Modular Design of Secure yet Practical Cryptographic Protocols

ISBN 90–74795–64–1

Modular Design of Secure yet Practical Cryptographic Protocols

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam, op gezag van
de Rector Magnificus prof. dr. J. J. M. Franse
ten overstaan van een door het college van
dekanen ingestelde commissie in het openbaar
te verdedigen in de Aula der Universiteit
op vrijdag 31 januari 1997 te 11.30 uur

door

Ronald John Fitzgerald Cramer

geboren te Haarlem.

Acknowledgements

Paul Vitányi, my promotor, is kindly acknowledged for his inspiring criticism and guidance during the preparation of the final manuscript. Consulting him regarding matters of a general scientific nature, has always been beneficial, and joyful too.

Ivan Damgård, my co-promotor, has given me continuous support and advice throughout the last three years. Most importantly, he has generously shared his knowledge and insights with me during these years of close cooperation. Much of the research reported on in this work is the result of our efforts to develop a theory of cryptographically secure *and* practical protocols. Ivan, it's a pleasure to thank you for your contributions to my development as a researcher.

David Chaum and Jan-Hendrik Evertse are responsible for triggering my interest into cryptology. I am grateful to David for inviting me to CWI [1] in 1992 and for putting me on the right track in more than one way.

I gratefully acknowledge CWI (Martin Kersten) for providing the fine conditions under which I have been able to carry out the research which led to this thesis. Many thanks to the members of the Ph.D.-committee: they contributed a lot of useful suggestions regarding the presentation of this work. And so did Berry Schoenmakers, who read substantial parts of the manuscript. With great satisfaction I will look back on joint work with my co-authors Ivan Damgård, Matthew Franklin, Torben Pedersen, Berry

[1]CWI is the Dutch National Research Institute for Mathematics and Computer Science.

Schoenmakers, and Moti Yung.

My colleagues Annette Bleeker and Louis Salvail have always responded with humour whenever I came to deliver a "fresh joke" that had occurred to me while travelling from home to CWI. I take the opportunity to express my honest and deepest anti-thanks to Nederlandse Spoorwegen (Dutch Rail) and to that Propaganda Institution of theirs, that they dare call "Klanten Service" (Customer Service).

The following persons are entitled to receive the cryptic acknowledgements, to confirm that we share some private keys. Berry Schoenmakers, Torben Pedersen, Matthew Franklin and Juan Garay, without your friendship and your cooperation, things would have been different. Some of the results in this thesis might not have been conceived, for instance. Or I might have been deprived of an excellent friend, office mate and travel companion (Berry), of a dry humorist (hanging out at Salman's with Torben), and of the symbolism of Bagel Bob's (Matt). Juan Garay is granted the Tapas-Man-of-the-Year Award. Marc Berns and Hans van der Kolk, we gave that coffee-machine on the 2nd floor a hard time, and we enjoyed it. Berry's careful and critical scrutiny (almost always available) has many times provided a very useful testing-ground for new ideas.

To J.B.W. Cramer-de Jongh, I wish you were here to see it. To Marta and to my family, particularly my parents Leonie and Ronald: your love and support has been crucial to me, and will continue to be so. On the way to new and exciting research!

Baarn, November 1996.

Contents

1 Introduction **1**

1.1 History 1

1.2 Research Goals 9

1.3 Overview 14

2 Σ-Protocols **19**

2.1 Model and Definitions 19

2.2 Requirements 22

2.2.1 Completeness 22

2.2.2 Special Soundness and Collision Intractability 22

2.2.3 Zero-Knowledge 24

2.2.4 Witness Hiding and Witness Indistinguishability . . 25

2.2.5 Proofs of Knowledge 26

2.2.6 Interactive Proofs 27

2.2.7 Signature Protocols 28

2.3 Examples 29

2.3.1 Introduction 29

2.3.2 Based on Discrete Logarithms 29

2.3.3 Based on RSA 31

2.4 Basic Theory of Σ-protocols 33

2.4.1 Collision Intractability 34

2.4.2 Approximating Special Soundness 37

2.4.3 Simulation Properties 40

2.4.4 Manipulating the Challenge Length 41

2.4.5 Trade-offs 43
2.5 Existence 46
2.5.1 One-Way Group Homomorphisms 47
2.5.2 Special One-Way Group Isomorphisms 48
2.5.3 Claw-Free Pairs of Trapdoor Permutations 53
2.5.4 Special Claw-Free Pairs of Trapdoor Permutations . 56

3 Partial Proofs and Applications **59**
3.1 Introduction 59
3.2 Proofs of Partial Knowledge 62
3.2.1 Introduction 62
3.2.2 Monotone functions 63
3.2.3 Secret Sharing 64
3.2.4 Main Results 69
3.2.5 Application to Identification and Signatures 77
3.3 Linear General ZK Proofs 77
3.3.1 Introduction 77
3.3.2 Bitcommitments and Linear Proofs of Contents . . . 81
3.3.3 General Approach 83
3.3.4 Zero Knowledge Interactive Proofs for NP 87
3.3.5 Interactive Arguments for NP 88
3.3.6 Existence of Bitcommitment Schemes 89
3.4 Monotone Function Closure of Zero Knowledge 95
3.4.1 Introduction 95
3.4.2 Notation and Definitions 97
3.4.3 Overview 98
3.4.4 The Closure Property for L 100
3.4.5 The Closure Property for L' 103
3.4.6 Extensions 105
3.5 Linear Secret Ballot Elections 105
3.5.1 Introduction 105
3.5.2 Overview of the Approach 106
3.5.3 Related Work 107
3.5.4 Cryptographic Primitives 108
3.5.5 Secret Ballot Election Scheme 110
3.5.6 Conclusion 114
3.6 Efficiently Combatting Man-in-the-Middle Attacks 114
3.6.1 Introduction 114
3.6.2 Relation to Identification Schemes 118
3.6.3 Main Result 118
3.6.4 Implementation 119
3.6.5 Security Analysis 119
3.6.6 Example 121
3.6.7 Applications 121

4 Secure and Efficient Digital Signatures **123**

4.1 Introduction 123
4.2 Our Contributions 126
4.3 Definitions 128
4.4 Shorter GMR-signatures 129
4.4.1 Notation 130
4.4.2 The Goldwasser-Micali-Rivest scheme 130
4.4.3 The Modification 132
4.4.4 Preliminaries 133
4.4.5 Description of the Signature Scheme 134
4.4.6 Proof of Security 136
4.4.7 Alternative Descriptions 139
4.5 Secure Signatures based on Interactive Protocols 140
4.5.1 Introduction 140
4.5.2 Notation 141
4.5.3 Overview of Our Method 142
4.5.4 Linear Signature Protocols 143
4.5.5 Existence of Linear Signature Protocols 144
4.5.6 The Algorithm DFS 145
4.5.7 Main Result 145
4.5.8 Proof of Security 147
4.5.9 Example 149
4.6 Security Amplifications for Signatures 151
4.6.1 The Transformation 152
4.6.2 Minimality of the Construction 154
4.7 Secure and Practical RSA-based Signatures 154
4.7.1 Basic Ideas 155
4.7.2 Description of the Scheme 156
4.7.3 Generating the List of Primes L 159
4.7.4 Performance of the Scheme 159
4.7.5 Proof of Security 160
4.7.6 Optimizations 163

Nederlandse Samenvatting **165**

Curriculum Vitae **173**

Bibliography **175**

Index **185**

1 Introduction

1.1 History

Traditionally, cryptography has been the practice of "secret writing". Although the name is derived from the Greek $\kappa\rho\upsilon\pi\tau o\sigma$ (= hidden) and $\gamma\rho\alpha\phi\epsilon\iota\nu$ (= to write), Julius Caesar is often presented as an early inventor and user of secret codes. To establish a means of private communication (a *private channel*) with his generals in the field, Julius Caesar would first encrypt his orders according to some fixed permutation of the alphabet, known to his generals only. The encrypted message can now safely be communicated by a courier, for instance. Upon receiving the encrypted orders, the plain text message was then recovered by applying the permutation in the reverse direction. An enemy intercepting the encrypted message after capturing the courier for instance, should have great difficulties recovering the actual message from that piece of garbled information. It seems that Julius Caesar had a preference for substituting each letter by its third successor in the alphabet. For instance, he substitutes the letter "u" for "r" and "a" for "x". It is clear that a resurrected Julius Caesar, might he wish to establish a private channel, would have to apply a different substitution. However, encryption methods employing a secret permutation of the alphabet have since long been known to be susceptible to various kinds of letter frequency analyses.

In 1926, G. S. Vernam proposed an unbreakable encryption method, commonly known as the *one-time pad.* A sender and receiver agree on a sequence of, say, t secret random bits, the *secret key.* Let $s \in \{0,1\}^t$ denote this key. The sender computes the encryption $c \in \{0,1\}^t$ of a plain text message $m \in \{0,1\}^t$ as $c = m \oplus s$, where '$\oplus$' is bitwise addition modulo 2. The receiver recovers the plain text m by computing $c \oplus s$. The secret key is to be used only once. This encryption method or "cipher" is unbreakable in the sense that an eavesdropper intercepting the encrypted message gets no information at all about the plain text, not even when having access to unlimited computational power. Indeed, for each possible plain text m' there is a unique s' such that $m' \oplus s'$ is equal to the intercepted encryption c. Since the secret key is chosen uniformly at random all s' are equally likely from the interceptors point of view. A drawback of this method, rendering it impractical for most applications, is that the length of the secret key is equal to the size of the plain text and that it can be used only once.

The Second World War marks the beginning of systematic, mathematical design and analysis of practical ciphers; ciphers where a single secret key can be used to encrypt a large number of plain text messages. Informally, a cipher consists of two computational procedures; the encryption- and decryption algorithms E and D, respectively. If two agents have access to a shared secret key s, they can establish a private channel as follows. The sender computes the encryption c by executing the algorithm E on input of the plain text message m and the secret key s. Upon receiving the encryption c, the receiver recovers the plain text m by executing the algorithm D on input c and the secret key s. The security of such ciphers is based on the supposed computational difficulty of recovering the plain text from a given cipher text without knowledge of the secret key. Cryptographic methods were extensively used by all battling parties to secure their communication from eavesdropping by the enemy. And every so often each of these parties made great efforts to break or cryptanalyze the ciphers used by the enemy. A major war-time event in this respect was the successful cryptanalysis of the German cipher "Enigma" (which is also used to refer to the machine that implemented the cipher) by the Allied Forces. The machine could be used for both encryption and decryption. For obvious reasons of security, the machine did not use one fixed secret key for all communication within the German forces. The machine could handle any suitable choice of the secret key when it was given as input together with the text it had to encrypt or decrypt. Analysis of captured Enigma-machines made clear how the Enigma-cipher computes the cipher text from a given plain text and a given secret key. Thus, the algorithm that the Enigma-machine executes was unveiled. Thorough study of the mathematical properties of this algorithm eventually enabled[1] to read (parts of) actual messages from intercepted encryptions.

[1]Earlier Enigma versions were broken by the Polish cryptanalyst Marian Rejewski. Having the details at their disposal eventually, this contributed substantially to the success of the cryptanalytic efforts of the Allied Forces.

A popular misconception about cryptographic systems is that keeping their workings secret strengthens these systems. The example above indicates that it is unreasonable to assume that the enemy does not get hold of the algorithm that the cipher prescribes. The Dutchman A. Kerkhoffs (1835–1903) stated that the security of a cipher must reside exclusively in the computational difficulty (or impossibility) of decrypting without knowledge of the secret key. Kerkhoffs' principle is that we must always assume that the cryptanalyst knows the algorithm of the cipher.

Until shortly after the Second World War, the use and study of cryptographic methods was restricted to national governments and the military. With the paper "Communication Theory of Secrecy Systems" by C. Shannon [110], *cryptology* (cryptography and cryptanalysis) entered the public domain and started to emerge as a science. Building on the Information Theory he developed in [109], he showed for instance that a cipher that achieves "perfect secrecy" (the security is independent of the enemies' computational resources) requires a one-time secret key that is at least as long as the plain text. Thus the one-time pad, impractical for most applications, is an optimal perfect cipher. This motivates the interest in cryptographic systems that can be used efficiently but that offer a relaxed level of security. Informally speaking, a computationally or practically secure cryptographic system is one where any conceivable cryptanalytic attempt requires an amount of computation that the enemy cannot perform in reasonable time.

The advent of the computer era inspired cryptologic research in at least two ways. First, the availability of powerful computing devices allows for the design of cryptographic methods employing more involved mathematical techniques and a much larger diversity of secret keys. Secondly, applications of cryptography now came within the scope of private sectors in society, such as business and industry. Banks were among the early adopters of cryptographic systems.

With their paper "New Directions in Cryptography" [56] in 1976, W. Diffie and M. Hellman caused nothing less than a revolution in cryptology. They set forth the notion of *public key cryptography*. Originally motivated by the key distribution problem for secret key ciphers (note that the distribution of the shared secret keys among participant using such a cipher, is a security problem in its self, since the method by which this is done, might itself be vulnerable to eavesdropping), they not only propose *public key ciphers* as a solution but also extend the domain of cryptology beyond that of private communication. They introduce *digital signatures*, the electronic equivalent of traditional, handwritten signatures.

The basic idea behind public key encryption is as follows. Suppose we have a network of users, connected by insecure channels, that may wish to communicate privately with each other. To set up a public key cipher each of them *locally* generates a pair of keys (pk, sk), the public- and private key. The way to this is specified by the public key cipher. Next, the public keys pk are posted in some reliable (or "trusted") publicly accessible directory, similar to a phone book. It is important that the information stored in this directory is authentic: if one looks up the public key for some given user, one must be sure that the public key one retrieves indeed belongs to that user. And it is in this sense that we assume that the public directory can be trusted. The private keys sk are stored locally and kept secret from each of the other users. An important difference with the secret key ciphers discussed earlier, is that with public key encryption no secret information needs to be shared between users. Thus, at the lesser cost of setting up a public directory, the key distribution problem is circumvented. The public key cipher can be thought of as consisting of three computational procedures or algorithms: G, E and D, the key generation-, encryption- and decryption-algorithms. The description of these algorithms is public knowledge. Each user runs the key generation algorithm to get a key pair (pk, sk). Since G selects these pairs at random from a very large (*exponential size*) set of possible pairs, different users obtain different key pairs. If any user B wishes to privately communicate a plain text message m to user A, user B first looks up the *public key* pk_A corresponding to user A. The encryption c of m is computed by executing the algorithm E on input m *and* the public key pk_A. Upon receiving c over the insecure channel that connects A and B, user A recovers the plain text m from the cipher text c by executing the algorithm D on input of c *and* A's private key sk_A.

Diffie and Hellman proposed a class of functions, *trapdoor one-way functions*, that support the implementation of public key ciphers. Informally speaking, a trapdoor one-way function f is a function which is easy (efficiently) to compute on each input taken from the domain of the function, but that is virtually impossible (infeasible) to invert on an arbitrary element from its range (*one-wayness*), *unless* one is given a short piece of advice s; the *trapdoor*. Thus, when given s, the inverse function f^{-1} of f can also be computed efficiently. Functions that do not necessarily have the trapdoor property but do satisfy the one-way property, are simply called *one-way functions*.

It is now easy to define a public key encryption scheme based on such functions. If we assume that we are given a large collection of trapdoor one-way *permutations* (functions with a unique correspondence between the elements of the domain and range) and that we can efficiently select random elements f from this collection together with a corresponding trapdoor s, each user just chooses an arbitrary element f with corresponding trapdoor s and publishes a description of f as the public key pk. The private key sk is equal to s. Note that user A knows how to compute the inverse f_A^{-1} of f_A, since A knows the trapdoor s. Any other user, by our assumptions on the properties of these functions, can only compute f_A efficiently but not f_A^{-1}. To encrypt m, c is computed as $c = f_A(m)$. A recovers m by computing $f_A^{-1}(c) = f_A^{-1}(f_A(m)) = m$. Note that we used here that f_A is injective.

Instead of providing secrecy, public key *digital signatures* establish the *authenticity* of a digital document. Suppose sender A communicates a message m to receiver B. How does B acquire certainty as to the originator of the message? To this end A provides B with an additional piece of information, depending on the message m and A's public key: a digital signature σ on the message m. Upon receiving the message and the signature, B performs a verification procedure to check whether the given signature is consistent with the message and A's public key. Only if so, B concludes that the message was indeed originated by A. More formally, a digital signature scheme consists of three publicly known algorithms G, S and V. Any user in the network can invoke the key-generation algorithm G to generate a key pair (pk, sk) consisting of a public key and a private key. As with public key encryption, the public keys pk of the users are stored in a trusted public directory. To compute a signature σ on a message m, sender A executes the signing-algorithm algorithm S on input m and A's private key sk_A. Receiver B checks the validity of the signature by verifying whether algorithm V return 'OK' on input of the signature σ, the message m and A's public key pk_A. If B wants to prove to a third party C that A signed the message m, B merely has to supply C with the signature σ, as C can as well as B check the validity on its own by invoking the algorithm V. Thus, as a certificate for authenticity, a digital signature is transferable. Just as with public key encryption, Diffie and Hellman showed how to utilize trapdoor one-way functions for the purpose of digital signatures. Today, digital signatures play a vital role in such areas as electronic commerce and banking.

Apart from their key-exchange protocol based on exponentiation in finite fields, which is a procedure by means of which two users can compute a common private key (which can henceforth be used to initiate a secret key cipher for instance) from their respective public keys, Diffie and Hellman suggested no explicit implementations of their public key systems. In 1978, Rivest, Shamir and Adleman [100] proposed a family of functions, commonly known as the RSA-functions, that they conjectured to be trapdoor one-way functions. Based on the widely recognized difficulty of factoring large composite integer numbers into their prime factors, they suggested exponentiation modulo such composites (*RSA-moduli*) as a suitable trapdoor one-way function. A public key consists of an RSA-modulus and an exponent. The trapdoor information consists of the factorization of the large integer modulus. This may seem contradictory, since how should one compute the trapdoor if factorization is such a difficult problem? The answer may be obtained by making a comparison with the work of a locksmith. It seems unreasonable for a locksmith to first construct a very strong and complicated lock and then figure out which key will open it. One may assume that in practice the locksmith applies a bit of "reverse-engineering". The trick with generating arbitrary RSA-functions is to pick two large prime factors first, each consisting of say 100 decimal digits, and then defining the RSA-modulus as the product of these primes. This method is based on the observations that it is relatively easy to select random large primes and to multiply them together. Since exponentiation modulo large integers is a computationally easy task as well, an RSA-function can be computed efficiently on all inputs. It is also easy to show that knowledge of the factorization enables efficient inversion of an RSA-function. It is still an open problem to prove that being able to invert an RSA-function is equivalent to knowing the factorization of the corresponding RSA-modulus, although this is widely believed to be true. Computing *discrete logarithms* (logarithms in finite, cyclic groups) is another important computationally difficult problem which enables cryptologic researchers to design secure cryptographic schemes. Especially discrete logarithms in the multiplicative group of a large and randomly chosen finite field are considered infeasible to compute. During recent years smartcards (small microprocessors placed on a credit-card-sized piece of plastic) dedicated to large integer arithmetic have become available that allow for the efficient implementation of public key methods.

The security of known public key methods predominantly relies on such mathematical problems as integer factorization and discrete logarithms. These problems can in theory be solved, if one is given *enough time and computing power.* Computational number theorists have been inspired since the introduction of RSA to devise fast methods for factorization. It must be noted that they have made progress unforeseen by the inventors of RSA. Although no efficient factorization method is known to date (nor a proof that no such efficient method exists), Lenstra *et al.* [41] have recently been able to factor an RSA-modulus consisting of 130 decimal digits. This modulus was posted as a factorization challenge by Rivest, Shamir and Adleman as part of their "RSA-challenge list", a sequence of RSA-moduli with increasing size, and back in 1978 they assumed the factorization RSA-130 (as it is called) would never see the light of day, unless a major breakthrough in the art of factoring was achieved.

The work of Lenstra et a˙ has elevated the notion "using all available computing power in the world" beyond mere imagination. Using the idle time of a host of computers spread across the Internet, whose use has become widespread over recent years, they implemented the most successful factorization method to date, the Number Field Sieve (NFS)[86]. Their result shows that the use of RSA-moduli with at least, say, 200 decimal digits, is necessary to ensure strong security of cryptographic systems based on the difficulty of factoring integers. Thus, the combination of massive parallelism and powerful mathematical techniques such as the NFS, invalidated the early claims about the difficulty of factoring integers of say, up to 150 decimal digits. We now briefly explain the asymptotic characteristics of methods like NFS and indicate how cryptography responds to such advances in the art of factoring.

The efficiency or running time of an algorithm is measured in terms of the amount of computation it requires. The *asymptotic* efficiency of an algorithm is obtained by viewing the amount of computation required as a function of the length of the input to the algorithm. An algorithm is called *efficient* or *polynomial time* if the running time increases at most moderately when the length of the input is increased, that is, if the running time is bounded above by some polynomial in the length of the input. Especially computations whose running time functions show an explosive growth, such as (sub)exponential functions, are considered to be infeasible to perform. For factorization algorithms such as NFS, the running-time function is sub-exponential and and it seems that with current methods and their implementations, factoring integers of much more than 200 decimal digits is virtually infeasible. It is hard, though, to predict future developments in computational number theory. Supported by progress in hardware developments, present implementations of cryptographic schemes relying on the difficulty of factoring take these developments into account by increasing the length of the moduli they work with correspondingly.

At least in theory, the model of *quantum computation*, has been shown to be powerful enough to accommodate factoring or computing discrete logarithms in polynomial time [111]. At this point it is unclear whether a *quantum computer*, whose workings should be based on principles taken from quantum physics, will ever exist. Currently, it is said that profound technological problems have to be overcome first, before such a machine would see the light of day. Skeptics state that external influences will necessarily cause such machines to be instable, slowing down considerably its computational power predicted by the theory.

