protocol, $\mathcal{A}'$ plays the roles of the honest parties in an interaction with $\mathcal{A}$ (this simulation is internal to $\mathcal{A}'$). On the other hand, for the i^{th} execution, $\mathcal{A}'$ externally interacts with the honest parties and passes messages between them and $\mathcal{A}$ (which it runs internally). The key point in the proof is that the honest parties hold no secret information (and do not coordinate between executions). Therefore, the simulation of the concurrent setting by $\mathcal{A}'$ for $\mathcal{A}$ is *perfect.* Thus, with probability $1/N$, the i^{th} execution is the one in which $\mathcal{A}$ succeeds. However, this means that $\mathcal{A}'$ succeeds in breaking the protocol for a single execution (where $\mathcal{A}'$'s success probability equals $1/N$ times the success probability of $\mathcal{A}$.) This contradicts the stand-alone security of Π. □

[2]The analogous definition for the composition of *unauthenticated* Byzantine Generals/Agreement is derived from Definition 7.2.4 by removing the reference to the preprocessing stage and setup-tapes.

7.3 Impossibility Results

In this section we present two impossibility results regarding the composition of authenticated Byzantine Agreement protocols. Recall that we are concerned with *stateless* composition. First, we show that it is impossible to construct an authenticated Byzantine Agreement protocol that composes in parallel (or concurrently), and is secure when $n/3$ or more parties are faulty. This result is analogous to the Fischer *et al.* [FLM86] lower bound for Byzantine Agreement in the standard model (i.e., without authentication). We stress that our result does not merely show that authenticated Byzantine Agreement protocols do not necessarily compose; rather, we show that one *cannot* construct protocols that will compose. Since there exist protocols for unauthenticated Byzantine Agreement that are resilient for any $t < n/3$ faulty parties and compose concurrently, this shows that the advantage gained by the preprocessing step in authenticated Byzantine Agreement protocols is lost when composition is required.

Next, we show a lower bound on the number of rounds required for *deterministic* authenticated Byzantine Agreement that composes sequentially. (Note that the impossibility of *parallel* composition holds even for randomized protocols.) We show that if an authenticated Byzantine Agreement protocol that tolerates $n/3$ or more faulty parties is to compose sequentially r times, then there are executions in which it runs for more than $r/2$ rounds. Thus, the number of rounds in the protocol is linear in the number of times it is to compose. This rules out any practical protocol that will compose for a (large) polynomial number of times.

Intuition. Let us first provide some intuition into why the added power of the preprocessing step in authenticated Byzantine Agreement does not help when composition is required. (Recall that in the stand-alone setting, there exist authenticated Byzantine Agreement protocols that tolerate any number of faulty parties. On the other hand, under parallel composition, more than $2n/3$ parties must be honest.) An instructive step is to first see how authenticated Byzantine Agreement protocols typically utilize the preprocessing step, in order to increase fault tolerance. A public-key infrastructure for signature schemes is used and this helps in achieving agreement for the following reason. Consider three parties A, B and C participating in a standard (unauthenticated) Byzantine Agreement protocol. Furthermore, assume that during the execution A claims to B that C sent it some message x. Then, B cannot differentiate between the case that C actually sent x to A, and the case that C did not send this value and A is faulty. Thus, B cannot be sure that A really received x from C. Indeed, such a model has been called the "oral message" model, in contrast to the "signed message" model of authenticated Byzantine Agreement [LSP82]. On the other hand, the use of signature schemes helps to overcome this exact problem: If C had *signed* the value x and sent this signature to A, then A could forward the signature to B. Since A cannot forge C's signature, this would then constitute a proof that C indeed sent x to A. Therefore, utilizing the unforgeability property of signatures, it is possible to achieve Byzantine Agreement for any number of faulty parties.

However, the above intuition holds only in a setting where a single execution of the agreement protocol takes place. Specifically, if a number of executions were to take place, then A may send B a value x along with C's signature on x, yet B would still not know whether C signed x in *this* execution, or in a different (concurrent or previous) execution. Thus, the mere fact that A produces C's signature on a value does not provide proof that C signed *this* value in *this* execution. As we will see in the proof, this is enough to render the public-key infrastructure useless under some types of composition.

We remark that it is possible to achieve concurrent composition, using state in the form of unique and common session identifiers. However, as we have mentioned, there are many scenarios where this does not seem to be achievable (and many others where it is undesirable).

Theorem 7.1.1 *No protocol for authenticated Byzantine Agreement that composes in parallel (even twice) can tolerate $n/3$ or more faulty parties.*

Proof. The proof of Theorem 7.1.1 is based on some of the ideas used by Fischer et al. [FLM86] in their proof that no unauthenticated Byzantine Agreement protocol can tolerate $n/3$ or more faulty parties. We begin by proving the following lemma:

Lemma 7.3.1. *There exists no protocol for authenticated Byzantine Agreement for three parties, that composes in parallel (even twice) and can tolerate one faulty party.*

Proof. Assume, by contradiction, that there exists a protocol Π that solves the Byzantine Agreement problem for three parties A, B and C, where one may be faulty. Furthermore, Π remains secure even when composed in parallel twice. Exactly as in the proof of Fischer et al. [FLM86], we define a hexagonal system S that intertwines two independent copies of Π. That is, let A_1, B_1, C_1 and A_2, B_2 and C_2 be independent copies of the three parties participating in Π. By independent copies, we mean that A_1 and A_2 are the same party A with the same key tape, that runs in two different parallel executions of Π, as defined in Definition 7.2.4. The system S is defined by connecting party A_1 to C_2 and B_1 (rather than to C_1 and B_1); party B_1 to A_1 and C_1; party C_1 to B_1 and A_2; and so on, as in Figure 7-1.