In the Eighties, research into public key cryptology led to a host of new and exciting notions, such as zero knowledge proofs, pseudo-randomness, multi-party-computations and anonymous transactions. Today, these concepts, as well as public key encryption and digital signatures, fulfill vital roles in most real-life applications of cryptography, and research into the practical and theoretical aspects of these concepts continues. We briefly outline their nature. Suppose one has found a proof to some interesting theorem. How does one convince a sceptical opponent of the veracity of the theorem without giving away the proof? Solving such questions is what *zero knowledge proofs* [68] are concerned with. A *cryptographic protocol* is an interactive "game" between its players. The protocol consists of a sequence of computations to be performed by the players and corresponding exchanges of messages. It turns out to be possible to design cryptographic protocols that enable a *prover* to demonstrate a theorem such that, no matter how the *verifier* deviates from the actions dictated by the protocol, the verifier gets no information beyond the fact that the theorem is true. In particular this means that the verifier cannot misuse the game with the prover to extract the proof of the theorem and prove the theorem to a third party later. If the theorem is false, the prover will not succeed in convincing the verifier. Zero knowledge proofs and the related witness hiding proofs have been successfully applied to secure log-in procedures, or more generally, to identification schemes. As an example, there exist zero knowledge protocols that allow a prover to demonstrate knowledge of the factorization of an RSA-modulus. Suppose that each of the users in some network privately generates a random RSA-modulus, as described before. Keeping the factors private, they each publish the modulus as a public key. Let n_A be user A's public key, and let's consider the case where A wishes to identify itself to B. To this end, they just execute the claimed zero knowledge protocol. From the point of view of B this clearly shows that the prover knows the factorization of n_A. But how can B conclude that A is the prover? Couldn't it be the case that some other user managed to factor n_A, possibly through extracting helpful information from A in a past execution of the proof? The answer is given by observing that, if the modulus is large enough, factoring n_A from scratch is infeasible. But, by the zero knowledge property of the proof, A gives away no knowledge about the factorization of n_A. Sophisticated zero knowledge techniques allow a prover to demonstrate knowledge of witnesses for membership of any NP-language (an NP-language is a set whose members have a short proof of membership, called a witness, and the zero knowledge proof concerns a theorem of the form $x \in L$, x is a given element and L is an NP-language). This implies that there exist zero knowledge techniques for such general problems as showing possession of a pre-image under some one-way function. It should now be clear, at least intuitively, that zero knowledge protocols are a powerful primitive. As such, they play an important role in more complex cryptographic schemes.

Another intriguing notion is that of *pseudo-randomness* (see for instance [108]). There, the idea is to stretch a short random string of bits into a much longer *pseudo-random* bitstring with the property that any *computationally bounded* observer (a party whose computational powers are restricted to polynomial time operations) witnessing the pseudo-random string cannot distinguish it from a truly random string. It has been shown that pseudo-random *generators* can be constructed from one-way functions. Yet another area is that of *secret sharing* [107]. Suppose one wishes to "distribute" a piece of information among a group of people in such a way that only certain subsets ("qualified subsets") are able to read it by pooling their information. There exist various methods, depending on the constraints placed by, for instance, the qualified subsets, for splitting up the information in pieces ("shares") which are subsequently given to the "share-holders". Using the method given in [107], it is possible to ensure that, for instance, only the shares of at least half of the share-holders are necessary and sufficient to reconstruct the original information. *Multiparty-computations* [73] deal with the general setting where a number of mutually mistrusting parties wish to jointly compute a given function, while their respective inputs should remain private. A practical example is that of secret ballot election schemes. There the players are individual voters and government agents. Each of the voters submits a vote, say "yes" or "no", and they wish to jointly compute the number of "yes"-votes, without revealing individual votes. An important example of *anonymous transactions* [28, 35] are electronic transaction systems, such as digital payments, where the identity of individual participants is protected by cryptographic means. The cryptographic notions presented above might seem diverse, and indeed they are. But they obey one principle, which may also be used to define modern cryptography in general (Goldreich [75]): cryptography is the science that is concerned with any problem in which one wishes to limit the effects of dishonest users.

1.2 Research Goals

Since 1976, theoretical efforts in the area of public key cryptographic research have resulted in a tremendous progress by developing such concepts as public key encryption, digital signatures, zero knowledge, multi-party computations, pseudo-randomness, and so on. Enhancements and variations of these concepts and weakening complexity assumptions under which these concepts can be realized, as well as achieving efficiency improvements (storage-, communication-, round-complexity, for instance) are key driving forces of the theoretical research in these areas.

For example, after the notion of digital signatures was introduced by Diffie and Hellman (1976) and an implementation (RSA) based on the difficulty of factoring integers was suggested by Rivest, Shamir and Adleman (1978), Goldwasser, Micali and Rivest (1984) were first to present a rigorous treatment of digital signatures and formalized the notion of security for signature schemes. The signature scheme they presented was the first to satisfy their natural security requirements, under reasonable assumptions.

Taking the developments in the theory of signature schemes as an example, we now clarify some important aspects of theoretical public key cryptography. Before doing so, we introduce some useful terminology. To facilitate exact descriptions and analyses of cryptographic schemes, the players or parties in such schemes, as well as their enemies, are usually defined as *Turing Machines.* A Turing machine can be seen as an abstraction of computers as they are known today. Briefly, a Turing machine is said to be *polynomial time* or *efficient*, if the total number of operations it performs is bounded by some polynomial in the length of the input. Computational tasks which cannot be performed by a polynomial time Turing machine are called *infeasible* or *intractable.* A Turing machine is called *interactive* if it is furnished with a communication link enabling the exchange of messages with other such machines. Turing machines having access to random bits are called *probabilistic*. An interactive Turing machine is allowed to use these random bits in any of its computations.

Giving an exact definition of security is an important step when designing or analyzing any cryptographic scheme. Such definitions usually take into account some enemy or attacker furnished with a well-defined amount of *computational resources*. Under some given *flexibility* in carrying out the attack, the enemies' goal is to *break* the cryptographic scheme under consideration. In the case of digital signatures, the computational resources of the enemy are that of a polynomial time probabilistic Turing machine. The enemies' flexibility is captured by providing a real signer as a *black-box* and allowing the enemy to query the black-box on any (polynomially bounded) sequence of input messages of the enemies' choice. The black-box signer, whose computations, memory and internal coin tosses are "invisible" from the point of view of the enemy, returns a valid signature for any of the messages. The enemies' goal is to compute, in polynomial time, a valid signature on some message, whose signature was not requested from the black-box. This is called an "existential forgery". Schemes resisting an attacker of this kind, are called "not even existentially forgeable under adaptive chosen message attacks".

Goldwasser *et al.* also proposed a realization of a digital signature scheme achieving this level of security, based on collections of so-called claw-free pairs of trapdoor permutations. Informally, these are pairs (f_0, f_1) of trapdoor one-way permutations with the extra property that for each pair it is infeasible to find two elements x and y such that $f_0(x) = f_1(y)$ (a "claw"), although each such pair is known to permute the same set. In theoretical cryptography, it is a common practice to design schemes from, for instance, collections of *abstract* functions: functions whose description is given only in terms of their (cryptographic) properties. Assuming their existence, the design then consists of suitably "composing" new functions and protocols from these abstract functions. Such a cryptographic scheme becomes *concrete* once the functions on which it is based, are given in such a way that they can actually be computed. Usually, the required cryptographic properties are shown to hold under some *specific* and seemingly plausible intractability assumption.

Once an appropriate security definition of the scheme at hand has been formulated, the mathematical *proof of security* then consist of showing that security as dictated by the definition is implied by the cryptographic properties of the underlying functions. These cryptographic properties are usually stated by assuming the inability to perform some particular computational task in polynomial time, such as computing claws as above or inverting a one-way function. Mathematically speaking, the proof of security is carried out by means of a *reduction*: assuming that a successful attacker exists, one then shows how such an attacker can be "abused", after some modifications, to successfully attack the cryptographic properties of the underlying functions. Since the enemy is given only polynomial time computational resources, this contradicts the assumptions and the security claims follow. Please note that these steps may be viewed as a formalization of *cryptanalysis*. For instance, Goldwasser, Micali and Rivest construct their signatures scheme from claw-free trapdoor permutations and show that if a successful attacker against their scheme exists, it can be turned efficiently into an efficient method for computing claws. Concretely, if one implements the Goldwasser-Micali-Rivest signature scheme using the factoring assumption and if one uses moduli with, say, 200 decimal digits, a theorem concerning the security of the scheme asserts that any algorithm that breaks the signature scheme in some given amount of time can be converted into another algorithm that factors integers also consisting of 200 decimal digits in roughly the same amount of time and using similar computational resources (since the security of this scheme is directly related to the length of such moduli, the chosen length is called a *security parameter*). Thus, if factoring such integers is "undoable", so must be breaking the signature scheme!

Finally, theoretical results in public key cryptology usually consider an *implementation* of the given scheme, under one or more *concrete intractability assumptions* (IA's), such as the assumptions that factoring integers or computing discrete logarithms are intractable. This means that the abstract functions and hence the cryptographic scheme constructed from them, now take a concrete form: they can be implemented on real-life computers. The claimed cryptographic properties of the functions are shown to hold if, for example, factoring integers is indeed an intractable problem, as Goldwasser, Micali and Rivest did with their claw-free pairs of trapdoor permutations, for instance.

One typical way to extend results in theoretical cryptography is to show that a given cryptographic scheme is available under *weaker cryptographic complexity assumptions* than was known before. For instance, it was later shown that signature schemes can be constructed under the (abstract) assumption that one-way functions exist. Indeed, it is easy to prove that the existence of claw-free pairs of trapdoor permutations implies the existence of one-way functions, while the reverse is unlikely to hold. Existence of one-way functions is the weakest complexity assumption in public-key cryptography. Other examples of commonly used abstract functions are given by one-way permutations, collision-intractable hash-functions (functions h mapping elements from a large set V onto elements of a small set W, such that it is hard to find two elements x and y in V such that $h(x) = h(y)$) or one-way group homomorphisms. All of these functions can be realized under the factoring and discrete logarithm assumptions. Yet other *primitives* from which cryptographic schemes are constructed, are given by the existence of Oblivious Transfer (an interactive protocol by means of which one party sends a message to a second party, such that it is received with probability 1/2, leaving the sender ignorant about whether it was received or not) or just the existence of secure public key encryption.

Reducing *communication-* or *round-complexity* is another important aspect of cryptographic research. For instance, secure signatures were shown to exist if one-way functions exist, but the price to be paid is signatures of much larger size. It is then a natural question to ask whether a given cryptographic service, such as digital signatures, is available with improved communication complexity, possibly under a slightly stronger complexity assumption. The theory of zero-knowledge proofs for NP-languages has been developed in a number of directions, including weakening cryptographic complexity assumptions and communication complexity. Concerning zero-knowledge proofs or in fact concerning any interactive cryptographic protocol, reducing the number of rounds (the number of times that messages are sent back and forth) may constitute an efficiency improvement. The achieved efficiency level (communication, computation and number of rounds, for instance) is usually related to the security parameter of a scheme, the number of participants, etc. and summarized in "cost-functions" describing just the global (*asymptotic*) behaviour.

Usually, designers of practical cryptographic protocols are less concerned with asymptotical security and efficiency properties of their schemes than theoreticians are. The constraints placed by, for instance, the computational power and (secure) storage capacities of existing hardware, and by available bandwidth, call for the design of cryptographic schemes that are cost-effective and can operate in real-time. Although motivated by theoretical notions which form the basis of the cryptographic applications, one often has to depart from the actual designs proposed in the theory and come up with a compromise between practicality and theoretical clarity, since many theoretical methods seem impractical given the available technology of today and maybe even tomorrow. As a small example, consider the practical use of the RSA-scheme. It is well-known that "plain RSA", as defined in 1977, cannot directly be applied in case of digital signatures. The arithmetical properties of the RSA-functions allow one to multiply any two signatures (in case of plain RSA) and obtain a signature on the product of the two corresponding messages, even without knowing the (secret) factorization of the modulus. This may be a real security threat to the signer. To prevent this possibility, many so-called redundancy methods have been proposed. A recent example is the ISO9796-standard [81]. Before signing a message, it is very efficiently modified through some publicly known process so as to disable this "homomorphic" attack. However, this composition of the RSA-functions together with, say, ISO9796 does not seem to lead to a signature scheme that can be proven secure if inverting RSA-functions (without knowledge of the modulus' factorization) is infeasible. Moreover, it is not clear which cryptographic properties of a redundancy scheme would render the signature scheme (provably) secure. Thus it may still be possible to attack the scheme, without even "breaking" the RSA-functions. We do not mean to suggest here that these methods are insecure, but it is clear that a practical signature scheme not suffering from this potential defect would have to be preferred. In general, it seems that many practical methods of design simply seem to prevent a careful and satisfying security analysis. On the other hand, if the theoretical result from [101] were implemented, a signature's length would be quadratic in the length of the modulus, which is undesirable in many practical circumstances.

For practicality, we would like to have *modular design methods* for constructing a variety of useful and secure cryptographic applications. The basis of such a method would be an abstract cryptographic module. This is a type of function or protocol, defined in terms of its cryptographic properties. A cryptographic application constructed from such modules should be provably secure, given the properties of the module and a suitable definition of security, of course. Furthermore, concrete modules should be efficiently and widely available. Indeed, for any composite scheme to be of practical value, its constituents must at least have this property as well. By "widely available", we mean that preferably the modules can be implemented under a host of previously known cryptographic complexity assumptions, including abstract *and* concrete ones. No such module seems to have been proposed to date.

Our present work indicates that a certain class of interactive two-party protocols is particularly suitable as a basis for the design of a variety of efficient and secure cryptographic services. Our abstract modules, called *Σ-protocols*, are realized when instantiated for almost any of the most commonly used cryptographic complexity assumptions, both concrete, such as difficulty of factoring or computing discrete logarithms, and otherwise, such as existence of certain cryptographic function families. Practical applications include efficient digital signatures, identification schemes, secret ballot election schemes, and communication efficient zero knowledge proofs for NP-languages.

We consider modules that constitute an interactive protocol between a "prover" and a "verifier". The protocol consists of three moves only. The prover and verifier share some common input x. The prover has access to some private input w, which is related to x via some efficiently testable binary relation. Typically, we choose such relations such that it is hard ("infeasible") to compute w having access to x only. Furthermore, the prover has access to public algorithms by means of which the messages corresponding to the first and third moves are computed. In the second move, the verifier is required to send a "challenge" to the prover, which consists of a sequence of randomly chosen bits . After the exchange of messages, the verifier checks whether the "conversation" consisting of the three messages exchanged, is "valid" with respect to the shared string x. If the conversation is valid, the prover is said to have passed the protocol.

The cryptographic properties we require from these modules are as follows. First, we require the protocol to be *honest verifier zero knowledge.* This means that, if the verifier follows the protocol (that is, indeed sends randomly chosen bits as required), then the verifier could have generated the corresponding probability distribution on the conversations without even talking to the prover. In other words, a verifier following the protocol will learn nothing (beyond the fact that the prover is able to pass the protocol) that could not have been computed before the start of the protocol. In our definition of honest verifier zero knowledge, we require nothing about what happens if the verifier does not follow the protocol. Second, *collision-intractability* expresses the idea that without having access to the private string w, the prover should not be able to pass the protocol. More precisely, we require that it is infeasible to compute two valid conversations having different challenges but the same first messages, if one observes the common string x only. In the cryptographic literature, there are numerous examples of three-move interactive protocols satisfying similar properties, see for instance the classical identification protocols of Feige-Fiat-Shamir [66] or Schnorr [106]. However, the primitive we consider has not been suggested earlier as a basis for modular design of practical and secure cryptographic applications.

Collision-intractability and honest verifier zero-knowledge in particular, may seem to give a relatively low level of security in any realistic setting. However, using various methods that are developed in our basic theory, compositions based on our modules can lead to practical cryptographic schemes that are secure against the most general (but computationally bounded) adversaries; suitable compositions will maintain security in the presence of enemies deviating arbitrarily ("adaptive attackers") from expected behaviour. An advantage of our approach is that it seems easier to first design an efficient scheme with security properties such as the ones we require, and complete it with an efficient transformation that achieves the desired security level. This is exemplified by our results on the existence of our module and the variety of cryptographic schemes that can be efficiently and securely constructed from it.

1.3 Overview

The research presented in this work is based on [43, 44, 45, 46, 47, 48, 49, 50, 51]. In Chapter 2 we start by defining Σ-protocols which support the modular design of the practical and secure cryptographic applications described in the remaining chapters. Next we develop a basic theory of Σ-protocols. We introduce methods of composing with Σ-protocols, most importantly some parallel composition techniques, and study the cryptographic properties of the compositions. We conclude Chapter 2 with results concerning the existence of Σ-protocols and present constructions of Σ-protocols based on, for example, the difficulty of factoring integers, computing discrete logarithms, one-way group homomorphisms and claw-free pairs of trapdoor permutations. Necessary cryptographic background is introduced when needed. For an interesting overview of the history and basic methods of modern cryptology, the reader is referred to [113].

In Chapter 3, we introduce *partial proofs.* These are a special type of Σ-protocols and constitute a fundamental primitive. The construction is based on the theory of Chapter 2. Briefly, these proofs concern a number of statements or theorems about which a prover may claim that, for instance, half of them are true. However, a verifier engaging with the prover in the corresponding protocol, obtains no information about which half. In fact, we will consider more general conditions than this threshold condition. Our protocols based on the primitive of Chapter 2 can handle any monotone condition, that is, one that consists of logical AND- and OR-operators. More precisely, we show that Σ-protocols satisfying honest verifier zero knowledge and collision intractability (or special soundness) lead to a Σ-protocol with the same security properties for any set of corresponding statements under any monotone condition. It follows that these protocols are also witness indistinguishable. In its most subtle version, this result makes extensive use of secret sharing in combination with simulation techniques. It turns out that our partial proofs even apply to the case of general honest verifier zero knowledge interactive proofs of the Σ-type, and we show that our approach extends the collection of languages for which zero knowledge proofs are known considerably. In addition, this result uses interactive hashing.

The applications of partial proofs we have considered include the design of efficient (general) zero knowledge proofs, witness hiding proofs and identification methods. Most of these services are essential components in any real-life application where information integrity is vitally important. We are particularly interested in minimizing communication and storage while at the same time the service should be realizable under weak cryptographic assumptions.

In practice, *witness hiding* protocols provide an efficient alternative to zero knowledge protocols. The goal of a zero knowledge protocol is to convince a skeptical verifier of some assertion without giving away the proof. In fact, zero knowledge assures that no information is released (beyond the truth of the assertion) that the verifier could not have computed by itself beforehand. As such, these techniques can be employed in digital identification schemes or as tools in more complex cryptographic schemes. As an example, these techniques can be used to demonstrate that some given value y was computed as $f(x)$, for some given cryptographic function f, while the input x remains hidden. It turns out that in many situations it is sufficient to rely on witness hiding: under suitable computational assumptions, one can relax the zero knowledge concept by saying instead that the verifier learns nothing *useful*. This means that the verifier learns nothing that would enable cheating in the given system. Witness hiding protocols are usually more efficient, since they do not require repetitions of some atomic protocol as is the case with many practical zero knowledge protocols. This partly motivates our research into witness hiding protocols, and these are shown to arise naturally from our partial proofs.

Witness hiding protocols per se are natural candidates for secure digital identification schemes. The security of such service demands that a verifier checking a party's identity should not be able to extract enough information from that party to enable successful impersonation attempts at a later stage. A number of secure and efficient identification schemes are known from the literature. Additionally, however, we consider a so-called man-in-the-middle attacker and identify scenarios where our witness hiding schemes provide particular efficient and secure identification mechanisms that withstand even this kind of attacker. A man-in-the-middle is best described as the person who plays chess against two grand-masters simultaneously, and get at least a draw out of it. To this end the attacker plays white against one of the players and black against the others in a way that the grand-masters are actually playing each other. This may involve some moving back and forth between separate rooms in the best traditions of situation comedies. However, in cryptographic systems that use protocols of the basic challenge-reply type, such attackers may be a real danger to the security of the system. Our solution to this identification problem is conceptually simple and can efficiently be implemented under most of the common cryptographic assumptions.

To prove a statement or theorem in zero knowledge is most efficiently done when the structure of the assertion (or the set from which it is chosen) is exploited. For instance, the earliest examples deal with assertions about quadratic residues. The corresponding zero knowledge proofs explicitly use the fact that these residues are members of a ring. A good deal of the research in this area is concerned with finding classes of languages or assertions that admit efficient zero knowledge proofs. Another direction into which this topic was investigated is general zero knowledge proofs. Here, one is only given that the statements to be proved in zero knowledge are selected from some general NP-language. The task is to design protocols that can handle this situation. Based on our partial proofs, we develop a *communication efficient* solution to this classical problem. Our result compares favorably to previous solutions. More precisely, given a family of Boolean circuits that verify witnesses for the language L in question, we present a zero knowledge proof that a given x is in L, using only a *linear* number of so-called bit commitments in the size of a circuit that verifies x's witnesses.

An area that recently received attention is that of public key cryptography in a distributed setting. There, sensitive information such as cryptographic keys or database information is shared among a number of servers to enhance the reliability of the system. Well-known examples are distributed signatures and encryption/decryption, or more generally, secure function sharing. Cryptography provides elegant mechanisms that are resilient to even malicious faults. Roughly speaking, these mechanisms ensure that parties deviating from their expected behaviour cannot prevent the honest parties to achieve the goal of the system or learn their secret inputs. General solutions are provided in the area of multi-party computations, and most problems in distributed cryptography can be viewed as instances thereof. It turns out in practice that tailor-made designs for a given problem are usually more practical.

We apply some of our theory to the problem of designing *electronic election schemes* and present a particularly efficient solution. Our system is robust against some quorum number of maliciously faulty "government agents". Even when such parties conspire, they cannot learn individual votes or disrupt the election. Forgery of votes is proved infeasible under suitable and plausible cryptographic assumptions. Recall that we gave an example how zero knowledge may be used in a larger protocol to prove that a computation was carried out correctly. However, in some cases, zero knowledge can be shown to be unnecessarily strong, leading to impractical solutions. If one gives a zero knowledge proof that y is computed as $f(x)$ for some x, the verifier learns *nothing at all*, beyond that fact. A *witness indistinguishable proof* may provide the same level of conviction. However, it may only ensure that the verifier learns *nothing about x*. This may very well be sufficient in some cases. Our result for electronic election schemes provides such an example. To prove that an *encrypted vote* was computed correctly (the encryption contains a valid vote) without revealing the actual vote, we use a witness indistinguishable proof. In this way, we achieve a considerable improvement of the communication complexity compared to previous solutions, which used zero knowledge proofs instead. This is practical evidence that it is worthwhile to consider witness indistinguishable protocols instead of zero knowledge protocols whenever appropriate. Whether our methods also lead to efficient and secure solutions to other problems in distributed cryptography, would be interesting to investigate.