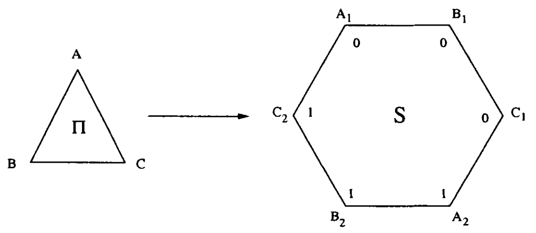

Figure 7-1: Combining two copies of Π in a hexagonal system S.

In the system S, parties A_1, B_1, and C_1 have input 0; while parties A_2, B_2 and C_2 have input 1. Note that within S, all parties follow the instructions of Π exactly. We stress that S is *not* a Byzantine Agreement setting (where the parties are joined in a complete graph on three nodes), and therefore the definitions of Byzantine Agreement tell us nothing directly of what the parties' outputs should be. However, S *is* a well-defined system and this implies that the parties have well-defined output distributions. The proof proceeds by showing that if Π is a correct Byzantine Agreement protocol, then we arrive at a contradiction regarding the output distribution in S. We begin by showing that B_1 and C_1 output 0 in S. We denote by $\mathsf{rounds}(\Pi)$ the upper bound on the number of rounds of Π (when run in a Byzantine Agreement setting).

Claim. *Except with negligible probability, parties B_1 and C_1 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 0 *in the system S.*

Proof. We prove this claim by showing that there exists a faulty party (or adversary) $\mathcal{A}$ who participates in two parallel copies of Π and simulates the system S, with respect to B_1 and C_1's view. The faulty party $\mathcal{A}$ (and the other honest parties participating in the parallel execution) work within a Byzantine Agreement setting where there are well-defined requirements on their output distribution. Therefore, by analyzing their output in this parallel execution setting, we are able to make claims regarding their output in the system S.

Let A_1, B_1 and C_1 be parties running an execution of Π, denoted Π_1, where B_1 and C_1 both have input 0. Furthermore, let A_2, B_2 and C_2 be running a parallel execution of Π, denoted Π_2, where B_2 and C_2 both have input 1. Recall that B_1 and B_2 are independent copies of the party B with the same key tape (as defined in Definition 7.2.4); likewise for C_1 and C_2.

Now, let $\mathcal{A}$ be an adversary who controls both A_1 in Π_1 and A_2 in Π_2 (recall that the faulty party can coordinate between the different executions). Party $\mathcal{A}$'s strategy is to maliciously generate an execution in which B_1's and C_1's view in Π_1 is *identical* to their view in S. $\mathcal{A}$ achieves this by redirecting edges of the two parallel triangles (representing the parallel execution), so that the overall system has the same behavior as S; see Figure 7-2.

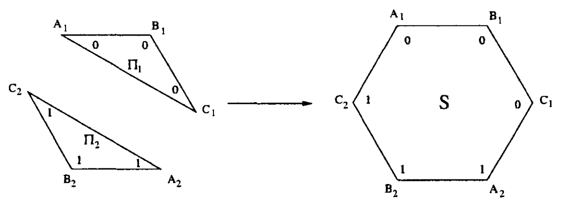

Figure 7-2: Redirecting edges of Π_1 and Π_2 to make a hexagon.

Specifically, the (A_1, C_1) and (A_2, C_2) edges of Π_1 and Π_2 respectively are removed, and the (A_1, C_2) and (A_2, C_1) edges of S are added in their place. $\mathcal{A}$ is able to make such a modification because it only involves redirecting messages to and from parties that it controls (i.e., A_1 and A_2).

Before proceeding, we present the following notation: let $\mathsf{msg}_i(A_1, B_1)$ denote the message sent from A_1 to B_1 in the i^{th} round of the protocol execution. We now formally show how the adversary $\mathcal{A}$ works. $\mathcal{A}$ invokes parties A_1 and A_2, upon inputs 0 and 1 respectively. We stress that A_1 and A_2 follow the instructions of protocol Π exactly. However, $\mathcal{A}$ provides them with their incoming messages and sends their outgoing messages for them. The only malicious behavior of $\mathcal{A}$ is in the redirection of messages to and from A_1 and A_2. A full description of $\mathcal{A}$'s code is as follows (we recommend the reader to refer to Figure 7-2 in order to clarify the following):

1. *Send outgoing messages of round i:* $\mathcal{A}$ obtains messages $\mathsf{msg}_i(A_1, B_1)$ and $\mathsf{msg}_i(A_1, C_1)$ from A_1 in Π_1, and messages $\mathsf{msg}_i(A_2, B_2)$ and $\mathsf{msg}_i(A_2, C_2)$ from A_2 in Π_2 (these are the round i messages sent by A_1 and A_2 to the other parties; as we have mentioned, A_1 and A_2 compute these messages according to the protocol definition and based on their view).

   - In Π_1, $\mathcal{A}$ sends B_1 the message $\mathsf{msg}_i(A_1, B_1)$ and sends C_1 the message $\mathsf{msg}_i(A_2, C_2)$ (and thus the (A_1, C_1) directed edge is replaced by the directed edge (A_2, C_1)).
   - In Π_2, $\mathcal{A}$ sends B_2 the message $\mathsf{msg}_i(A_2, B_2)$ and sends C_2 the message $\mathsf{msg}_i(A_1, C_1)$ (and thus the (A_2, C_2) directed edge is replaced by the directed edge (A_1, C_2)).