Chapter 4 deals exclusively with digital signatures. We only consider signatures whose security can be proved under common cryptographic assumptions, such as the difficulty of factoring, RSA-inversion, discrete logarithms, one-way group homomorphisms or claw-free pairs of trapdoor permutations. We show that the interactive protocols studied in Chapter 2 lead to secure signature schemes. We show that a certain class of Σ-protocols is sufficient for the existence of secure signatures whose complexity is similar to the most efficient theoretical result in this area, the Goldwasser-Micali-

Rivest scheme. As we show that our cryptographic complexity assumptions are weaker, we may view our result as an extension of theirs. We also show how to modify their method based on claw-free trapdoor permutations so as to facilitate signatures with smaller communication complexity. Another result we present concerns an efficient transformation that basically removes the need for considering so-called adaptively chosen message attacks. We show that for efficient and secure signatures to exist, it is sufficient that a signature scheme exists that is secure when an attacker only sees signatures on randomly chosen messages. We conclude this work by presenting the most efficient signature scheme known to date, that is provably secure under a common cryptographic assumption, namely the difficulty of RSA-inversion.

To be able to judge merits of our specific results, such as those concerning the design of zero knowledge protocols, digital signatures or secret ballot election schemes, discussions are provided where their significance is detailed. This provides one way of looking at our results. Our main point, however, is that Σ-protocols provide a sound basis for modular design of practical and secure cryptographic applications. We encourage the interested reader to test the applicability and usefulness of our methods in other cryptographic areas.

2
Σ-Protocols

2.1 Model and Definitions

We consider three-move interactive protocols between two parties, called "prover" A and "verifier" B, where the prover acts first. The prover and verifier are modelled as probabilistic, interactive Turing machines. We furthermore require these games to be of the Merlin-Arthur type [3]. This means that the verifier is expected to send only uniformly chosen bits. We refer to these protocols as Σ-protocols[1].

Turing machines can be seen as an abstraction of computers as they are known today. We give an informal description. To perform some computational task, the machine is provided with an adequate list of elementary instructions (an algorithm). Each of these instructions must be chosen from a finite set of eligible instructions, similar to a programming language as Pascal, for example. Any (private) input to such a machine is placed in a part of the memory called *knowledge tape*. The *auxiliary tape* can be used by the machine to write and read intermediate results from its computations. When the computations dictated by the algorithm have been completed, the machine halts and outputs the outcome on the *output tape*. A Turing machine is called *probabilistic*, if it is also furnished with a *random tape*, which is a part of the memory allocated for the storage of random bits. In this case the machine is allowed to read random bits and use them in any

[1]Spelled out, the first part of *Sigma* refers to "zig-zag" symbolizing the three-moves, while the last part is an abbreviation of "Merlin-Arthur".

of its computations. To enable communication between Turing machines, they can be given a *communication tape.* Interactive Turing machines are "connected" by such a tape, and can communicate by writing to and reading from this tape. A Turing machine is said to be *polynomial time*, or sometimes *efficient*, if the total number of read- and write-operations to and from its tapes, is bounded by some polynomial in the length of the input. Computational tasks which cannot be performed by a polynomial time Turing machine are called *infeasible* or *intractable.*

If Alg is an algorithm, then the notation "$a \leftarrow Alg(b)$" refers to the computation of the output "a", on input bit string "b". If V is a set, $v \leftarrow V$ denotes uniform and random selection of an element v from V. For $x, y \in \{0,1\}^*$, we use "$x|y$" to denote the usual concatenation of x and y.

The *common input* to both players in a Σ-protocol is some string x. Thus, x is placed on the knowledge tapes of both players. The prover has a string w as private input. The pair (x, w) is an element of some given binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$, which is typically (but not necessarily) polynomial time verifiable. For any string x the set $RW(x)$ is defined as the set of strings w such that $(x, w) \in R$. Such a string w is called a *witness for x*. We assume that for some given polynomial $p(\cdot)$ it holds that $|w| \leq p(|x|)$, where $|\cdot|$ denotes binary length. The set RX is defined as the set of strings x such that $RW(x)$ is non-empty. Prover A and verifier B are modelled by a probabilistic polynomial time (PPT) interactive Turing machines, unless stated otherwise.

In the following, R_A resp. R_B denotes the random tape of the prover A resp. the verifier B. The protocol that A and B will execute is denoted (A, B) and consists of the polynomial time algorithms $a(\cdot)$, $c(\cdot)$, $z(\cdot)$ and $\phi(\cdot)$, to be explained hereafter. The protocol (A, B) and the relation R are *public*; their descriptions are available to all players. By running the algorithm $a(\cdot)$ on the common string x, the private input w and possibly random bits taken from R_A, the prover computes the *initial message a*. The length of this initial message a is denoted t_A, the *authentication length.*

After having received a, the verifier B chooses a *challenge c*, by running the algorithm $c(\cdot)$ on input of its random tape R_B. The algorithm $c(\cdot)$ just picks an appropriate number, say t_B, of bits from the random tape. This move is completed when B sends the challenge c to A. In the following, we will drop the explicit use of the algorithm $c(\cdot)$ and will write the selection of c as "$c \leftarrow \{0,1\}^{t_B}$".

To compute the *reply z*, the prover A runs the algorithm $z(\cdot)$ on x, w, a, c and R_A. The computation of z may require that a and possibly auxiliary information are regenerated from the relevant random bits of R_A. The reply z is allowed to be probabilistic too, by using new random bits from R_A. The resulting output z is sent to the verifier B. Finally, the verifier invokes a polynomial time computable predicate ϕ to check whether the *conversation*

x, a, c, z is *accepting*. Whenever we speak of the *length* of a conversation, we mean the *binary* length of that conversation.

The algorithms $a(\cdot)$, $c(\cdot)$, $z(\cdot)$, $\phi(\cdot)$ will typically depend on the relation R and the purpose of the protocol, of course. We will say that (A, B) is a *Σ-protocol for relation R.*

Let A be an *honest* prover, that is, a prover that is given some $(x, w) \in R$ and that behaves as dictated by the protocol for A. An *honest* verifier B is a verifier that indeed chooses the challenge uniformly at random from $\{0,1\}^{t_B}$. For an arbitrary probabilistic (polynomial time) verifier B^* that outputs a bit string of length t_B, (A, B^*) denotes the interaction on some common input $x \in RX$ between the honest prover A, and the verifier B^*. Note that in this case we require nothing about B^*'s output distribution. Let A^* be an arbitrary probabilistic (polynomial time) prover. Then (A^*, B) denotes the interaction on some common input x between A^* and the honest verifier B. A^* acts as an honest prover A in the sense that A^* sends an initial message a, and a reply z (after having received a challenge c from B). However, A^* is not required to follow the computations as prescribed for A, whereas B is. We say that (A^*, B) *succeeds* with some given probability if the interaction results in an accepting conversation with that probability. Finally, $T_A(x)$, $T_B(x)$, $T_{A^*}(x)$ and $T_{B^*}(x)$ denote the running times of the respective algorithms.

$$\begin{tikzcd}
\mathsf{A} & & \mathsf{B} \\
a \leftarrow a(x, w, R_A) & & \\
& \xrightarrow{\quad a \quad} & \\
& & c \leftarrow c(R_B) \\
& \xleftarrow{\quad c \quad} & \\
z \leftarrow z(x, w, R_A, c) & & \\
& \xrightarrow{\quad z \quad} & \\
& & \phi(x, a, c, z) \overset{?}{=} \mathit{accept}
\end{tikzcd}$$

We will sometimes say that a given interactive Turing machine is available to us as a *black-box*. By this we mean that we can run the machine as many times as we desire (but at most a polynomial number of times). Moreover, we are allowed to put any strings of our choice on its communication tape. Whenever we draw conclusions about a black-box, these will rely on its output behaviour, not on inspection of its memory or the particular program it runs. As an example of a black-box, we say that some arbitrary PPT prover A^* is given as a black-box, if we are allowed to run an arbitrary PPT verifier B^* against A^*. Then B^* can pose as the verifier any polynomial number of times and interrogate A^* accordingly.

We say that an interactive Turing machine is given as a *rewindable* black-box, if additionally, we are allowed to supply the random bits for the random tape of the machine. Without loss of generality, we may assume that such a rewindable black-box is memory-less, that is, its computations are always uniquely determined by its initial state and the contents of its random tape.

2.2 Requirements

We now formulate various cryptographic properties that we can require from Σ-protocols. Let (A, B) be a Σ-protocol for relation R.

2.2.1 Completeness

Let $(A, B)_{x,w}$ denote the conversation between the honest prover and the honest verifier on common input x, and private input w to the prover A. If $(x, w) \in R$, then $(A, B)_{x,w}$ leads to acceptance by the verifier B with probability 1. Any Σ-protocol we consider will satisfy completeness.

2.2.2 Special Soundness and Collision Intractability

Let $x \in \{0,1\}^*$ be given. A pair of accepting conversations (x, a, c, z) and (x, a, c', z') with $c \neq c'$ ("accepting conversations for x, with the same first message but different challenges") is called a *collision*. Note that we did not require that these conversations are the possible result of running (A, B) on some input. The Σ-protocol (A, B) for relation R is said to have the *collision-property* if and only if the following holds. Given a collision for $x \in RX$, we can efficiently compute some w such that $(x, w) \in R$, that is, there exists a PPT machine (depending on (A, B) and R) that on input of a collision for x, outputs w such that $(x, w) \in R$. It is important to note that in case $x \notin RX$, nothing is required.

Now, the Σ-protocol (A, B) for relation R is said to satisfy *special soundness*, if and only if it has the collision-property and there exist no collisions for $x \notin RX$. In case $x \notin RX$, we claim that an arbitrary prover A^*'s success probability ϵ_{A^*} of leading the honest verifier B to acceptance is at most $1/2^{t_B}$. This is shown as follows. Consider the 0/1-matrix whose row-indices run over all possible values ρ of A^*'s random tape R_{A^*} and whose column-indices run over by all possible values c of the challenge. The entry (ρ, c) in this matrix is defined as 1 if B accepts when A^* would use the random string ρ and B would choose c as the challenge, and (ρ, c) is defined as 0 otherwise. Clearly, A^*'s success probability is equal to the fraction of 1-entries in the matrix. But if this success probability were greater than $1/2^{t_B}$, than there exists at least one row with two 1-entries, which is the same as saying that there exists a collision. This contradicts the fact that for $x \in\notin RX$, no collisions exist. Please note that the analysis above holds regardless whether or not A^* follows the instructions for an honest prover

A or not.

Another observation about special soundness is the following. When, for instance, membership of RX can be tested efficiently, this could be included in the verification of conversations, that is, in the description of ϕ. In this case, the collision-property implies special soundness.

In many examples where special soundness is in the play, R will actually be a polynomial time verifiable relation. For completeness however, we note the following. Since we didn't assume a priori that R is polynomial time verifiable, how can a prover check whether a given pair (x, w) will help to make a verifier accept x with probability 1? For $(x, w) \in \{0,1\}^* \times \{0,1\}^*$ such that $|w| \leq p(|x|)$, define $(A, B(\mathbf{0},\mathbf{1}))_{x,w}$ as $(A, B)_{x,w}$ while having A reply to the all-0 string $\mathbf{0}$ and to the all-1 string $\mathbf{1}$, and keeping the first message fixed. This results in two conversations with the same first message yet different challenges. Now the prover runs $(A, B(\mathbf{0},\mathbf{1}))_{x,w}$. If the resulting conversations are not accepting, then $(x, w) \notin R$. If they are, then by the collision-property, this allows the prover to compute w' such that $(x, w') \in R$. The prover proceeds in this case with (x, w'). Note that special soundness implies that RX must be an NP-language.

Many times we will assume that we are given a (publicly known) PPT instance generator G for relation R to generate *solved instances*: $(x, w) \leftarrow G(1^k)$, where $(x, w) \in R$ and where k is a security parameter. Let $[G(1^k)]$ denote the probability distribution induced on R by G, on input 1^k, and let $[G(1^k)]_x$ be the restriction of $[G(1^k)]$ to the public strings x, that is, $[G(1^k)]_x$ is the probability distribution obtained by running $G(1^k)$, getting $(x, w) \in R$ and deleting w. We say that x is generated according to G, when x is drawn from $[G(1^k)]_x$. The instance generator G is called an *invulnerable generator* if it is infeasible, given just a string x generated according to G, to compute a witness w. In other words, if we select x from $[G(1^k)]_x$, then no PPT algorithm will, on input x, output w such that $(x, w) \in R$ except with negligible probability in k (that is, negligible as a function of the security parameter k).

(A, B) is called *collision-intractable* over R and G, if there is no PPT algorithm that, on input x as generated by G, computes a collision, except with negligible probability. Note that this PPT algorithm is not given access to an honest prover for x (we deal with that scenario in Section 2.2.4).

In this case, we will refer to (A, B) as a collision intractable Σ-protocol for relation R with generator G. It follows that if (A, B) is collision intractable over R and G, then G must be invulnerable. Also, if G is an invulnerable generator and (A, B) has the collision-property, then the protocol (A, B) is collision intractable over R and G.

2.2.3 Zero-Knowledge

Zero knowledge is a vast and rich research area in cryptography and its seeds have been sown by Goldwasser, Micali and Rackoff [68], who formally defined the concept. The basic intuition is as follows. Suppose that A and B engage in an execution of their protocol on common input x. Here x may be a theorem that A knows how to prove (a witness w may be viewed as a proof for the statement x), or w may be a secret piece of information (a "secret key") related to x that allows A to enjoy privileges such as gaining access to some computer network.

In general B's interest is to verify that A holds a witness w (a proof or a secret key). On the other hand, it may very well be in A's interest to not give away w. Zero-knowledge is the idea that no matter how B behaves as a verifier, it will not learn any information that it could not have computed itself, even before the start of the protocol. Technically, there is an expected polynomial time machine M (called the *simulator*) that, on input x, produces accepting conversations with exactly the same distribution as conversations between a prover who is given $(x, w) \in R$ and the given verifier B. The machine M is expected polynomial time, in the sense that, on input of a fixed x, it generates the desired output in polynomial time for "almost all" possible random tapes of M.

We now define a particular and weaker variant of zero knowledge. (A, B) is said to be *honest verifier (perfect) zero-knowledge* if it is easy to (perfectly) simulate conversations with an honest verifier. If, additionally, the simulator works by taking any uniformly chosen challenge c as input and outputs an accepting conversation where c is the challenge (an accepting conversation (x, a, c, r)), then (A, B) is said to be *special honest verifier (perfect) zero-knowledge*. In this case as well as in that of ordinary honest verifier zero knowledge, we need to define the behaviour of M when it is given $x \notin RX$. This is only for reasons of completeness of some of the protocols to follow. If $x \notin RX$, then we just assume that $M(x)$ (or $M(x, c)$ respectively) either outputs an accepting conversation (where c is the challenge), or outputs '?'. No conditions on the output distribution are required in this case.

In the definition of ordinary zero knowledge, one has to take into account the potentially malicious behaviour of the verifier. This is done by additionally giving the verifier as a rewindable black-box to the simulator. This then captures the idea that the verifier can generate the same distribution, as when talking to the prover, on its own by just running the simulator. The simulator is required to run in expected polynomial time. Above, we considered *perfect* zero knowledge. If one relaxes the notion of indistinguishability of the simulation and the real conversations, one can define computational and statistical zero knowledge.

Why should we look at cryptographic protocols that are only zero knowledge against the honest verifier? Indeed malicious behaviour may never be excluded, at least in cryptographic theories. One reason is that such protocols are usually quite efficient compared to "general" zero knowledge protocols and relatively easy to design. A good reason is that there exist transformation methods [95, 53, 54] for turning certain classes of honest verifier zero knowledge protocols into zero knowledge ones! Thus, in theory, it is sufficient to design protocols that are zero knowledge against the honest verifier and apply a transformation afterwards.

Another reason is that, as follows from this work, honest verifier zero knowledge Σ-protocols (satisfying collision intractability), turn out to be fine primitives for constructing a range of efficient cryptographic services, secure against arbitrary polynomially bounded enemies.

2.2.4 Witness Hiding and Witness Indistinguishability

The concepts outlined here are due to Feige and Shamir [64], building on [66]. Witness hiding offers an attractive alternative to zero knowledge in many cases. Generally, it turns out in practice that witness hiding protocols are more efficient than zero knowledge protocols. Moreover, the security offered by witness hiding protocols is sufficient in many cases of protocol design.

Let a Σ-protocol (A, B) for relation R with generator G be given. We now consider the situation where a prover A is given an instance $(x_0, w_0) \in R$ as generated by $G(1^k)$ and where A is facing an arbitrary PPT enemy B^*. We assume that the protocol (A, B^*) can be executed on common input x_0 as many times as B^* desires. This means that A is given to B^* as a black-box. However, B^* is not allowed to rewind A (that is, A's random tape is not under control of B^*). *Witness hiding* captures the idea that no matter how maliciously the enemy interrogates an honest prover, it gets at most a negligible advantage when trying to compute w_0' in $RW(x_0)$, compared to the situation before the start of the protocol. We put here "w_0'" instead of "w_0", since the witness output by B^* may be different from w_0 (see also the example at the end of this section).

More formally, let ϵ_{B^*} denote the probability that B^* is able to produce $w_0' \in RW(x_0)$ after the interrogation of A^* on common input x_0, where the probability is taken over the coinflips of G and B^*. A *witness extractor* X is a PPT algorithm that receives as input a bit string x generated according to G. Furthermore, it has access to B^* as a rewindable black-box and to the protocol (A, B) and its generator G. In particular, X can generate many pairs $(x_0, w_0) \in R$ according to G, provide honest provers A with those pairs (x_0, w_0) and can have B^* launch its attack against those provers A (but for x *no* honest prover is given). The witness extractor succeeds when it outputs w' in $RW(x)$, given x distributed according to G. Let ϵ_X denote its success probability, taken over the coinflips of B^*, G and X. We say that the protocol (A, B) is *witness hiding* over G, if we can exhibit a witness extractor that succeeds with essentially the same probability as B^*, that is, $\epsilon_X < \epsilon_{B^*} + \nu(k)$, where $\nu(k)$ is negligible.

Note that the definition of zero knowledge guarantees that no information is revealed in case of any fixed common input x_0, whereas witness hiding only guarantees that no *useful* information is given away on the average: suppose that the generator G is invulnerable and that the protocol (A, B) is witness hiding over G. If a PPT enemy B^* could efficiently extract a witness w_0' from his interaction with the honest prover A on common input x_0, this would then imply the existence of an efficient algorithm that receives x as generated by G and outputs a witness w for x. Thus, talking to an arbitrary PPT enemy B^* is secure from the point of view of A, since the latter contradicts the assumption that G is invulnerable.

In practice, many proofs that a given protocol is witness hiding use the notion of *witness indistinguishability*. Assume that for each $x \in RX$, the set $RW(x)$ contains at least two elements. Let an arbitrary $x \in RX$ be given and let w_1 and w_2 be two distinct elements from $RW(x)$. Consider the honest prover A_1 who gets (x, w_1) as input while the honest prover A_2 gets (x, w_2) as input. Let B^* be an arbitrary (unbounded) verifier. We say that the protocol (A, B) is *witness indistinguishable over* R if B^* playing the role of B cannot distinguish between conversations with A_1 and A_2: the distributions of the conversations generated by (A_1, B^*) and (A_2, B^*) are (perfectly) indistinguishable. It follows trivially that if (A, B) is zero knowledge, then it is also witness indistinguishable.

The following way of reasoning is often encountered in a proof that a given protocol is witness hiding. Suppose that each $x \in RX$ has, say, exactly two distinct witnesses w and w', that the protocol (A, B) is witness indistinguishable and that an invulnerable generator G is given. Suppose furthermore that the invulnerability of G additionally implies that, given $(x, w) \in R$ as generated by G, it is infeasible to compute $w' \neq w$ such that (x, w') is in R as well. The proof is by contradiction. Suppose that we are given an efficient enemy B^* as above. We "abuse" B^* to break the invulnerability of G as follows. Generate $(x, w) \leftarrow G$ and give (x, w) to an honest prover A. Next, let B^* run its attack against A. With good probability, B^* outputs w' such that $(x, w') \in R$, by assumption on B^*. However, by the witness indistinguishability of the protocol (A, B), B^* cannot be biased as to which of the two possible witnesses are output. Thus, with probability $1/2$, we have $w \neq w'$, and we have contradicted the invulnerability of G.

2.2.5 *Proofs of Knowledge*

A proof of knowledge is a protocol between a prover and a verifier that allow a prover A to convince a sceptical verifier B that A "knows" some piece of information that is somehow related to their common input x. As an example, x may be a large composite integer of which A knows a non-trivial factor. The concept of proofs of knowledge was formalized by Feige, Fiat and Shamir [66].

Let (A, B) be a Σ-protocol for relation R. Let A^* denote a PPT prover that makes the honest verifier B accept the common string x with probability ϵ_{A^*}. Using the approach of [66], we say that a Σ-protocol (A, B) is a *proof of knowledge* for relation R if there exists a *knowledge extractor*: this is a PPT machine that, on input of the prover A^* as a rewindable black-box, outputs a witness w for x, with essentially the same success probability as that of (A^*, B). The probability is taken of the coinflips of A^* and the knowledge extractor (*not* over the distribution of x). If A^* is successful with non-negligible probability, A^* is said to "know" a witness w for x. For a thorough treatment of proofs of knowledge, please consult Bellare and Goldreich [9].