2. *Obtain incoming messages from round i:* $\mathcal{A}$ receives messages $\mathsf{msg}_i(B_1, A_1)$ and $\mathsf{msg}_i(C_1, A_1)$ from B_1 and C_1 in round i of Π_1, and messages $\mathsf{msg}_i(B_2, A_2)$ and $\mathsf{msg}_i(C_2, A_2)$ from B_2 and C_2 in round i of Π_2.

   - $\mathcal{A}$ passes A_1 in Π_1 the messages $\mathsf{msg}_i(B_1, A_1)$ and $\mathsf{msg}_i(C_2, A_2)$ (and thus the (C_1, A_1) directed edge is replaced by the directed edge (C_2, A_1)).
   - $\mathcal{A}$ passes A_2 in Π_2 the messages $\mathsf{msg}_i(B_2, A_2)$ and $\mathsf{msg}_i(C_1, A_1)$ (and thus the (C_2, A_2) directed edge is replaced by the directed edge (C_1, A_2)).

We now claim that B_1 and C_1's view in Π_1 is identical to B_1 and C_1's view in S.[3] This holds because in the parallel execution of Π_1 and Π_2, all parties follow the protocol definition (including A_1 and A_2). The same is true in the system S, except that party A_1 is connected to B_1 and C_2 instead of to B_1 and C_1. Likewise, A_2 is connected to B_2 and C_1 instead of to B_2 and C_2. However, by the definition of $\mathcal{A}$, the messages seen by all parties in the parallel execution of Π_1 and Π_2 are exactly the same as the messages seen by the parties in S (e.g., the messages seen by C_1 in Π_1 are those sent by B_1 and A_2, exactly as in S). Therefore, the views of B_1 and C_1 in the parallel execution maliciously controlled by $\mathcal{A}$, are *identical* to their views in S.[4]

[3]In fact, the views of *all* the parties in the parallel execution with $\mathcal{A}$ are identical to their view in the system S. However, in order to obtain Claim 7.3, we need only analyze the views of B_1 and C_1.

[4]We note the crucial difference between this proof and that of Fischer et al. [FLM86]: the faulty

By the assumption that Π is a correct Byzantine Agreement protocol that composes twice in parallel, we have that, except with negligible probability, in Π_1 both B_1 and C_1 halt within $\mathsf{rounds}(\Pi)$ steps and output 0. The fact that they both output 0 is derived from the fact that B_1 and C_1 are an honest majority with the same input value 0. Therefore, they must output 0 in the face of *any* adversarial A_1; in particular this holds with respect to the specific adversary $\mathcal{A}$ described above. Since the views of B_1 and C_1 in S are identical to their views in Π_1, we conclude that in the system S, they also halt within $\mathsf{rounds}(\Pi)$ steps and output 0 (except with negligible probability). This completes the proof of the claim. □

Using analogous arguments, we obtain the following two claims:

Claim. *Except with negligible probability, parties A_2 and B_2 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 1 *in the system S.*

In order to prove this claim, the faulty party is $\mathcal{C}$ and it works in a similar way to $\mathcal{A}$ in the proof of Claim 7.3 above. (The only difference is regarding the edges that are redirected.)

Claim. *Except with negligible probability, parties A_2 and C_1 halt within* $\mathsf{rounds}(\Pi)$ *steps and output the same value in the system S.*

Similarly, this claim is proven by taking the faulty party as $\mathcal{B}$ who follows a similar strategy to $\mathcal{A}$ in the proof of Claim 7.3 above.

Combining Claims 7.3, 7.3 and 7.3 we obtain a contradiction. This is because, on the one hand C_1 must output 0 in S (Claim 7.3), and A_2 must output 1 in S (Claim 7.3). On the other hand, by Claim 7.3, parties A_2 and C_1 must output the same value. This concludes the proof of the lemma. □

Theorem 7.1.1 is derived from Lemma 7.3.1 in the standard way [PSL80, LSP82] by showing that if there exists a protocol that is correct for any $n \geq 3$ and $n/3$ faulty parties, then one can construct a protocol for 3 parties that can tolerate one faulty party. This is in contradiction to Lemma 7.3.1, and thus Theorem 7.1.1 is implied. □

The following corollary, referring to concurrent composition, is immediately derived from the fact that parallel composition (where the scheduling of the messages is fixed and synchronized) is merely a special case of concurrent composition (where the adversary controls the scheduling).

Corollary 7.3.2. *No protocol for authenticated Byzantine Agreement that composes concurrently (even twice) can tolerate $n/3$ or more faulty parties.*

party $\mathcal{A}$ is able to simulate the entire $A_1-C_2-B_2-A_2$ segment of the hexagon system S by itself. Thus, in a *single* execution of Π with B_1 and C_1, party $\mathcal{A}$ can simulate the hexagon. Here, due to the fact that the parties B_2 and C_2 have secret information that $\mathcal{A}$ does not have access to, $\mathcal{A}$ is unable to simulate their behavior itself. Rather, $\mathcal{A}$ needs to redirect messages from the parallel execution of Π_2 in order to complete the hexagon.