As an example, consider the following situation. If (A, B) satisfies special soundness and if the challenge length t_B is, say, linear in the length of the common input x, it can be shown to be a proof of knowledge as follows. Suppose that $\epsilon_{A^*} > 1/2^{t_B}$. Select a random tape for A^* and execute the protocol (A^*, B) on common input x. If B accepts x (say, the resulting accepting conversation is (x, a, c, z)), then re-use the same random tape (causing A^* to send the same first message) and have B challenge A^* with a random challenge c' *different* from c. If A^* outputs an accepting conversation once again, say (x, a, c', z'), then we can immediately invoke the algorithm that efficiently computes a witness for x and that is guaranteed by the definition of special soundness. This particular knowledge extractor can be shown to run in expected time polynomial in $1/(\epsilon_{A^*} - 2^{-t_B})$ and the running time of A^* itself. Thus, if we assume that ϵ_{A^*} is non-negligible (as a function of $|x|$) and if we assume that say, $t_B = |x|$, the expected running time of the extractor is polynomial in $|x|$ (by running it poly($|x|$) times). The knowledge extractor and the technical details are given in the proofs of Proposition 2.1 and Corollary 2.1. If t_B is too small, we can turn a Σ-protocol (A, B) satisfying special soundness into a proof of knowledge by doing an appropriate number of parallel executions of (A, B).

Collision intractability may be viewed as capturing a *weak* version of knowledge soundness [66]. Even though a prover may have success probability 1, it is unclear, in theory, whether the prover knows a witness in the sense of [64]. But there exists an efficient algorithm that extracts a collision from such a successful prover, by running the prover as a rewindable black-box.

2.2.6 Interactive Proofs

Let L be a language (that is L is a subset of $\{0, 1\}^*$). Let $x \in L$. An *interactive proof* [68] for L is a game between a prover (which may have unlimited power) and a polynomially bounded verifier, by means of which the prover convinces the sceptical verifier that a given input word x is in the language L. If it is, the verifier will always accept (completeness). If not, then the verifier will accept only with exponentially small error probability (soundness) $1/2^k$, where k denotes the size of x. Usually an interactive proof consists of an "atomic" protocol that has some constant error probability, say $1/2$, and the required confidence level is then achieved by an appropriate number of repetitions. To such atomic protocols we will also refer as interactive proofs, the required repetitions to be understood.

A $PZKIP$ for a language L is a perfect zero knowledge interactive proof for L, see [68]. In a perfect zero knowledge *argument* for L, the prover is polynomially bounded, and the soundness condition holds provided that the prover is unable to break some given computationally hard problem of size k.

2.2.7 Signature Protocols

Let (A, B) be a Σ-protocol for relation R and generator G. If (A, B) satisfies

1. collision-intractability over R and G
2. special honest verifier zero-knowledge
3. $t_A < t_B$ (the length of the initial message is smaller than the length of the challenge),

then we call (A, B) a *signature protocol.* If (A, B) is collision-intractable over R and G and is honest verifier zero-knowledge (so it does not necessarily have a *special* simulator and does not necessarily satisfy $t_A < t_B$), we call (A, B) is a *quasi-signature protocol.*

These protocols are not to be confused with digital signature schemes. Nevertheless, we will show in Chapter 4 that they imply signature schemes, hence the name and significance of signature protocols. It turns out (Proposition 2.4) that signature protocols can be manipulated to satisfy any desired ratio between the length of the first message and the length of the challenge. This fact is exploited in Chapter 4. Thus, the condition on the length of the challenge allows us to use these protocols in a tree authentication structure since challenges in the manipulated protocol can be made large enough to "capture" some number of first messages: in some constructions to follow, we will let the challenge c of the verifier B consist of the concatenation of a number of first messages a', which are in subsequent executions used in the same mode as the current first message a. In order to build, say, a binary tree structure, the length t_B of a challenge c should be at least twice as large as the length t_A of a single first message. Such manipulations are particularly efficient in case $t_B - t_A$ is, say, linear in the security parameter, or in case the protocol given is based on (special) one-way group homomorphisms (Sections 2.5.1 and 2.5.2).

2.3 Examples

2.3.1 Introduction

An early example of three-move protocols, now systematically studied in this work, is the "Passport Protocol" of Fiat and Shamir. Based on the difficulty of factoring integers, they proposed practical identification protocols. Their results where later extended by Feige, Fiat and Shamir [66] and finally Feige and Shamir [64], culminating in the powerful notion of witness hiding. Since then many identification schemes of this format have been proposed, for instance by Schnorr [106], Guillou-Quisquater [76], Ong-Schnorr [94], Brickell-McCurley [26] and Okamoto [93]. The schemes from [94, 26, 93] have been proved to be witness hiding under plausible assumptions, while for the other schemes this is conjectured. All schemes can be shown to be Σ-protocols satisfying special honest verifier zero knowledge and collision intractability, and in some cases even special soundness. Most of these protocols are also proofs of knowledge. By sequential composition and choosing small challenges, these protocols can be turned into zero knowledge protocols. This, however, is usually undesirable for reasons of loss of efficiency. A witness hiding proof of knowledge can be used as a secure identification scheme (see [64]). A nice example of a secure identification scheme that is witness hiding but not a proof of knowledge, is provided by [94]. The proof was recently given in [112]. In our terminology, this scheme satisfies collision intractability, but not special soundness.

For any integer $N > 1$, let $\mathbb{Z}_N$ denote the ring of integers modulo N, and let $\mathbb{Z}_N^*$ denote its group of units (that is, the elements having an inverse under multiplication). It is well known that the order of $\mathbb{Z}_N^*$ is $\phi(N)$. Here $\phi(N)$ denotes the Euler totient-function of N, that is, the number of integers greater than 0 and smaller than N being relatively prime to N. We now give two of the most accessible examples.

2.3.2 Based on Discrete Logarithms

Let G_q be a group of prime order q, and let $g \in G_q$, $g \neq 1$. Now note, by elementary group theory, that for each $h \in G_q$, there is a unique $w \in \mathbb{Z}_q$ such that $g^w = h$. This w is called the discrete logarithm of h with respect to g, also denoted $\log_g h$. Now let h be a random element of G_q. The *discrete logarithm problem* for G_q is to find w given g and h. This problem has been first considered for cryptographic use by Diffie and Hellman [56], and underlies many of today's cryptographic schemes. More precisely, the problem for groups of prime order can be defined as follows. Let $\mathcal{G}$ be a family of groups of prime order such that the group operations can be performed efficiently, group elements can be efficiently sampled with uniform distribution and group membership as well as equality of group elements can be efficiently tested. Let Gen be a probabilistic polynomial time generator that on input 1^k outputs a description of a group $G_q \in \mathcal{G}$ (including the group order q, for some prime q), and two random elements g, h from G. Furthermore, elements from G_q can be represented with k bits. We say that the discrete logarithm problem for $\mathcal{G}$ is intractable (over Gen) if there is no probabilistic polynomial time algorithm that on input of (a short description of) G_q, g and h as output by $Gen(1^k)$ can compute $\log_g h$ with non-negligible probability in k.

In order for discrete logarithms to make sense, we must find such a group G_q where computing discrete logarithms is hard. Most commonly, G_q is taken as a subgroup of the multiplicative group of a finite field, or as a subgroup of the group of points on an elliptic curve. The discrete logarithm problem in these groups is generally perceived as very hard. For detailed information see for instance [87].

We have the following Σ-protocol (A, B) satisfying special soundness and special honest verifier zero knowledge: Schnorr's identification scheme [106]. It is also a proof of knowledge. Let $\mathcal{G}$ be a family of groups as above. A relation R is defined as follows. Let the group G_q of prime order, say q, be an arbitrary member of $\mathcal{G}$. Let $g \neq 1$ and h be arbitrary members of G_q. Now put $x = (G_q, g, h)$. The corresponding (and unique) witness is $w \in \mathbb{Z}_q$ such that $g^w = h$. The relation R consists of all such pairs (x, w).

The inclusion of "G_q" in the definition of x should be read as a short description of the group G_q (including the group order). For instance, groups G_q in $\mathcal{G}$ could be taken to be subgroups of order q in $\mathbb{Z}_p^*$ for some primes p and q. The short reference to G_q could then consist of the primes p and q. Note that primality of integers can be decided efficiently [2]. Furthermore, the prime number theorem (see for instance [78]) says, roughly speaking, that the fraction of primes among all integers having length k bits is $1/k$. Hence, large primes can be generated efficiently. Usually, p will have more than 500 bits, and q at least 160 bits.

An invulnerable generator G runs Gen first to obtain (a description of) $G_q \in \mathcal{G}$. Next, G selects $g \in G_q/\{1\}$ and $w \in \mathbb{Z}_q$ at random. Then, h is defined as $h \leftarrow g^w$. G outputs $x = (G_q, g, h)$ and w.

Key-generation: A runs G and gets $x = (G_q, g, h)$ and w. Common input to A and B: x. Private input to A: w.

Move 1: A selects $u \leftarrow \mathbb{Z}_q$ and computes $a \leftarrow g^u$. A sends a to B.

Move 2: B selects $c \leftarrow \mathbb{Z}_q$ and sends c to A.

Move 3: A computes $z \leftarrow cw + u \bmod q$ and sends z to B, who accepts if and only if $g^z = ah^c$.

As for *special soundness*, we note the following. Given two accepting conversations x, a, c, z and x, a, c', z' with $c \neq c'$, w is computed as $w \leftarrow (z - z')/(c - c')$. Also note that our example of G_q based on $\mathbb{Z}_p^*$ enjoys the property that the verifier can efficiently decide whether a given $x \in RX$. In that case the protocol indeed satisfies special soundness and with q large enough, it is a proof of knowledge for the relation R. Since G is invulnerable if the discrete logarithm problem is intractable for $\mathcal{G}$ (over Gen), we conclude that the protocol is *collision-intractable* as well.

Finally, we argue that the protocol is special honest verifier zero-knowledge. Select $c, z \leftarrow \mathbb{Z}_q$ and compute $a \leftarrow g^z h^{-c}$. Then x, a, c, z is an accepting conversation with the right distribution. This protocol is not known to be witness hiding or zero knowledge, but it is conjectured to be secure against an arbitrary polynomially bounded verifier, if computing discrete logarithms is intractable.

2.3.3 Based on RSA

The following protocol (A, B) is Guillou-Quisquater's identification scheme [76], and it is based on the RSA functions [100], which were proposed by Rivest, Shamir and Adleman.

We first describe the RSA functions. Suppose n is an odd integer that is the product of two distinct primes p and q. Such integers n are called RSA-moduli. Let v be an integer, of size polynomial in $|n|$, relatively prime to $\phi(n)$. Now let $\overline{v}$ denote the inverse of v in $\mathbb{Z}_{\phi(n)}^*$, that is, $v \cdot \overline{v} \equiv 1 \bmod \phi(n)$. By elementary number theory, we have that $(x^v)^{\overline{v}} = (x^{\overline{v}})^v = x \bmod n$ for each $x \in \mathbb{Z}_n^*$.

It is conjectured that finding $x^{\overline{v}}$ for random x, given just n, v and x is hard, if factoring integers is hard. If one is additionally given the factorization of n, this task can be performed efficiently. When given p and q, $\phi(n)$ is easily computed and hence $\overline{v}$ can easily be computed from v. All other operations, exponentiation mod$n$, computing multiplicative inverses mod $\phi(n)$ are well-known polynomial time computations. Although factoring random RSA-moduli is hard, generating factored and random RSA-moduli can be done efficiently, since selection of two random large primes and multiplying them can be performed efficiently. The exponent v is chosen at random from $\mathbb{Z}_{\phi(n)}^*$. In practice, the primes p and q are randomly chosen such they have (approximately) the same binary length and such that that the binary length of the resulting RSA-modulus n is at least 750 bits.

The RSA functions are as follows. Let p, q, n, v and $\overline{v}$ be as before.

$$RSA_{n,v} : \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_n^*$$

$$y \mapsto y^v \bmod n$$

Note that $RSA_{n,v}^{-1}$ (the inverse of $RSA_{n,v}$) is identical to $RSA_{n,\overline{v}}$. Based on our assumptions about RSA and the observations made above, $RSA_{n,v}(y)$ can be computed efficiently on any input $y \in \mathbb{Z}_n^*$ given just n, v and y. On the other hand, the inverse $RSA_{n,v}^{-1}$ can be computed on a random input $y \in \mathbb{Z}_n^*$ only if p and q are given as side-information (a trapdoor).

Now define the relation R as follows. Let n be an RSA-modulus, and let v be a *prime* smaller than n with v relatively prime to $\phi(n)$. Let h and w be members of $\mathbb{Z}_n^*$ such that $w^v \equiv h \bmod n$. Defining $x = (n, v, h)$, the relation R consists of all pairs (x, w). An invulnerable generator $G(1^k)$ for this relation first generates a random factored RSA-modulus n of the desired length k bits (its prime factors p and q having length approximately $k/2$ bits). Then the prime v is selected at random such that it is smaller than n and relatively prime to $\phi(n)$. Finally, w is chosen at random from $\mathbb{Z}_n^*$, and h is defined as $h \leftarrow w^v \bmod n$. The following scheme is the identification scheme of Guillou and Quisquater [76].

Key-generation: A runs the generator G and gets $x = (n, v, h)$ and w, as detailed above. Common input to A and B: x. Private input to A: w.

Move 1: A selects $u \in \mathbb{Z}_n^*$ and computes $a \leftarrow u^v \bmod n$. A sends a to B.

Move 2: B selects $c \leftarrow \mathbb{Z}_v$ and sends c to A.

Move 3: A computes $z \leftarrow w^c u \bmod n$ and sends z to B, who accepts if and only if $z^v = ah^c \bmod n$.

Note that the factorization of the modulus n is not required for the prover in this particular protocol. This protocol satisfies the following properties. As to *collision intractability*, note that from two accepting conversations x, a, c, z and x, a, c', z' with $c \neq c'$, w is computed as $w \leftarrow (z/z')^{\frac{1}{c-c'}}$, where $\frac{1}{c-c'}$ is computed in $\mathbb{Z}_v^*$ (implicitly we have assumed that the verifier B has checked the primality of v). In other words, the protocol satisfies the collision-property. Thus, any efficient attacker that could compute these collisions given just n, v and h, could immediately be used to invert the RSA functions! To appreciate this, note that we could just as well generate h as a random element of $\mathbb{Z}_n^*$ where n is an RSA-modulus generated by G but whose factorization is not given to us. Then, we present $x = (n, v, h)$ to the attacker (who cannot distinguish between this x and one generated by G). From the returned collision, we then compute $h^{\overline{v}} \bmod n$, and we have a contradiction with our assumptions on the RSA functions. We may interpret collision intractability as follows. Suppose that v is quite large, say, having a number of bits corresponding to a constant fraction of the length of the modulus n. A prover A who knows the corresponding witness

w to a given common string x as generated by G, can reply to all v possible challenges. On the other hand, a malicious A^* who is *only* given x, could at most reply to a 'small' subset of the possible challenges, since otherwise, A^* would be able to compute collisions efficiently. An attack that only takes x into consideration, is called a *passive attack.*

Please observe the following subtlety regarding *special soundness* and *collision intractability.* In proving collision intractability, we used an approach that may resemble that of special soundness. However, the latter requires that collisions cannot occur for $x \notin RX$. A quick inspection shows that in this example this is not the case. For instance, if n is not an RSA-integer, say a prime or a composite with more than two factors, collisions are possible as well. So what the protocol (A, B) does, is showing that A can express h as the v-th power of some element of $\mathbb{Z}_n^*$, but *not* that n is an RSA-modulus. By removing the restriction that n is an RSA-modulus and that v is relatively prime to $\phi(n)$ from the definition of R, (A, B) would satisfy special soundness. For collision intractability, the definition of the instance generator should remain unchanged.

As for *special honest verifier zero-knowledge*, a simulator is defined as follows. Select $c \leftarrow \mathbb{Z}_v$ and $z \leftarrow \mathbb{Z}_n^*$ and compute $a \leftarrow z^v h^{-c}$. Then x, a, c, z is an accepting conversation with the right distribution. As is the case with the previous example, this protocol is not proved to be secure against adaptive attacks (although it has been conjectured to be), that is, it is not known to be witness hiding or zero knowledge. In an adaptive attack, malicious A^* may first try to extract information about w from A, by choosing the challenges c 'cleverly'. The examples given here are at least secure against honest verifiers. This follows from the fact that the protocols are honest verifier zero knowledge and collision intractable. In [93], witness hiding variants of [106] and [76] are given.

2.4 Basic Theory of Σ-protocols

We develop a basic theory for Σ-protocols needed to support the results in Chapters 3 and 4. The existence of various kinds of collision-intractable, honest verifier zero knowledge Σ-protocols is studied in Section 2.5.

First, we take a closer look at collision-intractability and special soundness. Machines are defined that extract collisions efficiently from successful provers, when given as rewindable black-boxes. As to special soundness, we show how an interactive proof for a given language can be transformed into a new interactive proof, for the same language, that "almost" satisfies special soundness.

Next, we will study honest verifier zero knowledge simulators and derive some important, but elementary, properties. For technical reasons to become apparent in the later chapters, we are interested in increasing or decreasing the challenge length of a given Σ-protocol, while such properties as collision-intractability and honest verifier zero knowledge remain intact. In particular the latter properties are invariant under parallel composition of Σ-protocols. Using a "chaining"-technique, we show that the challenge length can be increased while keeping the length of the first message constant. We conclude this chapter by showing that a Σ-protocol satisfying special honest verifier zero knowledge can be obtained from one having just a normal honest verifier zero knowledge simulator, at the expense of increased communication. Then assuming collision-intractable hash-functions, we show that quasi-signature protocols imply signatures protocols.

2.4.1 Collision Intractability

Let a Σ-protocol (A, B) for relation R be given. Now assume that A^* is an arbitrary polynomially bounded prover that makes the honest verifier B accept the common string x with probability ϵ. What can the verifier conclude about A^*'s knowledge? In the analysis, the prover A^* is given as a rewindable black-box. Our only assumptions on A^* concern its success probability in the protocol (A^*, B) and its running time. For instance, we did not require that A^* is a prover that follows the rules for a prover A in the protocol (A, B). If it were promised that A^* follows the rules, we would simply inspect its knowledge tape. But looking from the verifier's perspective, our conclusions must be based on A^*'s output behaviour, so we may not assume that A^* follows the protocol.

Our analysis will result in constructive methods for "extracting" a collision for the common input x whose expected running time is given as an polynomial function of (the inverse of) A^*'s success probability and A^*'s running time. These methods only work by analyzing A^* as a rewindable black-box, and serve as an apparatus to capture an appropriate notion of "knowledge" of the machine A^*. In real-life, a verifier will, generally speaking, not have such access to a prover. But if the verifier establishes (from executing the protocol) that a given prover has some good probability of success, then the verifier may conclude that the party operating this prover, may by itself deduce a collision with essentially the same probability. This notion was first used by Feige, Fiat and Shamir [66] in their study of proofs of knowledge. We extend their approach to our context of Σ-protocols.

Let H be a 0/1-matrix with n rows and m columns, and let ϵ denote the fraction of 1-entries in H. Suppose $\epsilon > 1/m$. Consider the following "collision-game".

Step 1: Select a random entry in H.

Step 2: If it is a 1-entry, select a different and random entry *from the same* *row.*

Step 3: If this is a 1-entry as well, output "success".

We analyze the success-probability of one iteration of this game. Note that we may not assume, for example, that each row has the same fraction of 1's as H, since we do not know how the 1's are distributed in H. For the analysis we use Jensen's inequality (see for instance [114]). A real-valued function f is called *strictly concave* on interval I, when for each distinct pair of elements in I, the function value of their average is strictly larger than the average of their function values. If additionally f is continuous on I, then Jensen's inequality asserts that

$$\sum_{i=1}^{n} a_i f(x_i) \leq f(\sum_{i=1}^{n} a_i x_i),$$

where $\sum_{i=1}^{n} a_i = 1$, all a_i are positive, and all $x_i \in I$. Equality holds exactly when the x_i's are all the same. We now apply this to the collision-game.

LEMMA 2.1 *Let H be a 0/1-matrix with n rows and m columns, and let ϵ denote the fraction of 1-entries in H. Suppose $\epsilon > 1/m$. Then the probability of success in one iteration of the "collision-game" is greater than or equal to $\epsilon(\epsilon - 1/m)$.*

PROOF. Let e_i denote the number of 1-entries in the i-th row, $i = 1 \ldots n$, and let ϵ_i denote the fraction of 1-entries in the i-th row, that is, $\epsilon_i = e_i/m$. Clearly, the success-probability is equal to

$$\sum_{i=1}^{n} \frac{\epsilon_i}{n}(\epsilon_i - \frac{1}{m}).$$

Now put $a_i = 1/n$, $x_i = \epsilon_i$ and $f(x) = -x(x - 1/m)$ on the interval $[0, 1]$, and apply Jensen's equality. □

The following proposition asserts that from a given prover we can extract a collision with essentially the same success probability that the prover has in the protocol.