7.3.1 Sequential Composition of Deterministic Protocols

We now show that there is a significant limitation on *deterministic* Byzantine Agreement protocols that compose *sequentially*. Specifically, any protocol which terminates within r rounds can only be composed sequentially for at most $2r$–1 times. The lower bound is derived by showing that for any deterministic protocol Π, r rounds of the hexagonal system S (see Figure 7-1) can be simulated in $2r$ sequential executions of Π. As we have seen in the proof of Theorem 7.1.1, the ability to simulate S results in a contradiction to the correctness of the Byzantine Agreement protocol Π. However, a contradiction is only derived if the system S halts. Nevertheless, since Π terminates within r rounds, the system S also halts within r rounds. We conclude that the protocol Π can be sequentially composed at most $2r-1$ times.

We remark that in actuality, one can prove a more general statement that says that for *any* deterministic protocol, r rounds of 2 parallel executions of the protocol can be perfectly simulated in $2r$ sequential executions of the same protocol. (More generally, r rounds of k parallel executions of a protocol can be simulated in $k \cdot r$ sequential executions.) Thus, essentially, the deterministic sequential lower bound is derived by reducing it to the parallel composition case of Theorem 7.1.1. That is,

Theorem 7.1.2 *Let Π be a* deterministic *protocol for authenticated Byzantine Agreement that concludes after r rounds of communication. Then, Π can be sequential composed at most $2r-1$ times.*

7.4 Sequentially Composable Randomized Protocols

In this section we present two results. The first one is a protocol which tolerates any $t < n/2$ faulty parties and has polynomial communication complexity (i.e., bandwidth). The second one is a protocol that can tolerate any number of faulty parties but is exponential in the number of participating parties.

The building block for both of the above protocols will be a randomized (sequentially composable) protocol, $\text{ABG}_{3,1}$, for authenticated Byzantine Generals between 3 parties and tolerating one faulty party. Recall that $\text{ABG}_{n,t}$ denotes an authenticated Byzantine Generals protocol for n parties that tolerates up to t faults. We first present the protocol $\text{ABG}_{3,1}$ and then show how it can be used to achieve the above-described results.

7.4.1 Sequentially Composable $\text{ABG}_{3,1}$

For this protocol we assume three parties: the general, G, and the recipients P_1, P_2. The General has an input value x. According to Definition 7.2.1, parties P_1 and P_2 need to output the same value x', and, if G is not faulty, then $x' = x$. As is evident from the proofs of the impossibility, what hinders a solution is that faulty parties can *import* messages from previous executions, and there is no means to distinguish between those and the current messages. Thus, if some freshness could be introduced in the signatures, then this would foil the adversary's actions. Yet, agreeing on such freshness would put us in a circular problem. Nevertheless, the case of three parties is different: here there are only two parties who need to receive each signature. Furthermore, it turns out that it suffices if the parties who are *receiving* a signature can jointly agree on a fresh string. Fortunately, *two* parties can easily agree on a new fresh value: they simply exchange messages and set the fresh string to equal the concatenation of the exchanged values. Now, in the protocol which follows for three parties, we require that whenever a party signs a message, it uses freshness generated by the two remaining parties. We note that in the protocol, only the General, G, signs a message, and therefore only it needs a public key. The protocol is described in Figure 7-3. For simplicity, we assume that the signature scheme is defined such that $\sigma_{pk}(z)$ also contains the value z.

Public Input: Security parameter 1^k
Public key PK associated with G
Private Input of G: Secret key SK corresponding to PK
Value x

1. P_1 and P_2 agree on a random label ℓ, as follows:
   (a) P_i chooses a random k-bit string u_i and sends it to P_j
   (b) Set $\ell = u_1 \circ u_2$, where $\circ$ denotes concatenation.
2. P_1 and P_2 both send ℓ to G.
3. Let ℓ_i denote the message that G received from P_i in the previous round. G forms $m = \sigma_{SK}(x, \ell_1, \ell_2)$ and sends m to P_1 and P_2.
4. Denote by m_i the message received by P_i in the previous step from G. P_i checks whether the message m_i is a valid signature on the label ℓ (and another, possibly different, label). If so, P_i forwards m_i to P_j.
5. P_i denotes by m'_j the message that it received from P_j in the previous step. P_i outputs according to the following rules based on the examination of m_i and m'_j:
   If all valid signatures on messages containing the label ℓ are for the same value x, then output x.
   Otherwise, output a default value.

Figure 7-3: Authenticated BG for Three Parties

As we will wish to incorporate Protocol 7-3 into a protocol with n parties we state a broader claim for the composition than for a simple three party setting.

Lemma 7.4.1. *Assume that the signature scheme σ is existentially secure against adaptive chosen message attacks. Then, Protocol* $7-3$ *is a secure protocol for* $ABG_{3,1}$ *that can be composed sequentially within a system of* n *parties, in which* t *may become faulty, for any* $t < n$.

Proof. We prove the theorem by contradiction. Assume that a series of $\text{ABG}_{3,1}$ protocols are run sequentially, such that in some (or all) of them, the adversary succeeded in foiling agreement with non-negligible probability. We will show that in such a case, we can construct a forger $\mathcal{F}$ for the signature scheme who succeeds with non-negligible probability. This will then be in contradiction to the security of the signature scheme.

As there are n parties and the adversary can control up to t of them, there may be executions where two or three of the parties are corrupted. However, in such a case, agreement holds vacuously. On the other hand, any execution in which all three parties are honest must be correct. Therefore, agreement can only be foiled in the case that exactly one participating party is corrupted.

We first claim that when $\mathcal{A}$ plays the General in an execution, it cannot foil the agreement. This is because P_1 and P_2's views of the messages sent by $\mathcal{A}$ (playing G) are identical. Furthermore, their decision making process based on their view is deterministic. Therefore, they must output the same value. We stress that this is irrespective of how many executions have passed (and is also not dependent at all on the security of the signature scheme being used).

Thus, it must be the case that the foiled execution is one where the general is an honest party. As we have mentioned, we build a forger $\mathcal{F}$ for the signature scheme σ who uses $\mathcal{A}$. The forger $\mathcal{F}$ receives as input a public verification-key PK, and access to a signing oracle associated with this key. $\mathcal{F}$ begins by choosing at random one of the parties, say P_j, and associating the verification-key pk with this party. Intuitively, with probability $1/n$, this is the party who plays the general when $\mathcal{A}$ foils the agreement. For all other parties, the forger $\mathcal{F}$ chooses a key pair, for which it knows both the signing and verification keys. Then, $\mathcal{F}$ gives the adversary $\mathcal{A}$ the key pairs for all the initially corrupted parties. $\mathcal{F}$ now invokes $\mathcal{A}$ and simulates the roles of the honest parties in the sequential executions of Protocol 7-3, with $\mathcal{A}$ as the adversary. In particular, $\mathcal{F}$ works as follows:

- In all executions where the recipient/s P_1 and/or P_2 are not corrupted, $\mathcal{F}$ plays their role, following the protocol exactly as specified. This is straightforward as the recipients do not use signing keys during such an execution.

- In all executions where the general is some uncorrupted party $P_l \neq P_j$, the forger $\mathcal{F}$ plays the role of P_l, following the protocol and using the signing-key which it associated with P_l initially.

- In all executions where the general is the uncorrupted P_j, the forger $\mathcal{F}$ plays the role of P_j following the protocol. However, in this case, $\mathcal{F}$ does not have the associated signing-key. Nevertheless, it does have access to the signing oracle associated with pk (which is P_j's public verification-key). Therefore, $\mathcal{F}$ executes these signatures by accessing its oracle. In particular, for labels ℓ_1, ℓ_2 that it receives during the simulation, it queries the signature oracle for $\sigma_{pk}(x, \ell_1, \ell_2)$.

- *Corruptions:* If at any point, $\mathcal{A}$ corrupts a party $P_l \neq P_j$, then $\mathcal{F}$ hands $\mathcal{A}$ the signing-key that is associated with P_l (this is the only secret information that P_l has). On the other hand, if at any point $\mathcal{A}$ corrupts P_j, then $\mathcal{F}$ aborts (and does not succeed in forging).

Throughout the above-described simulation, $\mathcal{F}$ monitors each execution and waits for an execution in which exactly one party is corrupt and the agreement is foiled. If no such execution occurs, then $\mathcal{F}$ aborts. Otherwise, in the first foiled execution, $\mathcal{F}$ checks if the uncorrupted P_j is the general in this execution. If not, then $\mathcal{F}$ aborts (without succeeding in generating a forgery). Otherwise, we have an execution in which P_j is the general and agreement is foiled. In such a case, $\mathcal{F}$ succeeds in generating a forgery as follows.

As we have mentioned, agreement can only be foiled if exactly one party is faulty. Since by assumption P_j is not corrupted, we have that one of the recipients P_1 or P_2 are corrupted; without loss of generality, let P_1 be the corrupted party. (We note that $\mathcal{F}$ plays the roles of both honest parties P_j and P_2 in the simulation.) Now, since the agreement was foiled, we know that P_2 does not output P_j's input value x, which means that it defaulted in Step 5. This can only happen if P_2 received two valid signatures on the label ℓ which it sent P_j in this execution. Now, P_2 clearly received a correct signature m on P_j's input using the label ℓ from P_j itself. (In fact, by the simulation, this signature is generated by $\mathcal{F}$ accessing its signature oracle.) However, in addition, P_2 must have received a valid signature m' from P_1, where m' constitutes P_j's signature on a string that contains label ℓ and a different message x'. With overwhelming probability the label ℓ did not appear in any previous execution, because P_2 is honest and chooses its portion of the label at random. Thus, previously in the simulation, the signing oracle was never queried with a string containing ℓ. Furthermore, by the assumption that $x' \neq x$, the oracle query by $\mathcal{F}$ in this execution was *different* to the string upon which m' is a signature. We conclude that m' is a valid signature on a message, and that $\mathcal{F}$ did not query the signing oracle with this message. Therefore, $\mathcal{F}$ outputs m' and this is a successful forgery.

It remains to analyze the probability that $\mathcal{F}$ succeeds in this forgery. First, it is easy to see that when $\mathcal{F}$ does not abort, the simulation of the sequential executions is *perfect*, and that $\mathcal{A}$'s view in this simulation is identical to a real execution. Furthermore, the probability that P_j is the identity of the (uncorrupted) general in the first foiled agreement equals $1/n$ exactly. The fact that P_j is chosen ahead of time makes no difference because the simulation is perfect. Therefore, the choice of P_j by $\mathcal{F}$ does not make any difference to the behavior of $\mathcal{A}$. We conclude that $\mathcal{F}$ succeeds in forging with probability $1/n$ times the probability that $\mathcal{A}$ succeeds in foiling agreement (which is non-negligible). This contradicts the security of the signature scheme. $\square$

7.4.2 Sequentially Composable $\mathrm{ABG}_{n,n/2}$

Fitzi and Maurer [FM00] present a protocol for the Byzantine Generals problem that tolerates any $t < n/2$ faulty parties. Their protocol is for the information-theoretic and unauthenticated model. However, in addition to the point-to-point network, they assume that every triplet of parties is connected with an ideal (3-party) broadcast channel. As we have shown in Section 7.4.1, given a public-key infrastructure for signature schemes, it is possible to implement secure broadcast among three parties that composes sequentially. Thus, a protocol for $\mathrm{ABG}_{n,n/2}$ is derived by substituting the ideal 3-party broadcast primitive in the protocol of Fitzi and Maurer [FM00] with Protocol 7-3. Since Protocol 7-3 and the protocol of Fitzi and Maurer [FM00] both compose sequentially, the resulting protocol also composes sequentially.