PROPOSITION 2.1 *Let a Σ-protocol (A, B) for relation R be given, and let $x \in \{0,1\}^*$. Suppose that A^* is an arbitrary PPT prover such that (A^*, B) succeeds with probability ϵ, on common input x. Let $T_{A^*}(x)$ be A^*'s running time and suppose that $\epsilon > 1/2^{t_B}$. Then there exists a probabilistic algorithm Ext that outputs two accepting conversations x, a, c, z and x, a, c', z' with $c \neq c'$ (that is, a collision), with expected running time polynomial in $T_{A^*}(x)$ and $1/(\epsilon - 1/2^{t_B})$. Ext is allowed to run A^* as a rewindable black-box. The probability is taken over the coin tosses of Ext and A^*.*

PROOF. Define the 0/1-matrix H from Lemma 2.1 as follows. The rows correspond to the possible choices of A^*'s coin flips R_{A^*} and the columns correspond to all possible challenges $c \in \{0,1\}^{t_B}$. Position (R_{A^*}, c) in H is assigned 1 if the conversation produced by (A^*, B) is accepting, provided that A^*'s coin flips are given by R_{A^*} and B's challenge is equal to c. Otherwise, the entry is assigned 0. Note that a pair of distinct 1-entries in H defines a collision and that by our assumption that A^* is given as a rewindable black-box, we can play the collision-game for H. By our assumption on the success probability of (A^*, B), the fraction of 1-entries in H is equal to ϵ. Furthermore, since Ext is allowed to run A^* as a rewindable black-box, and since the pairs (R_{A^*}, c) are distributed uniformly, Ext can repeatedly play the collision-game from Lemma 2.1. This leads to the desired output after $1/(\epsilon^2 - \epsilon/2^{t_B})$ iterations, with high probability. Thus Ext needs expected time polynomial in $T_{A^*}(x)$ and $1/(\epsilon - 1/2^{t_B})$. □

Please note that if the protocol satisfies special soundness and $x \notin RX$, then we must have that $\epsilon \leq 1/2^{t_B}$. On the other hand, we have as a corollary that we can extract a witness for the common string x if the protocol has the collision-property and $x \in RX$:

COROLLARY 2.1 *Let a Σ-protocol (A, B) for relation R be given, having the collision-property, and let $x \in RX$. Suppose A^* is an arbitrary PPT prover such that (A^*, B) succeeds with probability ϵ on common input x. Suppose that $\epsilon > 1/2^{t_B}$. Then there exists a probabilistic algorithm Ext that outputs some $w \in RW(x)$ in expected running time polynomial in $T_{A^*}(x)$ and $1/(\epsilon - 1/2^{t_B})$. The probability is taken over the coin tosses of Ext and A^* and Ext is allowed to run A^* as a rewindable black-box.*

If the protocol is collision-intractable, then an arbitrary PPT prover A^* that is only given the common string x as generated by G, has negligible probability of succeeding in (A^*, B). This is the same as saying that (A, B) is *secure against passive attacks.*

COROLLARY 2.2 *Let a Σ-protocol (A, B) for relation R with generator G be given. Suppose that (A, B) is collision-intractable over R and G. Then, for an arbitrary PPT prover A^* that is given x as generated by $G(1^k)$, (A^*, B) does not succeed, except with negligible probability. The probability is taken over the coin tosses of G, A^* and B.*

The results above imply that if (A, B) is collision intractable and honest verifier zero knowledge, then an arbitrary PPT prover A^* given just x as generated by the generator G, can effectively only answer one challenge. This can be done by preparing a conversation with the simulator.

2.4.2 Approximating Special Soundness

Let a Σ-protocol (P_0, V_0) be given that constitutes an interactive proof for a given language $L \subset \{0,1\}^*$. Suppose now that we are given an input word $x \notin L$. The soundness condition for such proofs requires that in this case a prover should only be able to answer a relatively small subset of the possible challenges that the verifier can choose. But it does not follow that only one challenge can be answered in this case. When designing cryptographic protocols it is often convenient if we may assume that if $x \notin L$, the prover can effectively only answer *one* challenge. Can we transform an interactive proof into a new interactive proof that has this property, thus "approximating" special soundness?

As shown by Damgård [53], we can. The basic idea is to have the prover and verifier agree on two possible challenges, which are isolated by a *cut-and-choose* game. The purpose of such games is to have two mutually mistrusting players agree on a random element from some (large) set. A cut-and-choose game works as follows. The idea is that in each step one party cuts the set of the remaining possibilities (at random) in two halves, while the other party chooses (at random) which half they will continue with. Beforehand they agree on which party will take which role. The distinguishing feature of such games is that if at least one of the parties follows the rules of the game, at least one of the two isolated elements will be random. Furthermore, the choosing party can select an element of his choice before the game starts, an *original choice*, and have it as one of the isolated challenges in the end. If the initial choice is random, the other party cannot distinguish it from the other isolated element.

Note that specifying truly random cuts of the set $\{0,1\}^t$ cannot be done efficiently in general, since the total number N of possible cuts satisfies

$$N = \frac{1}{2}\binom{2^t}{2^{t-1}} > 2^{2^{t-1}-1},$$

and by assumption, each possible cut is chosen with equal probability. To specify such a cut, a number of bits exponential as a function of t is needed. In [90], an efficient method is given to specify cuts that are *statistically close* to random. This technique is called *interactive hashing* (*IH*). Let the set we start from be $\{0,1\}^t$ for some t. A cut is given by a random hyperplane in $\{0,1\}^t$ (when viewed as a vectorspace over $GF(2)$). Then, half of the elements lie in the hyperplane, while the other half lies outside.

We now apply interactive hashing to our purpose of approximating special soundness for the protocol (P_0, V_0). To link the hashing game to our model of Σ-protocols, we introduce the following notation: We assume that a predicate ϕ' is given that takes as input pairs (c, z), where $c \in \{0,1\}^t$, and z has length polynomially related to t. If $\phi'(c, z) = 1$, we say that z is *correct with respect to* c. The interactive hashing takes place between parties P and V, and we assume that for any c, P is capable of computing an z that is correct with respect to c.

1. P selects a random vector $c \in GF(2)^t$ which is kept secret from V. This c is called *P's original choice.*
2. V selects at random $t-1$ linearly independent vectors $h_1, \ldots, h_{t-1}$ in $GF(2)^t$.
3. Repeat the following for $j = 1, \ldots, t-1$: V sends h_j to P, and P sends $b_j = h_j \cdot c$ (the inner product) to V.
4. Both parties compute the two vectors c_0, c_1 with the property that $b_j = h_j \cdot c_i$ for all $j = 1..t-1$. P decides in an arbitrary way which of the two vectors is called c_0 and announces this to V. Of course, P's original choice c is equal to one of c_0, c_1.
5. V chooses $v = 0$ or 1 at random and sends it to P.
6. P computes a z that is correct w.r.t. c_v and sends it to V.

In [90], the following is proved about this procedure:

LEMMA 2.2 *At the end of step 4 above, an arbitrary cheating V^* has no Shannon information about which of c_0, c_1 is P's original choice.*

LEMMA 2.3 *Let P^* be any interactive Turing machine playing the role of P in the interactive hashing. Assume that P^* can return correct z-values for both c_0 and c_1 with probability ϵ, taken over the coin tosses of V and P^*. There is a probabilistic polynomial time machine M using any such P^* as an oracle that can, on input a random $c \in GF(2)^t$, compute an z correct with respect to c in expected running time $T(t, 1/\epsilon)$, where T is a function that is polynomial in both t and $1/\epsilon$. The probability is over the choice of c and the coin tosses of M.*

The following protocol (P_0^{IH}, V_0^{IH}) integrates interactive hashing with the interactive proof (P_0, V_0) for language L. The players are denoted P_0^{IH} and V_0^{IH}, running the programs for P_0 and V_0 as subroutines. Let $x \in L$ and let k denote the binary length of x.

1. P_0^{IH} copies random bits into the random tape of P_0. It then starts running P_0 on input x. This results in a first message a. This is sent to V_0^{IH}.
2. P_0^{IH} and V_0^{IH} go through the interactive hashing process described above, playing the roles of P and V, respectively. The purpose will be to agree on the challenge to be answered. We define the predicate ϕ' by $\phi'(c, z) = \phi(x, a, c, z)$. Thus, a correct z is a value that will make

V_0 accept. At the beginning of step 6 of the interactive hashing, two values c_0, c_1 have already been isolated, and V_0^{IH} has sent a bit v. P_0^{IH} can now obtain a correct z-value by passing c_v to P_0 and relaying the z returned to V_0^{IH}.

3. V_0^{IH} uses V_0 to decide if the conversation (x, a, c_v, z) would lead to accept. If so, V_0^{IH} outputs "accept", otherwise it outputs "reject".

(P_0^{IH}, V_0^{IH}) up to and including step 4 of the interactive hashing is called the *preamble* of (P_0^{IH}, V_0^{IH}). The following lemma shows that we have indeed obtained what we were after:

PROPOSITION 2.2 *Suppose the input $x \notin L$ and consider the probability, taken over the coin flips of V_0^{IH}, that there exists correct answers to both $v = 0$ and $v = 1$ in step 2 above. Let $\delta(k)$ be this probability, maximized over all possible strategies for the prover. Then $\delta(k)$ is negligible in k.*

PROOF. Consider a prover P^* that plays an optimal strategy in order to have V_0^{IH} accept $x \notin L$. This in particular means that we may assume that P^* always sends a fixed first message a which maximizes the number of c-values for which a correct answer exists. Thus the probability that this P^* succeeds is $\delta(k)$. It now follows from Lemma 2.3 that if $\delta(k)$ is not negligible in k, there are infinitely many k's where the fraction of c-values for which correct answers exist is larger than a polynomial fraction. This would contradict the soundness of (P_0, V_0). □

Suppose (P_0, V_0) is honest verifier zero knowledge, with simulator M. The following protocol (M_0^{IH}, V_0^{IH}) describes a machine M_0^{IH} that a prover can use to simulate the preamble of (P_0^{IH}, V_0^{IH}) on input any word x and a private bit b chosen in advance by the prover. This is done in such a way that the prover can complete the protocol successfully in case the verifier's challenge v equals b.

1. M_0^{IH} runs the honest verifier simulator M_0 for (P_0, V_0) on input x to obtain a conversation (x, a, c, z). If this conversation is not an accepting conversation, M_0^{IH} outputs ? and stops.

2. M_0^{IH} executes the interactive hashing with V_0^{IH} using the c from the conversation just produced as the original choice. It names the two values c_0, c_1 isolated by the hashing such that $c = c_b$. Finally, M_0^{IH} outputs z and stops.

We conclude by stating the following property of the protocol (M_0^{IH}, V_0^{IH}) when executed on common input $x \in L$.

PROPOSITION 2.3 *Assume that, on input $x \in L$ and any bit b, we execute (M_0^{IH}, V^*) where V^* is any machine playing the role of V_0^{IH}. This always results in M_0^{IH} outputting a correct answer to $v = b$. If (P_0, V_0) is perfect,* *resp. statistical honest verifier zero-knowledge, then the amount of Shannon information that V^* has about which v-value M_0^{IH} can answer is 0 resp. negligible in k.*

PROOF. The proposition follows from Lemma 2.2, the fact that the c-value used is uniform or statistically close to uniform, and the fact that when $x \in L$, first there exists a correct answer to any c-value and secondly M_0 always produces an accepting conversation. □

Note that although $x \in L$, we may not assume that M_0^{IH} can execute P_0's program: P_0 may have additional auxiliary information which is clearly not given to M_0^{IH}, or P_0 may be unlimited powerful, while M_0^{IH} must run in expected polynomial time.

2.4.3 Simulation Properties

Let (A, B) be a Σ-protocol for relation R and suppose it is honest verifier zero knowledge with simulator M. Let $(x, w) \in R$. By definition M generates conversations with the same distribution as the conversations between the honest prover and the honest verifier. If, additionally, M is a special simulator, we can prove a somewhat stronger result.

Namely, M can perfectly simulate conversations with any verifier that chooses the challenge independently of the prover's first message, provided that the verifier is given to M as a black-box. This is argued as follows. Let $B^*(c)$ denote the verifier who always chooses c as the challenge, for some fixed $c \in \{0,1\}^{t_B}$. It is immediately clear that $M(x, c)$ perfectly simulates $(A, B^*(c))$, where w is private input to A and x is common input to A and $B^*(c)$.

Now consider an arbitrary verifier B^* that chooses the challenge independently from A's first message a, given as a black-box to M. The distribution of (A, B^*) is perfectly simulated by $M(x, c \leftarrow B^*)$, since the latter is just selecting a challenge c according to the distribution of B^*, and independently and uniformly selecting (according to $a(\cdot)$ and subsequently $z(\cdot)$) a conversation where c is the challenge. But this is exactly what happens in the protocol (A, B^*). Thus we have the following lemma.

LEMMA 2.4 *Let (A, B) be a Σ-protocol for relation R satisfying special honest verifier zero knowledge, with simulator M. Let B^* be an arbitrary verifier who chooses the challenge independently of the prover's first message. If M is given B^* as a black-box, it can perfectly simulate the protocol (A, B^*).*

The following lemma is straightforward, however, it is interesting to note that although a Σ-protocol can be just *honest verifier* zero knowledge, it must be witness indistinguishable (against an arbitrary verifier).

LEMMA 2.5 *Let (A, B) be a Σ-protocol for relation R satisfying honest verifier zero knowledge. Then the protocol is also witness indistinguishable.*

PROOF. The assertion follows from the following general observation. Let $x \in RX$ be the common input and consider two different provers A_1 and A_2. Suppose there exist $w_1 \neq w_2$ such that $(x, w_i) \in R$ for $i = 1, 2$. Then give w_i to A_i. Let B^* denote an arbitrary verifier. Now consider the respective coinflips R_{A_1} and R_{A_2}. Saying that (A, B) is (perfect) honest verifier zero knowledge is equivalent to saying that for each accepting conversation the number of possible choices of R_{A_1} and R_{A_2} leading to this particular conversation are (exactly) the same. This is clearly independent of the way in which the challenge is chosen. □

2.4.4 *Manipulating the Challenge Length*

First, we note that the properties we consider in our model (collision-intractability, collision-property, special soundness from Section 2.4.1, (special) honest verifier zero knowledge from Section 2.2.3) are preserved under parallel composition (that is, the execution of the protocol a number of times in parallel) of Σ-protocols. The proof is straightforward.

LEMMA 2.6 *Let (A, B) be a Σ-protocol for relation R and generator G. Suppose that it satisfies special honest verifier zero knowledge and collision intractability (or, the collision-property, or special soundness). Let (A', B') denote the Σ-protocol that consists of any polynomial number of parallel executions of (A, B). Then (A', B') is a Σ-protocol for relation R and generator G , and satisfies (special) honest verifier zero knowledge and collision intractability (resp., the collision-property, special soundness).*

Next, we consider removing (that is, ignoring) bits from the challenge, as in some situations we like to have a smaller challenge length in a given protocol. As an example, we might impose that a fixed number of fixed bit positions in the challenge is set to 0 always. To simulate the new protocol, we can only run a simulator for the original and hope that with good probability the challenge in the output conversation satisfies the same constraints. However, if we ignore too many bits, the honest verifier zero knowledge simulator might require too much time[2]: if we set the last half of the challenge bits to zero always and if the challenge length is, say, equal to $|x|$ bits (where x is the common input), then the probability that the simulator outputs an accepting conversation in which the challenge bits satisfy the same constraint, is negligible as a function of $|x|$. Of course, this problem can be circumvented if we assume that the simulator is special.

[2]This problem does not occur when ignoring $O(\log |x|)$ bits.

Lemma 2.7 *Let (A, B) be a Σ-protocol for relation R and generator G. Suppose that it satisfies special honest verifier zero knowledge and collision intractability (or, the collision-property, or special soundness). Let (A', B') denote the Σ-protocol obtained from (A, B) by ignoring s bits (e.g. setting the last s bits to 0 always) out of the t_B challenge bits ($s < t_B$). Thus, $t_{B'} = t_B - s$. Then (A', B') is a Σ-protocol for relation R and generator G and satisfies special honest verifier zero knowledge and collision intractability (resp., the collision-property, special soundness).*

Combining these lemmas, we see that for the case of special honest verifier zero knowledge, we can in fact construct any challenge size we want.

Lemma 2.8 *Let (A, B) be a Σ-protocol for relation R and generator G. Suppose that it satisfies special honest verifier zero knowledge and collision intractability (or, the collision-property, or special soundness). Let t be any polynomially bounded function of t_B. Then there exists a Σ-protocol (A', B') for relation R and generator G, satisfying special honest verifier zero knowledge and collision intractability (resp., the collision-property, special soundness), such that $t_{B'} = t$.*

In the lemmas above, the challenge size can be increased at the cost of a larger size of the initial message. Suppose now that (A, B) is a signature protocol for relation R and generator G (see Section 2.2.7). Using a "chaining technique", the following proposition shows that, we can increase the size of the challenge while keeping the size of the initial message constant. We use the fact that a signature protocol satisfies $t_A < t_B$.

Proposition 2.4 *Suppose there exists a signature protocol (A, B) for relation R with generator G. Let k be the security parameter. Then there exists an efficient transformation that takes (A, B) to a signature protocol (A', B') for R and G, satisfying $t_{B'} = t(k)$ for any fixed polynomial $t(\cdot)$.*

Proof. Without loss of generality, we may assume that $t_A + 1 = t_B$. Let $t_{B'} = t(k)$ be the desired challenge length, a fixed polynomial $t(\cdot)$. For convenience, put $t \equiv t(k)$. The claimed transformation runs as follows:

Key generation: The prover A' runs $G(1^k)$ to obtain $(x, w) \in R$. Common input to A' and B': x. Private input to A': w.

Move 1: A' computes an initial message a as generated in the protocol (A, B), and sends a to B'.

Move 2: The verifier B' selects $b_1, \ldots, b_t \leftarrow \{0, 1\}$, where $t = t(k)$ and sends these to A'.

Move 3: Put $a_1 = a$. By invoking prover A on input (x, w), A' computes the following values. For $i = 1 \ldots t$, A' computes an accepting (in the protocol (A, B)) conversation (x, a_i, c_i, z_i) where $c_i \leftarrow a_{i+1} | b_i$ for

$i < t$ and $c_t \leftarrow 0|\cdots|0|b_t$. A' sends the a_i's, c_i's and z_i's to B'. The conversation is accepting in (A', B') if and only if (for $i = 1 \dots t$) $a_1 = a$, $c_i = a_{i+1}|b_i$ for $i < t$, $c_t = 0|\cdots|0|b_t$ and the conversation (x, a_i, c_i, z_i) is accepting conversations in (A, B).

By construction, the challenge length t for (A', B') can be chosen what we want it to be, provided $t = \text{poly}(k)$. Suppose now that we are given two accepting conversations in (A', B') for some public string x with the same first message a, but with different challenges $(b_1, \dots, b_t)$ and $(b'_1, \dots, b'_t)$. Let, for $j = 1 \dots t$, (a_j, c_j, z_j) and (a'_j, c'_j, z'_j) be the respective replies in those conversations in (A', B'), and let i be an index such that $b_i \neq b'_i$. Clearly, this implies that $c_i \neq c'_i$. Take i to be the smallest index such that $c_i \neq c'_i$. If $i = 1$, we have a collision in (A, B) with respect to x, as by definition of (A', B'), we must have $a_1 = a'_1 = a$. On the other hand, if $i > 1$, c_{i-1} must be equal to c'_{i-1}. Thus we have $a_i|b_{i-1} = a'_i|b'_{i-1}$. But then $a_i = a'_i$ and we have a collision (x, a_i, c_i, z_i), (x, a'_i, c'_i, z'_i) in (A, B). Therefore, (A', B') is collision-intractable over R and G.

As for special honest verifier zero-knowledge of (A', B'), we now exhibit a special simulator M' for (A', B'), that runs the simulator M for (A, B) as a subroutine. M' starts by receiving a public string x and uniformly chosen challenge bits $b_1, \dots, b_t$ as input. It proceeds by putting $c_t = 0|\cdots|0|b_t$, and feeding x and c_t to M. After M has output an accepting conversation x, a_t, c_t, z_t in (A, B), M' repeats the following for $i = t-1 \dots 1$. Put $c_i = a_{i+1}|b_i$, feed x and c_i to M and receive an accepting conversation x, a_i, c_i, z_i from M. It is now clear that M' generates accepting conversations in (A', B') with respect to x, with exactly the same distribution as the conversations with the honest verifier in (A', B'). □

Note that each bit of the new challenge size costs one conversation in the original protocol. It is immediately clear that this can be improved to one conversation per $t - t_B$ bits of the new challenge. More efficient transformations of this type exist for protocols (A, B) based on one-way group homomorphisms (see Sections 2.5.1 and 2.5.2).

2.4.5 Trade-offs

At the expense of increased communication complexity, we can compile Σ-protocols satisfying honest verifier zero knowledge and collision intractability (or special soundness) into ones that additionally have a special simulator.

PROPOSITION 2.5 *Let (A, B) be a Σ-protocol for relation R with generator G. Suppose that (A, B) is honest verifier zero-knowledge, and collision-intractable over R and G (or has the collision-property, or satisfies special soundness). Then (A, B) can be compiled into a Σ-protocol (A', B') for relation R and generator G, that is also collision-intractable over R and*

G (resp., has the collision-property, satisfies special soundness) but that additionally satisfies special *honest verifier zero-knowledge.*

PROOF. Let l of polynomial size in the security parameter k, and put $l \leftarrow t_{B'}$. We abbreviate t_B to t. The prover A' and the verifier B' run A and B resp. as subroutines. The claimed transformation works as follows.

Key generation: The prover A' runs $G(1^k)$ to obtain $(x, w) \in R$. Common input to A' and B': x. Private input to A': w.

Move 1: A' computes l initial messages $a_1, \ldots, a_l$, each of which generated as in the protocol (A, B), and sends these to B'.

Move 2: The verifier B' selects $c = (b_1, \ldots, b_l) \leftarrow \{0,1\}^l$ and sends c to A'.

Move 3: A' selects $\beta_1, \ldots, \beta_l \leftarrow \{0,1\}^{t-1}$, and puts $c_1 = \beta_1|b_1, \ldots, c_l = \beta_l|b_l$. A' finally computes $z_1, \ldots, z_l$ such that (x, a_i, c_i, z_i) is an accepting conversation in the protocol (A, B), for $i = 1 \ldots l$. A' sends the c_i's and z_i's to B'. The conversation is accepting in (A', B') if and only if the last bit of c_i is equal to b_i and a_i, c_i, z_i is an accepting conversation in the protocol (A, B), for $i = 1 \ldots l$.

First, we show that collision-intractability is preserved under this compilation. Suppose we are given two accepting conversations in (A', B'), with the same initial message $a = (a_1, \ldots, a_l)$, but with different challenges $c = (b_1, \ldots, b_l)$ and $c' = (b'_1, \ldots, b'_l)$. Let the respective replies in those conversations be $\beta_1, \ldots, \beta_l$, $z_1, \ldots, z_l$ and $\beta'_1, \ldots, \beta'_l$ $z'_1, \ldots, z'_l$, and let i be an index such that $b_i \neq b'_i$. Then clearly, $x, a_i, \beta_i|b_i, z_i$ and $x, a_i, \beta'_i|b'_i, z'_i$ are two accepting conversations in (A, B) for the same common string x, with $\beta_i|b_i \neq \beta'_i|b'_i$. We conclude that (A', B') is collision-intractable over R and G (resp. satisfies special soundness).

The special simulator M' for (A', B') runs (A, B)'s simulator M as a subroutine, and is defined as follows. We run the simulator M for (A, B) a sufficient number of times on the common input x such that we have gathered l conversations in (A, B) where the last bit of the challenges is 0, and l ones where this bit is 1. The probability of failure must be negligible. When given $c = (b_1 \ldots b_l) \in \{0,1\}^l$, M' just selects for each bit b_i in c a conversation from those generated as above with the last bit equal to b_i.

Let $L \equiv \max(l, |x|)$. We use *Chernoff's Bound* to show that running M $3L$ times is sufficient. Let $X_1, \ldots, X_n$ be independent random variables, taking on values in $\{0,1\}$. Let $p \leq 1/2$, and suppose that the probability that X_i takes on value 1 is equal to p, for each i. Then, for all δ with $0 < \delta \leq p(1-p)$, Chernoff's bound says that

$$\text{Prob}\left(\left|\frac{\sum_{i=1}^{n} X_i}{n} - p\right| > \delta\right) < 2 \cdot \exp^{-\frac{\delta^2}{2p(1-p)} \cdot n}.$$

We apply this inequality to our case as follows. Let m_1 denote the number of times that M outputs a conversation with least significant bit of the challenge equal to 1. We have $0 \leq m_1 \leq 3L$. Note that if $1/3 \leq m_1/(3L) \leq 2/3$, then we have gathered those conversations we are after. Put $\delta = 1/6$, $p = 1/2$, and $n = 3L$ in the Chernoff's bound. Then we see that the probability ρ that we do not succeed satisfies

$$\rho < 2 \cdot \exp^{-\frac{1}{6} \cdot L},$$

which is negligible as a function of $|x|$. This concludes our description of the special simulator M' for (A', B'). Finally, it is clear that the honest verifier in (A', B') receives l conversations from (A, B) where each of these conversations is according to conversations with an honest verifier in (A, B). By construction, it is clear that M' does the same: using simulator M to select honest verifier conversations in (A, B) according to the last bit in the challenge, while the selection is according to uniform bits. □

REMARK 2.1 *The transformation from Proposition 2.5 can be modified such that* $\log |x|$ *challenge bits of the* c_i*'s (from the conversations in* (A, B)*) are "reserved" for the bits in the challenge* c *(from the protocol* (A', B')*. This reduces the communication in* (A', B')*, but increases the running time of the simulator.*

When given a collision-intractable hash-function (a function that maps elements from a large set onto elements of a much smaller set, such that it is infeasible to find two different inputs that map to the same value, see for instance [114]), quasi-signature protocols (see Section 2.2.7) can be converted into signature protocols. The protocol works just like the one from Proposition 2.4, except that in the current case we may not assume that $t_A < t_B$. The hash-function simply maps the a_i-values from the protocol in Proposition 2.4 to bit strings of length $t_B - 1$. Of course, we must have $t_B >> 1$. If the protocol we start from does not have a *special* simulator, we first apply Proposition 2.5.

DEFINITION 2.1 *Let* (A, B) *be a quasi-signature protocol for relation* R *and generator* G *and let* $\mathcal{H}$ *be a family of collision intractable hash functions. Then* $R_{\mathcal{H}}$ *consists of pairs* $((x, h), w)$ *where* $(x, w) \in R$ *and* $h \in \mathcal{H}$ *has output length* $t_B - 1$*. The generator* $G_{\mathcal{H}}$ *first runs* $G(1^k)$ *to get* $(x, w) \in R$*. Then it selects* $h \in \mathcal{H}$ *with the desired output length.*

PROPOSITION 2.6 *Suppose there exists a quasi signature protocol* (A, B) *for relation* R *and generator* G *and that a family* $\mathcal{H}$ *of collision intractable hash functions exists. Then there exists a signature protocol* (A', B') *for* $R_{\mathcal{H}}$ *and* $G_{\mathcal{H}}$*.*

PROOF. If needed, first apply Proposition 2.5. So we may assume that (A, B) has a *special* simulator. Consider the following protocol. Let t denote $t_A + 1$.

Key generation: The prover A' runs $G_{\mathcal{H}}(1^k)$ to obtain $(x, w) \in R$ and $h \in H$. Common input to A' and B': x, h. Private input to A': w.

Move 1: A' computes an initial message a as generated in the protocol (A, B), and sends a to B'.

Move 2: The verifier B' selects $b_1, \ldots, b_t \leftarrow \{0, 1\}$ and sends these to A'.

Move 3: A' computes $a_1, \ldots, a_t$ and $z_1, \ldots, z_t$ such that a_i, c_i, z_i is an accepting conversation in the protocol (A, B), for $i = 1 \ldots t$. Here $a_1 = a$ and $c_1 = h(a_2)|b_1, \ldots, c_{t-1} = h(a_t)|b_{t-1}$ and $c_t = 0|\cdots|0|b_t$. A' sends the a_i's, c_i's, and z_i's to B'. The conversation is accepting in (A', B') if and only if $a_1 = a$, $c_i = h(a_{i+1})|b_i$ for $i = 1 \ldots t-1$, $c_t = 0|\cdots|0|b_t$ and all conversations x, a_i, c_i, z_i are accepting conversations in (A, B).

The remaining part of the proof is similar to that of Proposition 2.4, except that in addition the collision-intractability of h is used. □

2.5 Existence

We demonstrate that Σ-protocols satisfying special honest verifier zero knowledge and collision intractability exist under such assumptions as the existence of one way group homomorphisms, claw-free pairs of trapdoor permutations, the difficulty of computing discrete logarithms, RSA inversion or factoring integers. Since in all these results we can achieve that the challenge size is larger than the size of the first message, we state the results in terms of signature protocols. We also introduce *special* claw-free pairs of trapdoor permutations and *special* one-way group isomorphisms. Some of our most efficient constructions are based on these functions. In Chapters 3 and 4 we frequently assume the existence of a Σ-protocol with such properties as (special) honest verifier zero knowledge and collision intractability (or special soundness). This chapter provides the ways that are currently known to us, to realize such protocols.

We emphasize that the Σ-protocols presented in this section are the building blocks to our results in Chapters 3 and 4. Please recall that Σ-protocols as such do not constitute any reasonable cryptographic service or application [3]. From a security point of view, for instance, honest verifier zero knowledge does not guarantee protection in an adaptive scenario (that is,

[3]However, Σ-protocols satisfying honest verifier zero knowledge and collision-

a setting where (computationally bounded) players may deviate arbitrarily from the protocols). From the results in Chapters 3 and 4 on partial proofs, witness hiding, (general) zero knowledge protocols, identification schemes, secret ballot elections and digital signatures, the usefulness of Σ-protocols in adaptive settings becomes clear.

2.5.1 One-Way Group Homomorphisms

A family of one-way group homomorphisms $\mathcal{F}$ with generator G is a collection of triples (f, K, L) where K and L are groups and $f : K \to L$ is a group homomorphism. Furthermore, if we let k_f denote the number of bits needed to represent an element of L, it is required that for $w \in K$, $f(w)$ can be computed in time polynomial in k_f. Similarly, the group operations (multiplication/addition and inversion) in K and L are required to be computable in time polynomial in k_f. On input 1^k (where k is the security parameter), the PPT algorithm G outputs an element $(f, K, L) \in \mathcal{F}$ with $k_f = k$ in a suitable description. Also, it must be possible, in probabilistic polynomial time, to sample uniformly from K and to decide membership of K and L. Finally, $\mathcal{F}$ is *one-way*, if no PPT algorithm, receiving as input (f, K, L) (as generated by G) and x (where $x = f(w)$ for uniformly chosen w) can compute $w' \in K$ such that $f(w') = x$, except with negligible probability in k.

Typical examples of one-way group homomorphisms are discrete exponentiation modulo a prime (under an intractability assumption for computing discrete logarithms, see also Section 2.3.2), or discrete exponentiation modulo a composite (under an intractability assumption for factoring integers, see also Section 2.3).

We will now show that one-way group homomorphisms (OWGH's) imply signature protocols.

DEFINITION 2.2 *Let $\mathcal{F}$ be a family of OWGH's. Then $R_{\mathcal{F}}$ is the binary relation consisting of pairs $((f, K, L, x_1, \ldots, x_{k+1}), (w_1, \ldots, w_{k+1}))$, where $(f, K, L) \in \mathcal{F}$ and $f(w_i) = x_i$. $G_{\mathcal{F}}$ is the generator that on input 1^k runs G to get (f, K, L) and then selects $w_1, \ldots, w_{k+1}$ at random from K and finally computes $x_i = f(w_i)$. Here k denotes the number of bits needed to encode elements of L, and G is a generator for $\mathcal{F}$.*

PROPOSITION 2.7 *Suppose $\mathcal{F}$ is a family of one-way group homomorphisms with generator G. Then there exists a signature protocol for $R_{\mathcal{F}}$ and $G_{\mathcal{F}}$.*

PROOF. Consider the following protocol.

---

intractability may be viewed as digital identification schemes secure against random challenge attacks. See Section 3.6.

Key-generation: A puts $((f, K, L, x_1, \ldots, x_t), (w_1, \ldots, w_t)) \leftarrow G_{\mathcal{F}}(1^k)$ with $t = k+1$. Common input to A and B: $x = (f, K, L, x_1, \ldots, x_t)$. Private input to A: $w = (w_1, \ldots, w_t)$.

Move 1: A selects $u \leftarrow K$, computes $a \leftarrow f(u)$ and sends a to B.

Move 2: B selects $c_1, \ldots, c_t \leftarrow \{0, 1\}$ and sends c to A.

Move 3: A computes $z \leftarrow uw_1^{c_1} \cdots w_t^{c_t}$ and sends z to B, who accepts if and only if $f(z) = ax_1^{c_1} \cdots x_t^{c_t}$.

The protocol given above is clearly complete with probability 1. Special honest verifier zero knowledge is clear by standard arguments: first choose z and $c_1, \ldots, c_t$ at random, then use this to compute an a-value. It is also clear that the challenge is one bit longer than the first message from the prover. Thus, only the collision intractability property remains to be argued. Let a collision $(x, a, c_1, \ldots, c_t, z)$, $(x, a, c_1', \ldots, c_t', z')$ be given. Define $\sigma = z/z'$ and $\gamma_i = c_i - c_i'$, for $i = 1 \ldots t$. Then $f(\sigma) = x_1^{\gamma_1} \cdots x_t^{\gamma_t}$ where the γ_i's are equal to 0, -1 or 1 and at least one of them is non-zero. Now suppose that there exists a probabilistic polynomial time algorithm E that receives x as generated by $G_{\mathcal{F}}$ and outputs a collision for x with non-negligible probability. Given a triple (f, K, L) as output by G and a random member $Y \in L$ we now show how to invert f at Y, running E as a subroutine as follows. Select i at random from $\{1, \ldots, t\}$ and $w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_t$ at random from K. Next, put $x_i = Y$ and $x_j = f(w_j)$ for $1 \leq j \leq t$ and $j \neq i$. Finally, put $x = (f, K, L, x_1, \ldots, x_t)$ and present x to E. We may assume that E outputs a collision for x with non-negligible probability. With notation as above, we have that $\gamma_i \neq 0$ with probability at least $1/t$. This is because x gives no information about i. But now we have

$$f(\sigma^{\gamma_i} w_1^{-\gamma_i\gamma_1} \cdots w_{i-1}^{-\gamma_i\gamma_{i-1}} w_{i+1}^{-\gamma_i\gamma_{i+1}} \cdots w_t^{-\gamma_i\gamma_t}) =$$

$$f(\sigma)^{\gamma_i} f(w_1)^{-\gamma_i\gamma_1} \cdots f(w_{i-1})^{-\gamma_i\gamma_{i-1}} f(w_{i+1})^{-\gamma_i\gamma_{i+1}} \cdots f(w_t)^{-\gamma_i\gamma_t} =$$

$$f(\sigma)^{\gamma_i} x_1^{-\gamma_i\gamma_1} \cdots x_{i-1}^{-\gamma_i\gamma_{i-1}} x_{i+1}^{-\gamma_i\gamma_{i+1}} \cdots x_t^{-\gamma_i\gamma_t} = x_i^{\gamma_i^2} = x_i = Y.$$

Thus we have a contradiction with our assumption that the group homomorphisms in $\mathcal{F}$ are one-way. □

2.5.2 *Special One-Way Group Isomorphisms*

We introduce special one-way group isomorphisms.

DEFINITION 2.3 *Let $\mathcal{F}$ be a family of one-way group homomorphisms as before. $\mathcal{F}$ is called a family of special one-way group isomorphisms if the following two additional requirements are satisfied.*

1. *All $(f : K \longrightarrow L) \in \mathcal{F}$ are* isomorphisms.

2. *There exist probabilistic polynomial time algorithms D_1 and D_2 with the following properties. Algorithm D_1 takes as input an element $(f : K \longrightarrow L) \in \mathcal{F}$ and outputs an odd prime T_f (typically depending on the element). Algorithm D_2 takes as input $(f : K \longrightarrow L) \in \mathcal{F}$, any $g \in L$ and T_f, and outputs $\delta \in K$ such that $f(\delta) = g^{T_f}$.*

EXAMPLE 2.1 *We now give examples of special one-way group isomorphisms. The first example is discrete exponentiation in a group G_q of prime order q, which is trivially an isomorphism. When implemented as in Section 2.3, the collection of groups $\{G_q\}$ (groups G_q of prime order q) has an efficient test of membership. Defining f as discrete exponentiation (taking the exponent modulo q) with respect to a fixed generator of G_q, we have a family of special one-way group isomorphisms. Next consider RSA. Let n be an RSA-modulus and let the prime v be such that it does not divide $\phi(n)$. A family of RSA functions $RSA_{n,v}$ based on this, will certainly suffice (see Section 2.3.3. If we furthermore require that v is larger than n, than exponentiation modulo n is an isomorphism anyway, with $T = v$.*

It turns out that we can do the transformation from Proposition 2.7 more efficiently for special one-way group isomorphisms. There, we have to introduce a new x_i for each bit of the new challenge. By Lemma 2.9 below we have that if we choose the c_i from $[0, T-1]$ instead of $\{0,1\}$ in the protocol of Proposition 2.7 and use special one-way isomorphisms, it satisfies collision-intractability and special honest verifier zero knowledge as well. But moreover, we only need to introduce $t/|T|$ x_i's. This results is important in Section 4.5 for the construction of secure and efficient signatures. We have the following proposition.

PROPOSITION 2.8 *Let $\mathcal{F}$ be a family of special one-way group isomorphisms, and let k be the security parameter. Then there exists a signature protocol where the size of the common input is $(t/|T|)k$ bits and t is the challenge length. In particular, if t and $|T|$ are linear as a function of k, then the length of conversations, including the common string, is linear as a function of k.*

The proof of Proposition 2.8 follows the lines of the proof of Proposition 2.7. It differs at one point, which is filled in by the following lemma.

LEMMA 2.9 *Let f, K, L and the prime T be as above. Suppose we are given $\sigma \in K$, $x_i \in L$ and a non-zero integer γ_i such that $f(\sigma) = x_i^{\gamma_i}$ and T does not divide γ_i. Then we can efficiently compute $\Delta \in K$ such that $f(\Delta) = x_i$.*

PROOF. Let s and l be integers such that $s\gamma_i = 1 + lT$. Let δ be such that $f(\delta) = x_i^T$. By our assumptions on $\mathcal{F}$, δ can be efficiently computed. Let Δ denote $\delta^{-l}\sigma^s$. Then $f(\Delta) = f(\delta^{-l}\sigma^s) = f(\delta)^{-l}f(\sigma)^s = x_i^{-lT}x_i^{s\gamma_i} = x_i$. □

We now give two examples of *signature protocols* based on special one-way group isomorphisms. In both cases, we can get $|T|$ linear as a function of k. First, based on Schnorr's protocol [106], we have the following signature protocol (A', B'), which we will refer to as DL^*. Fix any $s > 1$ polynomially bounded in $|q|$. The protocol obeys the construction from Proposition 2.8. We are given a group G_q of prime order q, where computing discrete logarithms is hard, and random $g \in G_q$. Let the set-up be as in Section 2.3.2.

Key-generation: G is as in Section 2.3.2, except that this time it selects $w_1, \ldots, w_s \leftarrow \mathbb{Z}_q$, and computes $h_1 \leftarrow g^{w_1}, \ldots, h_s \leftarrow g^{w_s}$. A runs G, and obtains $x = (G_q, g, h_1, \ldots, h_s)$ and $w = (w_1, \ldots, w_s)$. Common input to A' and B': x. Private input to A': w.

Move 1: A' selects $u \in \mathbb{Z}_q$ and computes $a \leftarrow g^u$. A' sends the a to B'.

Move 2: B' selects $c_1, \ldots, c_s \leftarrow \mathbb{Z}_q$ and sends, for $i = 1 \ldots s$, the c_i's to A'.

Move 3: A' computes $z \leftarrow u + c_1 w_1 + \cdots c_s w_s \bmod q$ and sends z to B', who accepts if and only if $g^z = a h_1^{c_1} \cdots h_s^{c_s}$.

Note that by choosing s large enough in DL^*, we get any challenge length we may desire. For instance, if elements of G_q can be encoded by a number of bits linear in $|q|$ (as is the case when G_q is the unique subgroup of order q in $\mathbb{Z}_p^*$, where, for instance, $p = 2q + 1$), then to achieve $t = t_{B'}$ linear in $|q|$, s can be a constant. In the latter case, the length of the conversations, including the common string is linear as a function of k.

Of course, we can do a similar construction based on RSA. The value $T = v$, where v is the RSA-exponent, can be chosen of length linear in $|n|$, where n is the RSA-modulus. But there exists an interesting alternative. With some modifications, the scheme (A, B) based on RSA from Section 2.3.3, can be turned into a signature protocol (A', B'), which we will refer to as RSA^*. We are given an odd integer n that is the product of two distinct primes p and q, and a prime v that does not divide $\phi(n)$. Let e be polynomially bounded in $|n|$. If we want to achieve a given challenge length $t_{B'}$ in the following protocol, we choose e minimal such that $v^e \geq t_{B'}$. Note that if the prime v satisfies $v > n$, then exponentiation to the v-th (or v^e-th) power modulo n is guaranteed to be a bijection. Let the set-up be as in Section 2.3.2.

Key-generation: A' runs G and gets (x, w) where $x = (n, v, h, e)$ and $w \in \mathbb{Z}_n^*$ is such that $h = w^{v^e} \bmod n$ (notice the slight change in the definition of the generator G). Common input to A' and B': x. Private input to A': w.

Move 1: A' selects $u \in \mathbb{Z}_n^*$ and computes $a \leftarrow u^{v^e} \bmod n$. A' sends a to B'.

Move 2: B' selects $c \leftarrow \mathbb{Z}_{v^e}$ and sends c to A'.

Move 3: A' computes $z \leftarrow w^c u \bmod n$ and sends z to B', who accepts if and only if $z^{v^e} = ah^c \bmod n$.

We indicate that the ability to compute collisions in the protocol RSA^* implies the ability to compute v-th roots modulo n. We are given a random $h \in \mathbb{Z}_n^*$ and present h and n to the collision finding algorithm. We compute $h^{\frac{1}{v}}$ as follows. From any collision it outputs, we can compute $z \in \mathbb{Z}_n^*$ and $c \in \mathbb{Z}_{v^e}$ such that $z^{v^e} = h^c \bmod n$. Write $c = v^f c'$ with $\gcd(v, c') = 1$. Then we must have $z^{v^{e-f}} = h^{c'} \bmod n$, since taking v-th powers modulo n is a bijection. Raising both sides to the power $1/c' \bmod v$ gives us the desired value. The protocol RSA^* plays an important role in our results from Section 4.7. If we want to achieve $t_{B'} = v^e$ linear in the security parameter k, we just choose e large enough. In that case, the length of the conversations, including the common string is linear as a function of k.

Our interest in special one-way group isomorphisms comes additionally from the fact that we can build very efficient Σ-witness hiding proofs of knowledge from them (see Proposition 2.9). These will satisfy special soundness and honest verifier zero knowledge. Such protocols can also be seen as very efficient and secure identification schemes, as we will indicate later. Later on, in Chapter 3, we will show how protocols based on these functions lead to very efficient general zero knowledge proofs.

Let $g \in L$. Now consider the following function $\hat{f}_g$.

$$\hat{f}_g : [0, T-1] \times K \longrightarrow L$$

$$(w_1, w_2) \mapsto g^{w_1} f(w_2)$$

DEFINITION 2.4 *We define the relation $\hat{R}_{\mathcal{F}}$ as follows. It consist of all pairs (x, w) where $x = (f, K, L, T, g, h)$ and $w = (w_1, w_2)$ such that $\hat{f}_g(w_1, w_2) = h$. The corresponding generator $\hat{G}_{\mathcal{F}}$ runs the generator G for $\mathcal{F}$, computes T, selects w_1, w_2 and g at random and finally computes h as $h \leftarrow \hat{f}_g(w_1, w_2) = h$.*

PROPOSITION 2.9 *Let $\mathcal{F}$ be a family of special one-way group isomorphisms and let k be a security parameter. Then there exists a Σ-protocol (A, B) for relation $\hat{R}$ and generator $\hat{G}$ that satisfies special honest verifier zero knowledge and the collision property. Furthermore, the protocol is collision-intractable and witness-hiding.*

PROOF. Consider the following protocol.

Key-generation: A runs $\hat{G}_{\mathcal{F}}$ to get $x = (f, K, L, T, g, h)$ and $w = (w_1, w_2)$. Common input to A and B: x. Private input to A: w.

Move 1: A selects $u_1 \leftarrow [0, T-1]$ and $u_2 \leftarrow K$, computes $a \leftarrow \hat{f}_g(u_1, u_2)$ and sends a to B.

Move 2: B selects $c \leftarrow [0, T-1]$ and sends c to A.

Move 3: A computes $\delta \in K$ such that $f(\delta) = g^T$, $z_1 \leftarrow u_1 + cw_1 \bmod T$ and $z_2 \leftarrow \delta^{u_1 + cw_1 \text{ div } T} u_2 w_2^c$. Finally, A sends $z = (z_1, z_2)$ to B, who first computes δ. Then B accepts if and only if $\hat{f}_g(z_1, z_2) = ah^c$.