Theorem 7.1.3 *Assume that there exists a signature scheme that is existentially secure against chosen message attacks. Then, there exists a randomized protocol for authenticated Byzantine Generals that tolerates $t < n/2$ faulty parties and composes sequentially any polynomial number of times.*

As we show in Section 7.5, it is possible to execute many copies of an authenticated Byzantine Generals protocol concurrently, by allocating each execution a unique identifier that is common and known to all parties.

The Fitzi-Maurer protocol can be altered in such a way that a unique index is allocated to invocation of the $\mathrm{ABG}_{3,1}$ within protocol. We can therefore run the $\mathrm{ABG}_{3,1}$ protocols in parallel (rather than sequentially), improving the round complexity of the resulting protocol. In particular, our protocol is of the same round complexity as the underlying Fitzi-Maurer protocol. (We stress that the fact that the $\mathrm{ABG}_{3,1}$ subprotocols can be executed in parallel within the $\mathrm{ABG}_{n,n/2}$ protocol *does not* imply that the $\mathrm{ABG}_{n,n/2}$ protocol itself can compose in parallel. Rather, by our impossibility result, we know that it indeed *cannot* be composed in parallel.)

7.4.3 Sequentially Composable $\mathrm{ABG}_{n,t}$ for any t

In this section we describe a protocol for the Byzantine Generals problem for n parties, which can tolerate *any number of faulty parties.* However, this protocol is exponential in the number of participating parties. Therefore, in our setting, the protocol can only be carried out for $n = \log k$ parties (where k is a security parameter). We stress that the fact that the number of parties must be logarithmic in the security parameter is due to two reasons. First, we wish the protocol to run in polynomial time. Second, we use a signature scheme and this is only secure for polynomial-time adversaries, and a polynomial number of signatures.

Our protocol is constructed by presenting a transformation that takes a sequentially composable ABG protocol for $n-1$ parties which tolerates $n-3$ faulty parties, $\mathrm{ABG}_{n-1,n-3}$, and produces a sequentially composable ABG protocol for n parties which tolerates $n-2$ faulty parties, $\mathrm{ABG}_{n,n-2}$. Then, given our protocol for broadcast among three parties which tolerates one faulty party, $\mathrm{ABG}_{3,1}$, we can apply our transformation and obtain $\mathrm{ABG}_{n,n-2}$ for any n.

The idea for the transformation is closely related to the ideas behind the protocol for Byzantine Generals for three parties. The solution for the three-party broadcast assumes two-party broadcast (which is trivial). Using two-party broadcast, agreement on a fresh label can be reached. Having agreed on this label, the two point communications with the General are sufficient. Each party sends its claimed fresh label to the General, and the General includes the two received labels inside any signature that it produces. Our general transformation will work in the same manner. We use the $\mathrm{ABG}_{n-1,n-3}$ protocol to have all parties (apart from the General) agree on a random label. Then, each party privately sends this label to the General, who then includes all labels in its signatures. Thus, we prove:

Theorem 7.4.2. *Assume that there exists a signature scheme that is existentially secure against chosen message attacks, for adversaries running in time poly(k). Then, there exists a Byzantine Generals protocols for $O(\log k)$ parties, that tolerates any number of faulty parties and composes sequentially.*

7.5 Authenticated Byzantine Agreement using Unique Identifiers

In this section we consider an augmentation to the authenticated model in which each execution is assigned a unique and common identifier. We show that in such a model, it is possible to achieve Byzantine Agreement/Generals that composes concurrently, for *any* number of faulty parties. We stress that in the authenticated model itself, it is not possible for the parties to agree on unique and common identifiers, without some external help. This is because agreeing on a common identifier amounts to solving the Byzantine Agreement problem, and we have proven that this cannot be achieved for $t \geq n/3$ when composition is required. Therefore, these identifiers must come from outside the system (and as such, assuming their existence is an augmentation to the authenticated model).

Intuitively, the existence of unique identifiers helps in the authenticated model for the following reason. Recall that our lower bound is based on the ability of the adversary to *borrow* signed messages from one execution to another. Now, if each signature also includes the session identifier, then the honest parties can easily distinguish between messages signed in this execution and messages signed in a different execution. It turns out that this is enough. That is, we give a transformation from almost any Byzantine Agreement protocol based on signature schemes, to a protocol that composes concurrently when unique identifiers exist. By "almost any protocol," we mean that this transformation applies for any protocol that uses the signature scheme for signing and verifying messages only. This is the natural use of the signature scheme and all known protocols indeed work in this way.

More formally, our transformation works as follows. Let Π be a protocol for authenticated Byzantine Agreement. We define a modified protocol $\Pi(id)$ that works as follows:

- Each party is given the identifier id as auxiliary input.
- If a party P_i has an instruction in Π to sign a given message m with its secret key sk_i, then P_i signs upon $id \circ m$ instead (where $\circ$ denotes concatenation).
- If a party P_i has an instruction in Π to verify a given signature σ on a message m with a public key pk_j, then P_i verifies that σ is a valid signature for the message $id \circ m$.

We now state our theorem:

Theorem 7.5.1. *Let Π be a secure protocol for authenticated Byzantine Agreement which uses an existentially unforgeable signature scheme. Furthermore, this scheme is used for generating and verifying signatures only. Let the protocol $\Pi(id)$ be obtained from Π as described above, and let $id_1, \ldots, id_\ell$ be a series of ℓ unique strings. Then, the protocols $\Pi(id_1), \ldots, \Pi(id_\ell)$ all solve the Byzantine Agreement problem, even when run concurrently.*

We conclude by noting that it is not at all clear how such an augmentation to the authenticated model can be achieved in practice. In particular, requiring the on-line participation of a trusted party who assigns identifiers to every execution is clearly impractical. (Furthermore, such a party could just be used to directly implement broadcast.) However, we do note one important scenario where Theorem 7.5.1 can be applied. As we have mentioned, secure protocols often use many invocations of a broadcast primitive. Furthermore, in order to improve round efficiency, in any given round, many broadcasts may be simultaneously executed. The key point here is that *within* the secure protocol, unique identifiers can be allocated to each broadcast (by the protocol designer). Therefore, authenticated Byzantine Agreement can be used. Of course, this does not change the fact that the secure protocol itself will not compose in parallel or concurrently. However, it does mean that its security is guaranteed in the stand-alone setting, and a physical broadcast channel is not necessary.

7.6 Open Problems

Our work leaves open a number of natural questions. First, an unresolved question is whether or not it is possible to construct randomized protocols for authenticated Byzantine Generals that sequentially compose, for *any* n and any number of faulty parties. Second, it is unknown whether or not it is possible to construct a deterministic protocol that terminates in r rounds and sequentially composes ℓ times, for some $2 \leq \ell \leq 2r-1$. Another question that arises from this work is to find a realistic computational model for Byzantine Agreement that *does* allow parallel and concurrent composition for $n/3$ or more faulty parties.

Bibliography

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer Verlag, 2000.

[Ang] Dana Angluin. Lecture notes on the complexity of some problems in number theory. `http://web.mit.edu/6.857/www/handouts/angluin-yale-tr243.ps`.

[AT01] Giuseppe Ateniese and Gene Tsudik. Quasi-efficient revocation of group signatures. `http://eprint.iacr.org/2001/101`, 2001.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 106–115, 2001.

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer-Verlag, 1994.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992.

[BF01] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Verlag, 2001.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of* *the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, Illinois, 2–4 May 1988.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer-Verlag, 1992.

[BGK95] Ernie Brickell, Peter Gemmel, and David Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAMs*, pages 457–466. Association for Computing Machinery, January 1995.

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial time in simulation and extraction. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. Manuscript, to appear in ASIACRYPT2001. Available from `http://crypto.stanford.edu/d̃abo/abstracts/weilsigs.html`, 2001.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer Verlag, 1997.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communication Security*, pages 62–73. Association for Computing Machinery, 1993.

[Bra99] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy.* PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[Bri98] David Brin. *The Transparent Society: Will Technology Force Us to Choose Between Privacy and Freedom?* Perseus Publishing, 1998.

[BS01] Emmanuell Bresson and Jacques Stern. Group signatures with efficient revocation. In Kwangjo Kim, editor, *Proceedings of 4th International Workshop on Practice and Theory in Public Key Cryptography,* *PKC2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 190–206. Springer, 2001.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. Manuscript obtained by personal communication, 2002.

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: non-interactive zero-knowledge based on any trapdoor permutation. *Journal of Cryptology*, 9(3):149–166, 1996.

[Cam97] Jan Camenisch. Efficient and generalized group signatures. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer Verlag, 1997.

[Cam98] Jan Leonhard Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* PhD thesis, ETH Zürich, 1998. Diss. ETH No. 12520, Hartung Gorre Verlag, Konstanz.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CE87] David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 118–167. Springer-Verlag, 1987.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer Verlag, 2001.

[CFT98] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy come – easy go divisible cash. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer Verlag, 1998.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[Che95] Lidong Chen. Access with pseudonyms. In E. Dawson ann J. Golić, editor, *Cryptography: Policy and Algorithms*, volume 1029 of *Lecture Notes in Computer Science*, pages 232–243. Springer Verlag, 1995.

[CKOS01] Giovanni Di Crescenzo, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Efficient and non-interactive non-malleable commitment. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494. Springer Verlag, 2001.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires omega(log n) rounds. In *Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 570–579, 2001.

[CL01] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Verlag, 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, Lecture Notes in Computer Science, pages 61–76. Springer Verlag, 2002.

[CM99a] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer Verlag, 1999.

[CM99b] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 413–430. Springer Verlag, 1999.

[CP95] Lidong Chen and Torben Pryds Pedersen. New group signature schemes. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1995.

[Cra97] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocol.* PhD thesis, University of Amsterdam, 1997.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer Verlag, 1997.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–52. ACM press, nov 1999.

[CvH91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[Dam90] Ivan Bjerre Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer Verlag, 1990.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[Dam02] Ivan Damgård. On σ-protocols. Available at `http://www.daimi.au.dk/~ivan/Sigma.ps`, 2002.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, April 2000.