The *honest verifier simulator* clearly chooses c and z_1 at random from $[0, T-1]$ and z_2 at random from K, and computes $a \leftarrow \hat{f}_g(z_1, z_2)h^{-c}$. For the *collision-property*, note that from a collision we can always compute non-zero α and β from $[-T+1, T-1]$ and $\gamma \in K$ such that $g^\alpha f(\gamma) = h^\beta$. Now compute β' such that $\beta\beta' \equiv 1 \bmod T$ and $\theta \in K$ such that $f(\theta) = h^T$. Put

$$w_1' = \alpha\beta' \bmod T,$$

$$w_2' = \gamma^{\beta'} \delta^{\alpha\beta' \text{ div } T} \theta^{-(\beta\beta' \text{ div } T)}.$$

Then we have $\hat{f}_g(w_1', w_2') = h$. *Collision-intractability*: suppose there exists an efficient algorithm for generating collisions, when given x as generated by $\hat{G}_{\mathcal{F}}$. It can be turned into an efficient algorithm for inverting the functions in $\mathcal{F}$ as follows. First note that from a collision output by that algorithm, we can compute a witness $w' = (w_1', w_2')$ for x. But since for each g and h in L, there are exactly T pairs (w_1, w_2) such that $\hat{f}_g(w_1, w_2) = h$ and since x gives no information about which pair was used in the generation, (w_1', w_2') will be different from (w_1, w_2) with probability $(T-1)/T$. But in that case, we have $\hat{f}_g(w_1, w_2) = \hat{f}_g(w_1', w_2')$. This implies that $g^{w_1 - w_1'} = f(w_2^{-1} w_2')$, where $w_1 \neq w_1'$ since f is an isomorphism and $(w_1, w_2) \neq (w_1', w_2')$. But since we must have that T does not divide $w_1 - w_1'$, we can compute $\Delta \in K$ such that $f(\Delta) = g$ by Lemma 2.9. But this means that the one-wayness of $\mathcal{F}$ is contradicted. As for *witness hiding*, note that for each conversation (x, a, c, z) and for each such pair (w_1, w_2), there are T pairs (u_1, u_2) that fit with the conversation. Thus, by the random choice of u_1 and u_2, the protocol is witness indistinguishable. Collision-intractability and witness-indistinguishability now imply that the protocol is witness hiding. □

If we would like to use the protocol from Proposition 2.9 for the purpose of secure identification, we only need to assume that the challenge length $|T|$ is large enough, say linear in the security parameter k: an efficient attacker V^* that succeeds in extracting enough information from A (by querying A in any adaptive fashion), to later pass successfully as A, must know a witness by the collision-property (see Corollary 2.1, and note that $1/2^{|T|}$ is negligible as a function of k) *and* the fact that x has a witness (since it is generated according to $\hat{G}_{\mathcal{F}}$). The existence of such an attacker would contradict the fact that the protocol is witness hiding.

Interestingly, if we define $\mathcal{F}$ as in Example 2.1 (taking q large enough in case of discrete logarithms, and taking v large enough in case of RSA) then we get Okamoto's identification schemes [93] based on discrete logarithms and RSA, respectively. Thus our protocol can be seen as a generalization of Okamoto's result.

In the above nothing is required about what happens when the prover presents x such that for no w, it holds that (x, w) is in the relation. And we may not exclude the possibility that for some x of that kind, the prover will be successful! Nothing can be said about what the verifier may conclude from a successful run of (A, B), if it is not given that x was generated according to the prescribed generator. In Section 3.3, we will apply this protocol to special types of bit commitments. There, the prover must show that the contents of a commitment satisfies certain constraints and that the prover can "open" the commitment. For situations as the latter, we will need that the protocol is a proof of knowledge. Therefore, we state the following proposition.

PROPOSITION 2.10 *Let $\mathcal{F}$ be a family of special one-way group isomorphisms. Suppose $\mathcal{F}'$ is a family of special group isomorphisms (dropping the requirement on one-wayness) such that the length of T is linear in k, $\mathcal{F} \subset \mathcal{F}'$ and membership of $\mathcal{F}'$ can efficiently be tested, then the protocol (A, B) from Proposition 2.9 is a proof of knowledge for relation $\hat{R}_{\mathcal{F}'}$ satisfying special soundness and special honest verifier zero knowledge. The length of the conversations is linear as a function of k. Naturally, in order for (A, B) to be witness hiding, the instances should still be generated according to $\hat{G}_{\mathcal{F}}$.*

PROOF. First note that the protocol is complete for relation $\hat{R}_{\mathcal{F}'}$, since the one-way property is irrelevant there. The collision-property remains intact as well. By incorporating an efficient check, whether a given $f \in \mathcal{F}$, in the verification of a conversation, we then conclude that the protocol satisfies special soundness. The length of T guarantees that if A^* has non-negligible probability of making B accept in (A^*, B), then the knowledge extractor from Corollary 2.1 runs efficiently. □

$\mathcal{F}'$ can, for instance, be realized by taking discrete exponentiation in G_q, where G_q is the unique subgroup of order q in $\mathbb{Z}_p^*$ and, say, $2q + 1 = p$. In this case we have $\mathcal{F}' = \mathcal{F}$. Another example of $\mathcal{F}'$ is exponentiation to the v-th power modulo an integer n ($v > n$, v prime). Then $\mathcal{F}$ is the subset of functions where n is an RSA-modulus.

2.5.3 Claw-Free Pairs of Trapdoor Permutations

Claw-free trapdoor permutations were defined in [69] and [70]. Informally, a pair $f = (f_0, f_1)$ of permutations f_0 and f_1 of a set D_f is called *claw-free* if it is infeasible to compute, given only f and D_f,

$$b, b' \in D_f \text{ such that } f_0(b) = f_1(b').$$

The following notations are from [70]. Let f_0 and f_1 be two permutations of the same domain D. For any $a \in \{0,1\}^*$ and for any $b \in D$, define the function $f_{[a]}(b)$, composed of f_0 and f_1, as follows.

$$f_{[a]}(b) = f_{a_1}(f_{a_2}(\cdots(f_{a_t}(b))\cdots)),$$

where for some positive integer t, $a = a_1|\cdots|a_t$ and $a_i \in \{0,1\}$ for $i = 1\ldots t$. Note that for each fixed $a \in \{0,1\}^*$, $f_{[a]}(\cdot)$ permutes D_f. When we have a pair of such permutations f_0 and f_1, we will sometimes simply say "the pair of permutations f", its constituents f_0 and f_1, permutations of the same domain, being understood. Also, we will sometimes refer to the domain of a pair f as D_f. The proof of the following lemma is elementary.

LEMMA 2.10 *Let f be a pair of claw-free permutations. It is infeasible to find $a, a' \in \{0,1\}^*$, neither one a prefix of the other, and $b, b' \in D_f$ such that*

$$f_{[a]}(b) = f_{[a']}(b').$$

PROOF. By contradiction. Suppose we could find the values in question. Put $a = a_1|\cdots|a_t$ and $a' = a'_1|\cdots|a'_{t'}$. Let j be the smallest index such that $a_j \neq a'_j$. Such j exists by our assumption on a and a'. Say $a_j = 0$ and $a'_j = 1$. If $j = t$, define $\alpha = b$ and $\beta = b'$. Else, denote $a_{j+1}|\cdots|a_t$ by γ and $a'_{j+1}|\cdots|a'_t$ by δ. Put $\alpha = f_{[\gamma]}(b)$ and $\beta = f_{[\delta]}(b')$. Then, by the fact that f_0 and f_1 permute the same set, we must have $f_0(\alpha) = f_1(\beta)$. This contradicts the assumption (f_0, f_1) is a claw-free pair. □

Informally, a *family* of claw-free *trapdoor* permutations pairs is a collection of efficiently computable claw-free pairs f permuting efficiently samplable (with uniform distribution) sets D_f, for which we are given an efficient generator that outputs a description of a pair f from this family, together with a description of the domain D_f *and* trapdoor information s_f. It is assumed that membership of D_f can be efficiently tested given just D_f's description. Given s_f, the inverses f_0^{-1} and f_1^{-1} can now also be efficiently computed. Thus, any computationally bounded adversary who sees only f and D_f, as output by the generator, cannot find a claw for f, while the party that generated f, having access to s_f, can efficiently compute claws and invert f_0 and f_1. Also notice that this party can also efficiently compute $(f_{[a]})^{-1}(b)$ for any $a \in \{0,1\}^*$ and any $b \in D_f$. For a precise definition, see [70].

PROPOSITION 2.11 *[70] If factoring integers is intractable, then a family of claw-free trapdoor permutations exists.*

The *Legendre-symbol* of a (non-zero) residue x modulo a prime r, denoted $L_r(x)$, is equal to 1, if x is a square modulo r (a "quadratic residue") and -1 otherwise (x is then called a "non-quadratic residue"). For each non-zero $x, y \in \mathbb{Z}_r$, we have $L_r(xy) = L_r(x) \cdot L_r(y)$. For our composite $n = pq$, the *Jacobi-symbol* of a (non-zero) residu x modulo n, denoted $J_n(x)$, is defined as $L_p(x) \cdot L_q(x)$.

The family of functions from the proposition above is defined as follows. Let n be an integer that is the product of two primes p and q, such that $p \equiv 3 \bmod 8$ and $q \equiv 7 \bmod 8$. Then, -1 is a non-quadratic residue modulo p and q, $J_n(-1) = 1$ and $J_n(2) = -1$ (see for instance [78]), In the following, we will refer to these integers n as Blum-integers (see also [70]). For any Blum integer n define

$$D_n = \{0 < x < \frac{n}{2} \mid J_n(x) = 1\}.$$

Note that by definition of the Jacobi-symbol, we must have $\gcd(x, n) = 1$, for all $x \in D_n$. Moreover, membership of D_f can be efficiently tested since $J_n(x)$ can be computed efficiently given just n and x. From n, a pair of claw-free permutations f_0 and f_1 of D_n is constructed as follows.

$$f_0 : D_n \longrightarrow D_n$$

$$x \mapsto \begin{cases} x^2 \bmod n : 0 < x^2 \bmod n < \frac{n}{2} \\ -x^2 \bmod n : \frac{n}{2} < x^2 \bmod n < n, \end{cases}$$

$$f_1 : D_n \longrightarrow D_n$$

$$x \mapsto \begin{cases} 4x^2 \bmod n : 0 < 4x^2 \bmod n < \frac{n}{2} \\ -4x^2 \bmod n : \frac{n}{2} < 4x^2 \bmod n < n. \end{cases}$$

Given the factorization of n, f_0 and f_1 can be efficiently inverted: for instance, a square-root of a residue a modulo a prime r with $r \equiv 3 \bmod 4$, is computed as $a^{(r+1)/4} \bmod r$. Thus, given a square $a \in \mathbb{Z}_n^*$, we can efficiently compute a root (if we know p and q) by applying the observation above in conjunction with the Chinese Remainder Theorem (see for instance [78]). Furthermore, in [70] it is shown that given $x, y \in D_n$ such that $f_0(x) = f_1(y)$ (a "claw"), the factorization of n can be efficiently computed. Basically, the argument shows that from a given claw, one can efficiently compute two different (even when changing the signs) square-roots of the same square mod$n$. This immediately leads to the factorization of n.

We now show that claw-free trapdoor permutations give rise to signature protocols in a natural way.

DEFINITION 2.5 *Let $\mathcal{F}$ be a family of claw-free pairs of trapdoor permutations with generator G and let $f \in \mathcal{F}$ denote such a pair (f_0, f_1) and let f^{-1} denote the trapdoor that allows for efficient inversion of f_0 and f_1. Put $x = (D_f, f)$ and $w = f^{-1}$. Then R is the binary relation consisting of all such pairs (x, w).*

PROPOSITION 2.12 *Let $\mathcal{F}$ is a family of claw-free pairs of trapdoor permutations with generator $G_{\mathcal{F}}$. Then there exists a signature protocol (A, B) for $R_{\mathcal{F}}$ and $G_{\mathcal{F}}$.*

PROOF. Let a security parameter k be given, and let $t \equiv t_B$ be of polynomial size in k. Consider the following protocol.

Key-generation: A selects (x, w) by running $G(1^k)$. Let D_f denote the domain of the pair f. Common input to A and B: $x = (D_f, f)$. Private input to A: $w = f^{-1}$.

Move 1: A selects $a \leftarrow D_f$ and sends a to B.

Move 2: B selects $c \leftarrow \{0,1\}^t$ and sends c to A.

Move 3: A computes $z \leftarrow f_{[c]}^{-1}(a)$ and sends z to B, who accepts if and only if $f_{[c]}(z) = a$.

By proper choice of t, we have $t \equiv t_B > t_A$. As for collision intractability, note that given two accepting conversations (x, a, c, z) and (x, a, c', z') with $c \neq c'$, we immediately have a claw for the pair f by applying Lemma 2.10. Finally, a special honest verifier zero-knowledge simulator runs as follows. Select $c \leftarrow \{0,1\}^t$, $z \leftarrow D_f$ and compute $a \leftarrow f_{[c]}(z)$. Then x, a, c, z is an accepting conversation with the right distribution. □

2.5.4 Special Claw-Free Pairs of Trapdoor Permutations

We now modify the definitions from [70] and define a family of *special* claw-free pairs of trapdoor permutations. In some cases, one would like that any polynomial collection of claw-free trapdoor permutation pairs with security parameter k works on essentially the same domain. Since this property is not provided by the general definition in [70], nor by their implementation based on the difficulty of factoring integers, we modify this definition as follows.

DEFINITION 2.6 *Let k be the security parameter. A family of claw-free trapdoor permutations pairs $\mathcal{F} = \{\mathcal{F}_k\}$ is called* special *if, additional to f_0, f_1, f_0^{-1}, f_1^{-1} and the description of D_f, its generator G outputs, on input 1^k, an efficient embedding*[4] *$\kappa_f : \{0,1\}^{\overline{k}} \longrightarrow D_f$, where $\overline{k}$ only depends on k. Furthermore, $2^{\overline{k}}/\#D_f > 1/k^c$ for some constant c and sufficiently large k.*

[4]Here, an efficient embedding means the following. κ_f, which is injective, efficiently maps any $\overline{k}$ bitstring to an element of D_f. Furthermore, one can efficiently decide whether or not any given element of D_f is in the image of κ_f. Also, given an element y in the image of κ_f, one can efficiently compute x such that $\kappa_f(x) = y$. Finally, for our purposes, it is sufficient if κ_f is defined on all, except for maybe a negligible fraction, of $\{0,1\}^{\overline{k}}$.

Note that this definition[5] guarantees that for k sufficiently large, the probability that a random element of D_f is in the image of κ_f, is non-negligible for each permutation pair f as output by $G(1^k)$.

Let a Blum integer n be given as before. We will now demonstrate how we can construct a pair of claw-free permutations f_0 and f_1 of $\mathbb{Z}_n^*$, from a pair of claw-free permutations (F_0, F_1) of D_n. The basic idea is to partition $\mathbb{Z}_n^*$ into four pairwise disjoint subsets $D_{i,j}$ of the same size, one of which is equal to D_n, and define 4 bijections $\tau_{i,j}$, $i, j \in \{-1, 1\}$, between these sets and D_n. Next, the indicator function σ maps an arbitrary element $x \in \mathbb{Z}_n^*$ to the pair (i, j) with the property that $x \in D_{i,j}$. For $b = 0, 1$, f_b is defined as follows. Given $x \in \mathbb{Z}_n^*$, map x into D_n with the relevant bijection (depending on which of the subsets x is in) apply F_b, and map the result back into the subset that x is in.

More technically, this process works as follows. For $i, j \in \{-1, 1\}$, the sets $D_{i,j}$ are defined as

$$D_{i,j} = \{0 < x < n \mid \operatorname{sign}_n(x) = i \text{ and } J_n(x) = j\},$$

where $\operatorname{sign}_n(x)$ is equal to 1 if $0 < x < n/2$ and equal to -1 if $n/2 < x < n$. Note that $D_{1,1} = D_n$. Then, the functions $\tau_{i,j}$ are as follows. The function $\tau_{1,1}$ is the identity function, and $\tau_{-1,1}$ maps $x \in D_{-1,1}$ to $-x \bmod n$. Finally, if $j = -1$, then $\tau_{i,-1}$ first flips the Jacobi-symbol of $x \in D_{i,-1}$ by computing $2x \bmod n$. This is the output of $\tau_{i,-1}$ unless $n/2 < 2x \bmod n < n$ in which case the output is $-2x \bmod n$. From the properties of Blum-integers as given above, it is easily derived that the sets $D_{i,j}$ partition $\mathbb{Z}_n^*$ into 4 disjoint sets of the same size and that the functions $\tau_{i,j}$ are bijective. It also follows that the indicator function σ is well-defined for all $x \in \mathbb{Z}_n^*$. Define the new pair of claw-free trapdoor permutations as follows.

$$f_0 : \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_n^*$$

$$x \mapsto \tau_{\sigma(x)}^{-1} \circ F_0 \circ \tau_{\sigma(x)}(x),$$

$$f_1 : \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_n^*$$

$$x \mapsto \tau_{\sigma(x)}^{-1} \circ F_1 \circ \tau_{\sigma(x)}(x).$$

The functions sign_n, σ and $\tau_{i,j}$ (and $\tau_{i,j}^{-1}$) can be efficiently computed when given the modulus n. Thus f_0 and f_1 can be efficiently computed when given n. Furthermore, f_0 and f_1 can efficiently be inverted when given the factorization of n.

[5]We will also assume that the pair f, as output by $G(1^k)$, can be represented by a k-bit string and that from such a description, the corresponding embedding κ_f and the domain D_f can efficiently be derived. Finally, we assume that $\overline{k} \leq k$.

LEMMA 2.11 *f_0 and f_1 constitute a pair of claw-free permutations of $\mathbb{Z}_n^*$, if F_0 and F_1 constitute a pair of claw-free permutations of D_n.*

PROOF. Observe that f_0 and f_1, when restricted to $D_{i,j}$, are in fact permutations of $D_{i,j}$, $i,j \in \{0,1\}$. Now suppose that we can compute $x, y \in \mathbb{Z}_n^*$ such that

$$f_0(x) = f_1(y),$$

given only n, but not its factorization. It follows that

$$\tau_{\sigma(x)}^{-1} \circ F_0 \circ \tau_{\sigma(x)}(x) = \tau_{\sigma(y)}^{-1} \circ F_1 \circ \tau_{\sigma(y)}(y)$$

But then we must have, for some $i, j \in \{-1, 1\}$,

$$\sigma(x) = \sigma(y) = (i, j),$$

by the first remark and the fact that all $D_{i,j}$ are disjoint. Therefore, we have

$$F_0(\tau_{i,j}(x)) = F_1(\tau_{i,j}(y)),$$

with $\tau_{i,j}(x), \tau_{i,j}(y) \in D_n$. Thus we have created a claw for F_0 and F_1. □

We can now define our special family $\mathcal{F}$ of claw-free pairs of trapdoor permutations based on the difficulty of factoring integers, by defining a *special* generator G.

Let $N(k)$ denote the set of all Blum-integers n such that $2^{k-1} < n < 2^k$. Let $n \in N(k)$, and let $f = (f_0, f_1)$ be the pair of claw-free permutations of $\mathbb{Z}_n^*$ as constructed above. Define $E(k)$ to be the set of integers x such that $0 \leq x < 2^{k-1}$, and put $\overline{k} = k - 1$. Suppose that the prime factors of n have size approximately equal to $\sqrt{n}$. Then:

1. All but a negligible fraction of E_k is contained in $D_f = \mathbb{Z}_n^*$.
2. $\#E_k / \#D_f \geq \frac{1}{2} - \epsilon$, where ϵ is negligible in k.

DEFINITION 2.7 *(Special Generator G)*
On input of 1^k, G outputs a random $n \in N(k)$ together with its prime factors p and q, subject to the condition that these prime factors have size approximately $\sqrt{n}$. The corresponding claw-free permutation pair $f = (f_0, f_1)$ is defined as above. The trapdoor information s_f consist of the factorization p and q of n. The embedding $\kappa_f : \{0,1\}^{k-1} \longrightarrow D_f = \mathbb{Z}_n^*$ is defined by identifying $E(k)$ with $\{0,1\}^{k-1}$ and mapping $x \in E(k)$ to $x \bmod n$. The family of special claw-free pairs of trapdoor permutations, generated by G, is denoted $\mathcal{F}_{fact}$.

REMARK 2.2 *The embedding κ_f is defined for all, except a negligible fraction, of $\{0,1\}^{k-1}$.*

PROPOSITION 2.13 *If factoring integers is intractable, then there exists a* special *family of claw-free pairs of trapdoor permutations.*

3
Partial Proofs and Applications

3.1 Introduction

Most competitive people, like businessmen, artists and scientists, have the desire to convince the world that they are experts in their field. But wouldn't it be typical of cryptographers to show that they are experts in something without revealing much about what it is? Let's illustrate this with a small example. Suppose one were to convince an opponent that one has expert knowledge about Komodo dragons or about the life and times of Pierre de Fermat, without revealing which.

One could propose the following protocol. For each of the two subjects, the opponent supplies a numbered list of 1000 expert questions. Before one is going to be challenged about one's knowledge, one is allowed to look up all answers to the questions in both lists. To this end, the opponent supplies two lists with the answers. In exactly 5 minutes one must hand back the lists with answers to the opponent. Now, suppose one knows everything about the Komodo dragons. One then chooses a random number r in the range from 1 to 1000, and one looks up the answer to question number r about Fermat, and memorizes it. After 5 minutes are over and the lists with answers have been returned, one is challenged with a random number c that is selected by the opponent. One then computes $r' = c - r \pmod{1000}$ and one gives the opponent the answers to questions r and r', where question r' must be taken from the list with questions about the Komodo dragons: a very obscure question indeed, it turns out. Nevertheless, one is happy and eager to answer it.

This protocol will convince the opponent that one is an expert in one of the two subjects, but giving no information about which. When in turn, one challenges the opponent to guess which of the two subjects one really masters, the opponent will answer: "I don't know about the dragons and Fermat, but you play such weird games that you must be a cryptographer."

To put it more abstractly and seriously, the example just given is concerned with convincing an opponent of the veracity of a subset of a number of given statements without revealing which subset is in the play. In our little example, there are two statements and a subset is accepted if it has at least one member. This idea can for example be extended to saying that a proof about n statements is accepted if at least d of them are true. If more than d statements are true, the proof is still accepted, and it is in this sense that the condition on the subsets of statements that are accepted is *monotone*. In fact, any such condition on the set $\{1, \ldots, n\}$ can be expressed by a Boolean formula that uses AND- and OR-operators only and vice versa.