[DF01] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. `http://eprint.iacr.org/2001`, 2001.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, Nov. 1976.

[DLN96] Cynthia Dwork, Jeffrey Lotspiech, and Moni Naor. Digital signets: Self-enforcing protection of digital information. In *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*, 1996.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92. Springer Verlag, 2000.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, 1998.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[FM00] Matthias Fitzi and Ueli Maurer. From partial consistency to global broadcast. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2000.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 1997.

[FO98] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 32–46. Springer Verlag, 1998.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solution to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1987.

[Gar00] Simson Garfinkel. *Database Nation: The death of privacy in the 21st century.* O'Reilly & Associates, Inc., 2000.

[GB] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. `http://www.cs.ucsd.edu/users/mihir/papers/gb.html`.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer Verlag, 1999.

[GK96] Oded Goldreich and Hugo Krawczyk. n the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[GLR95] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 267–287, 1995.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 291–304, 1985.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal of Computing*, 18(1):186–208, February 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a method of cryptographic protocol design. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 174–187. IEEE Computer Society Press, 1986.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 1987.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, pages 228–244. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[Gol98] Oded Goldreich. Secure multi-party computation. Manuscript. Available from `http:www.wisdom.weizmann.ac.il/~oded/pp.html`, 1998.

[Gol01] Oded Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001.

[Gol02] Oded Goldreich. Concurrent zero-knowledge with timing revisited. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 332–340, 2002.

[GPR98] Oded Goldreich, Birgit Pfitzman, and Ronald Rivest. Self-delegation with controlled propagation — or — what if you lose your laptop. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1642 of *Lecture Notes in Computer Science*, pages 153–168, Berlin, 1998. Springer Verlag.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28(4):1364–1396, 1999.

[JL00] Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: introducing cocurrency, removing erasures. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 190–206. Springer Verlag, 2000.

[JN01] Antoine Joux and Kim Nguyen. Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Manuscript. Available from `http://eprint.iacr.org`, 2001.

[Jou00] Antoine Joux. A one-round protocol for tripartite Diffie-Hellman. In *Proceedings of the ANTS-IV conference*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2000.

[KLL01] Hyun-Jeong Kim, Jong In Lim, and Dong Hoon Lee. Efficient and secure member deletion in group signature schemes. In D. Won, editor, *ICISC 2000*, number 2015 in Lecture Notes in Computer Science, pages 150–161. Springer Verlag, 2001.

[KP98] Joe Kilian and Erez Petrank. Identity escrow. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1642 of *Lecture Notes in Computer Science*, pages 169–185, Berlin, 1998. Springer Verlag.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report Technical Report CSL-98, SRI International, October 1979.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 171–189. Springer Verlag, 2001.

[Lip01] Helger Lipmaa. Statistical zero-knowledge proofs from diophantine equations. Manuscript. Available from `http://eprint.iacr.org/2001/086`, 2001.

[LLM$^+$01] Moses Liskov, Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Adam Smith. Mutually independent commitments. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 385–401. Springer Verlag, 2001.

[LLR02] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 514–523, 2002.

[LRSW99] Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[Lys99] Anna Lysyanskaya. Pseudonym systems. Master's thesis, MIT, Cambridge, Massachusetts, 1999.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, Lecture Notes in Computer Science, pages 597–612. Springer Verlag, 2002.

[Mer90] Ralph Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer Verlag Berlin, 1990.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[MR92] Silvio Micali and Phillip Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1992.

[MR01] Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 542–565. Springer Verlag, 2001.

[MR02] Silvio Micali and Ronald L. Rivest. Micropayments revisited. In Bart Preneel, editor, *Proceedings of the Cryptographer's Track at the RSA Conference*, volume 2271 of *Lecture Notes in Computer Science*, pages 149–163. Springer Verlag, 2002.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130. IEEE Computer Society Press, 1999.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):51–158, 1991.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, Washington, 15–17 May 1989. ACM.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PW96] Birgit Pfitzmann and Michael Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t >= n/3$. Technical Report Research Report RZ 2882 (#90830), IBM Research Division, 1996.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254. ACM press, nov 2000.

[Rab79] Michael Rabin. Digitalized signatures as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[Rey01] Leonid Reyzin. *Zero-Knowledge with Public Keys.* PhD thesis, Massachusetts Institute of Technology, June 2001.

[RK99] Random Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer Verlag, 1999.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 387–394, Baltimore, Maryland, 1990. ACM.

[RS86] Michael O. Rabin and Jeffrey O. Shallit. Randomized algorithms in number theory. *Communications in pure and applied mathematics*, 39:239–256, 1986.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch91] Claus P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. In *ACM Transaction on Computer Systems*, volume 1, pages 38–44, 1983.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Verlag, 1985.

[Sil86] Joseph Silverman. *The arithmetic of elliptic curve.* Springer-Verlag, 1986.

[Son01] Dawn Xiaodong Song. Practical forward secure group signature schemes. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 225–234. ACM press, nov 2001.

[SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer Verlag, 1995.

[ST01] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer Verlag, 2001.

[Sud] Madhu Sudan. Algorithmic introduction to coding theory. MIT course taught in Fall 2001. Lecture notes available from `http://theory.lcs.mit.edu/~madhu/FT01/`.

[TW87] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science*, pages 472–482, Los Angeles, California, 12–14 October 1987. IEEE.

[Ver01] Eric Verheul. Self-blindable credential certificates from the weil pairing. Manuscript, to appear in ASIACRYPT2001, 2001.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

[Yao86] Andrew C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

4935-92