In this chapter we develop cryptographic techniques that are useful in the scenario sketched above. But more importantly, we demonstrate that the resulting techniques are useful as efficient and secure building blocks in a variety of cryptographic areas. Applications to *identification schemes* that are secure even in the presence of a *man-in-the-middle* [11], and more generally to *witness hiding protocols* are presented. Apart from providing a general framework that tells us how to efficiently and securely design such protocols under a wide variety of cryptographic assumptions, some of the implementations are very competitive to existing proposals. As an application we propose a *secret ballot election scheme* that compares favorably to previous schemes in that it minimizes work and communication required.

More theoretically, but within reach of what is realizable in practice today, we show that our techniques support the most efficient general zero knowledge proofs that are known to date. A general zero knowledge proof is one which works for any (NP) language L, provided that a suitable description of L is given, such as one based on circuits that verify witnesses. It is now feasible, for instance, to prove that one possesses a DES-key mapping a given plain text pair to a given ciphertext. While these methods rely on suitable complexity assumptions, we also study which classes of NP-languages admit zero knowledge under monotone operations to be conducted using no complexity assumptions.

In Section 3.2 we first study Σ-protocols between a prover and a verifier that have n strings as common input instead of just one common input x. Subject to a given monotone condition as discussed above, the prover demonstrates knowledge about a subset of these n strings, without revealing which subset it is. We take honest verifier zero knowledge Σ-protocols satisfying special soundness (or collision intractability) and show that for any monotone condition there is a corresponding honest verifier zero knowledge Σ-protocol satisfying special soundness (or collision intractability). Technically speaking, the idea that a prover does not reveal the subset regarding which the proof is conducted, relates to the notion of witness indistinguishability. As in our context witness indistinguishability is implied by honest verifier zero knowledge, our results allow a prover to prove knowledge about a subset of 1000 RSA-moduli, where any subset defined by a monotone condition is accepted but without revealing which subset is used.

If the instances of our scheme are properly generated, this leads to *witness hiding* protocols. As noted before, these are a fundamental tool in the design of secure and efficient cryptographic protocols. Roughly speaking our work differs from other work in this area [102, 103] in that we consider a different model of protocols facilitating the kind of services we are interested in. Moreover, our methods may lead to more efficient implementations since we have found an intimate relation between the efficiency of the protocols under consideration and available *efficient secret sharing schemes* for the corresponding monotone conditions.

In Section 3.3 we apply the techniques of Section 3.2 to the problem of designing zero knowledge protocols for proving general NP-statements [71, 24]. More precisely, we present zero knowledge proof for *circuit satisfiability*, with *low communication complexity*. The communication in our protocol corresponds to a number of bit commitments that is linear in the number of gates of the circuit that verifies an NP-witness. We therefore improve on the results of [22, 83, 84]. Our construction is efficient enough to be practical in many applications.

In Section 3.4 we study NP-languages that admit an honest verifier Σ-interactive proof and show that, under composition with monotone conditions as in Section 3.2, we can obtain Σ-zero knowledge proofs for the composite languages. Similar results are obtained even if we allow the atomic languages to be complementary NP-languages to the ones we started with. A key technique is *interactive hashing* that essentially allows us to view the atomic protocols in terms of the results of the previous section. This extends the results of [103].

In Section 3.5 we present a fast and secure secret ballot election scheme [27] that can be implemented under a discrete logarithm assumption. Our approach follows the paradigm of [17], but achieves a significant reduction of the amount of communication required by showing that some of the efficient witness indistinguishable protocols from Section 3.2 are sufficient to implement some of the key ingredients of secret ballot election schemes. Previously, these were implemented using more costly zero knowledge protocols.

In Section 3.6 we show that under the general assumption that honest verifier zero knowledge and collision intractable Σ-protocols exist, we can implement secure identification schemes that even withstand so-called man-in-the-middle attacks in several important scenarios. When instantiated for protocols based on the factoring problem or the discrete logarithm problem, our construction leads to practical identification schemes that compete with known ones that do not provide security against the man-in-the-middle adversary.

The material in this chapter is based on [47, 51, 50, 48, 46].

3.2 Proofs of Partial Knowledge

3.2.1 Introduction

This section provides the theoretical basis for most of the results to follow in this chapter.

Suppose we are given n RSA-moduli $m_1, \ldots, m_n$, and suppose that for, say, half of them a prover knows a non-trivial factor. Given a Σ-protocol that constitutes a proof of knowledge that the prover knows a non-trivial factor of some given composite integer, we construct in this section a protocol that allows the prover to show that he knows non-trivial factors for at least $n/2$ of the n given RSA-moduli. But most importantly, the verifier is not able to decide for which half of the n moduli the prover knows non-trivial factors.

Our results are much more general than this example. First, we can deal with any threshold condition "d-out-of-n". But in fact we can construct a proof of knowledge that allows a prover to show, for instance, that he knows non-trivial factors for at least one third of the odd-numbered m_j's and for at least one fourth of the even-numbered m_j's. In its most general form, our constructions can handle any condition that can be formulated by a *monotone Boolean formula*, that is, a formula consisting of AND-, and OR-operators only.

As an example of our main result in this section, we now describe a special case. Let f be a monotone Boolean formula on n input bits, and let (A, B) be a Σ-protocol for relation R. Suppose that (A, B) satisfies special soundness and honest verifier zero knowledge. Let arbitrary bit strings $x_1, \ldots, x_n$ be given (with the same length), and let I be an arbitrary subset of $\{1, \ldots, n\}$. Next, suppose we are given a collection of bit strings $w = \{w_j\}_{j \in I}$ such that for all $j \in I$, $|w_i| \leq p(|x_i|)$ for some fixed given polynomial p. Let a_I denote the characteristic vector of I (that is, the i-th bit of a_I is equal to 1 if $j \in I$ and equal to 0 otherwise, $j = 1, \ldots, n$). We say that w is an (R, f)-witness for $(x_1, \ldots, x_n)$ if and only if $f(I) = 1$ and $(x_j, w_j) \in R$ for all $j \in I$. Our result implies a Σ-protocol (A', B'), satisfying special soundness and honest verifier zero knowledge, that the prover knows an (R,f)-witness for $(x_1,\dots,x_n)$. Since, by Lemma 2.5, honest verifier zero knowledge implies witness indistinguishability, no verifier can distinguish between two provers who are given distinct (R,f)-witnesses for $(x_1,\dots,x_n)$.

Given an invulnerable generator G for the protocol (A,B) (and some condition on f), the resulting protocol (A',B') is shown to be witness hiding. This means that no polynomially bounded malicious verifier B' can extract an (R,f)-witness for $(x_1,\dots,x_n)$. Under some extra condition on f, it is shown that the malicious verifier cannot even extract a single witness w_j (with respect to the relation R) for any of the x_j's ($j = 1\dots n$). And as such the results in this section are interesting in their own right, as we give a general and efficient method of design for such protocols. By the results in Section 2.5, these protocols are now efficiently available under a wide variety of complexity assumptions.

Our technical approach makes extensive use of the *simulation properties* of our basic protocols (see Section 2.4.3), and *secret sharing schemes for monotone functions*, to be explained below.

3.2.2 Monotone functions

A function $f:\{0,1\}^n \to \{0,1\}$, $f \not\equiv 0,1$, is called *monotone* if the following holds. If $f(a_1\dots a_n) = 1$ and if $b_1\dots b_n \in \{0,1\}^n$ is such that, for $i = 1\dots n$, $b_i = 1$ if $a_i = 1$, then $f(b_1\dots b_n) = 1$. By $\mathcal{F}_n$ we denote a family of monotone functions where each of its members takes n bits as input. $\mathcal{F} = \cup_{n>0}\mathcal{F}_n$, denotes the union of such collections: a family of monotone functions. In the notation $f_n \in \mathcal{F}$, the subscript n to f_n serves as a reminder that $f_n \in \mathcal{F}_n$. Although this is not reflected in our terminology, we will only consider families $\mathcal{F}$ of monotone functions f_n that can be computed in time polynomial in n. Furthermore, we will assume that membership of $\mathcal{F}$ can be efficiently decided. Finally, by saying $f \in \mathcal{F}$ we mean the *function* itself, or a bitstring *encoding* this function efficiently. The distinction should be clear from the context.

Let $I \subset \{1,\dots,n\}$. We define $a_I \in \{0,1\}^n$ by setting the i-th position in a_I to 1 if $i \in I$ and to 0 otherwise. Then $f(I)$ denotes $f(a_I)$. A set I is called *minimal* if and only if $f(I) = 1$ and there is no proper subset J of I with $f(J) = 1$.

The *dual* $f^*:\{0,1\}^n \to \{0,1\}$ is defined as follows. For each $a \in \{0,1\}^n$, $f^*(a) = 1$ if and only if $f_n(a \oplus \mathbf{1}) = 0$, where $\mathbf{1}$ denotes the all-one string and '$\oplus$' denotes bit-wise xor. It is easily verified that f^* is monotone as well. If $\mathcal{F}$ is a family of monotone functions, then its dual $\mathcal{F}^*$ is obtained by replacing all monotone functions in $\mathcal{F}$ by their duals.

By a *monotone Boolean formula* we mean a function given as a Boolean formula consisting of AND-operators and OR-operators only. We assume these operators to have two input bits and one output bit. A family $\mathcal{F}$ of monotone Boolean formulas is polynomially sized if the number of operators is polynomially bounded in n. If f is a monotone boolean formula, then its dual f^* is obtained by swapping the AND-operators and OR-operators. It follows that the dual of a polynomially sized family of monotone Boolean formulas is polynomially sized as well.

Let n, d be positive integers with $d \leq n$. A (n,d)-*threshold function* $f_{n,d} : \{0,1\}^n \to \{0,1\}$ returns 1 if the Hamming-weight $w_H(a)$ of $a \in \{0,1\}^n$ (that is, the number of positions in the string a that are equal to 1) is at least equal to d and 0 otherwise. Such functions are clearly monotone. It holds that $f^*_{n,d} = f_{n,n-d+1}$. Indeed $f^*_{n,d}(a) = 1 \Leftrightarrow f_{n,d}(a \oplus \mathbf{1}) = 0 \Leftrightarrow w_H(a \oplus \mathbf{1}) \leq d-1 \Leftrightarrow w_H(a) \geq n-d+1$.

3.2.3 Secret Sharing

Consider a monotone function family $\mathcal{F}$. We now define an *efficient secret sharing scheme with completion* $\mathcal{S}$ for $\mathcal{F}$. Let $f_n \in \mathcal{F}$. First, a probabilistic algorithm `dist` (distribution) is given that takes a (secret) bit b and integer n as input and outputs n shares $s_1, ..., s_n$, running in time polynomial in n, whence the length of the i'th share is upper bounded by a polynomial $l_i(n)$, which is given. Furthermore, the following conditions must be satisfied:

1. *Perfectness*: for any I such that $f_n(I) = 1$, the bit b can be reconstructed in time polynomial in n using the polynomial time algorithm `rec` on input $\{s_i | \ i \in I\}$, whereas if $f_n(I) = 0$, the distribution of $\{s_i | \ i \in I\}$ generated from $b = 1$ is the same as when $b = 0$; this means that if $f_n(I) = 0$, then $\{s_i | \ i \in I\}$ gives no information about b.

2. *Consistency Verification*: given strings $s_1, \ldots, s_n$, with $s_i \in \{0,1\}^{l_i(n)}$ for $i = 1 \ldots n$, and a bit b, it is possible, using the algorithm `cons` to test in time polynomial in n that for each collection $\{s_i | \ i \in I\}$ with $f_n(I) = 1$, the reconstruction procedure outputs the bit b. If so, we say that "the full set of shares $s_1, \ldots, s_n$ is consistent with b".

3. *Completion*: if J is a set for which $f_n(J) = 0$, then the distribution of the subset of shares corresponding to J is independent of the secret bit, as follows from the perfectness condition. Call this distribution D_J. Given a subset of shares distributed according to D_J and a bit b, it is always possible, using the probabilistic algorithm `comp` to complete this subset to a full set of shares consistent with b in polynomial time in n, and with the same distribution as the output distribution of $\mathcal{S}$.

The algorithms above depend on which $f_n \in \mathcal{F}$ is taken, of course. Implicitly, we assume that some suitable and efficient reference to these functions is given, and that the algorithms take such a reference as input. A scheme satisfying the first condition is called a *perfect secret sharing scheme* (see for instance [114]). The significance of the novel notions of completion and consistency verification will become clear from the "partial proofs" presented in Theorem 3.3.

More precisely, the algorithms that constitute an efficient secret sharing scheme $\mathcal{S}$ with completion are as follows. Let $f_n \in \mathcal{F}$. Neglecting dependence on f_n as before, these algorithms are defined as follows.

dist takes as input a secret bit b and a random bitstring $\rho \in \{0,1\}^u$ and outputs a collection of shares $(s_1, \ldots, s_n)$ corresponding to a member of the set $\{0,1\}^{l_1} \times \cdots \times \{0,1\}^{l_n}$. Here u and the l_i are of polynomial length in n. For any set $J \subset \{1, \ldots, n\}$, $\texttt{dist}(b,\rho)|_J$ denotes the shares corresponding to the members of J. If J is the empty set, then this also denotes the empty set.

rec takes as input $I \subset \{1, \ldots, n\}$ with $f_n(I) = 1$ and shares $\texttt{dist}(b,\rho)|_I$, and outputs the secret bit b. More formally, we require that $\forall b \in \{0,1\}$, $\forall \rho \in \{0,1\}^u, \forall I \subset \{1, \ldots, n\}$ with $f_n(I) = 1$, we have $\texttt{rec}(I, \texttt{dist}(b,\rho)|_I) = b$.

cons takes as input $s = (s_1, \ldots, s_n)$, with $s_i \in \{0,1\}^{l_i(n)}$ for $i = 1 \ldots n$ and a bit b, and outputs[1] *accept* if s is consistent with b, and *reject* otherwise.

comp takes as input any two secret bits b and b', and $\rho' \in \{0,1\}^u$. It outputs $\rho \in \{0,1\}^u$, with the following property. $\forall b, b' \in \{0,1\}$, $\forall \rho' \in \{0,1\}^u$, $\forall J \subset \{1, \ldots, n\}$ with $f_n(J) = 0$, $\texttt{comp}(b, b', \rho') = \rho$ such that $\texttt{dist}(b,\rho)|_J = \texttt{dist}(b',\rho')|_J$.

Note that our definition of an efficient secret sharing scheme with completion allows the distribution of n shares consistent with one secret bit b. However, all definitions given above can easily be adapted to the situation where the secret sharing scheme takes as input a secret bit string of variable length instead of just one bit b. The following lemma is straightforward.

LEMMA 3.1 *Let a family $\mathcal{F}$ of monotone Boolean functions be given, and a collection $\mathcal{S}$ of algorithms* dist, rec, comp *and* cons *defined as above, running in time polynomial in n. If* comp *is bijective when the first two arguments are fixed, then $\mathcal{S}$ is an efficient secret sharing scheme with completion for $\mathcal{F}$.*

Let $\mathcal{F}$ be a polynomially sized family of monotone Boolean formulas. We now describe an efficient secret sharing scheme $\mathcal{S}$ with completion for $\mathcal{F}$ which is based on the perfect secret sharing scheme due to Benaloh and Leichter [14]. Let $f \in \mathcal{F}$ take n input bits. For clarity, we now also give the

[1]Observe that if the s_i's were generated by dist on input b (and some string ρ), then cons will output *accept*.

monotone function according to which the bit b is distributed as input to the algorithms `dist`, `rec`, `comp` and `cons`.

The construction of the algorithm `dist` below uses induction on the number of operators in the formula f. AND-, and OR-operators are denoted "$\wedge$" and "$\vee$" respectively. First note that if f has at least one operator, then f can either be written as

1. $f = f_1 \wedge f_2$, or
2. $f = f_1 \vee f_2$,

where f_1 and f_2 are monotone Boolean formulas as well. We start by treating the case where f consists of a single operator. In the first case, for any $b \in \{0,1\}$, we put $\texttt{dist}(b, \rho, f) = (b, b \oplus \rho)$, where ρ is a random bit. In the second case, $\texttt{dist}(b, \rho, f) = (b, b)$ for any $b \in \{0,1\}$. Here ρ is the empty string. If f is the identity function on one bit, we define $\texttt{dist}(b, \rho, f) = b$, where ρ is the empty string.

As an induction hypothesis, let $d > 1$ and suppose that `dist` handles monotone Boolean formulas consisting of at most d operators. Now assume that f has exactly $d + 1$ operators. The two cases from above are treated as follows. Write u, u_1, u_2 for the number of $\wedge$-operators in f, f_1 and f_2, respectively. Let $r \in \{0,1\}$, and let $\rho_j \in \{0,1\}^{u_j}$, $j = 1, 2$. If $f = f_1 \wedge f_2$, write the random string $\rho \in \{0,1\}^u$ as (r, ρ_1, ρ_2). If $f = f_1 \vee f_2$, then ρ is written as (ρ_1, ρ_2). For any $b \in \{0,1\}$, we now define

1. $\texttt{dist}(b, \rho, f) = \texttt{dist}(b, \rho_1, f_1) \cup \texttt{dist}(b \oplus r, \rho_2, f_2)$, if $f = f_1 \wedge f_2$.
2. $\texttt{dist}(b, \rho, f) = \texttt{dist}(b, \rho_1, f_1) \cup \texttt{dist}(b, \rho_2, f_2)$, if $f = f_1 \vee f_2$.

In this construction, the $\cup$-operation must be interpreted as follows. Assume that f takes n input bits $a_1, \ldots, a_n$. Consider an arbitrary input bit a_i of the formula f, for some $1 \leq i \leq n$. Assume that f reads input bit a_i $l^{(i)}$ times while f_j reads a_i $l_j^{(i)}$ times, $i = 1, \ldots, n$, $j = 1, 2$. Of course we have $l_1^{(i)} + l_2^{(i)} = l^{(i)}$. The construction can easily be seen to give one bit "through" each of these input wires. Breaking up f into f_1 and f_2 as above, $\texttt{dist}(\cdot, f_j)$ gives $l_j^{(i)}$ bits "to a_i", $j = 1, 2$. If we denote the respective shares by $s_1^{(i)}$ and $s_2^{(i)}$, the share "for a_i" as a result of $\texttt{dist}(\cdot, f)$ is taken to be $(s_1^{(i)}, s_2^{(i)})$ for each $1 \leq i \leq n$. Thus, a full set of shares can be seen as corresponding to an element of the set $\{0,1\}^{l^{(1)}} \times \cdots \times \{0,1\}^{l^{(n)}}$. Also, the number of additions is easily to be seen as u, where u is the number of $\wedge$-operators in f. This concludes our description of `dist`.

Write $f = f_1 \wedge f_2$ or $f = f_1 \vee f_2$. In the following, for any set $J \subset \{1, \ldots, n\}$, J_1, respectively J_2, denotes the collection of $i \in J$ such that f_1, respectively f_2, reads input variable a_i. We are now prepared to prove the following theorem.

THEOREM 3.1 *Let $\mathcal{F}$ be a polynomially sized family of monotone boolean formulas. Then there exists an efficient secret sharing scheme with completion $\mathcal{S}$ for $\mathcal{F}$.*

PROOF. All arguments to follow use induction on the number of operators in f. We will only treat the case $f = f_1 \wedge f_2$, as the other case follows similarly. The statements to follow are trivially true for formulas consisting of just one operator. Induction hypotheses are made implicitly.

Let $I \subset \{1, \ldots, n\}$ with $f(I) = 1$. Then we have $f_j(I_j) = 1$, $j = 1, 2$. Note that

$$\texttt{rec}(\texttt{dist}(b, \rho, f)|I) = \texttt{rec}(\texttt{dist}(r, \rho_1, f_1)|I_1) \oplus \texttt{rec}(\texttt{dist}(b \oplus r, \rho_2, f_2)|I_2).$$

The latter two inputs to rec can be constructed from the first one, which is given. This is done by inspection of the formula f, and by separating the parts output by $\texttt{dist}(\cdot, \cdot, f_j)$, $j = 1, 2$. Now we apply the induction hypothesis. We have that rec uses u additions.

$\texttt{cons}(s, b, f)$ works by recursively computing the coin tosses ρ. If any inconsistencies occur, output *reject*. If this process results in some value $\rho \in \{0, 1\}^u$, check if $\texttt{dist}(b, \rho, f) = s$. If so, output *accept*. Else output *reject*. cons uses at most $2u$ additions. Here, cons satisfies a stronger property than required by the definition of cons: it follows easily[2] that s is consistent with the bit b if and only if s is a possible output of dist on input b and some suitable random string ρ.

The construction and properties of the comp-algorithm are based on the following claim.

CLAIM 3.1 *Let $J \subset \{1, \ldots, n\}$ such that $f(J) = 0$ and $J \neq \emptyset$. There exists an efficiently computable function $T : \{0, 1\}^{u+2} \longrightarrow \{0, 1\}^u$, depending on f and J, such that $\forall b, b' \in \{0, 1\}$, $\forall \rho \in \{0, 1\}^u$,*

$$\texttt{dist}(b', \rho, f)|_J = \texttt{dist}(b, T(b, b', \rho), f)|_J,$$

where T is bijective if b and b' are fixed.

We now prove the claim. We have $\texttt{dist}(b', \rho, f)|_J = \texttt{dist}(r, \rho_1, f_1)|_{J_1} \cup \texttt{dist}(b' \oplus r, \rho_2, f_2)|_{J_2}$ and $0 = f(J) = f_1(J_1) \wedge f_2(J_2)$. Without loss of generality, we may assume that $f_1(J_1) = 0$. If $J_1 = \emptyset$, we must have $J_2 \neq \emptyset$. Then, $\texttt{dist}(b', \rho, f)|_J = \texttt{dist}(b' \oplus r, \rho_2, f_2)|_{J_2}$. The claim follows by putting $T(b, b', \rho) = (b \oplus b' \oplus r, \rho_1, \rho_2)$.

Now suppose that $J_1 \neq \emptyset$. In this case, use of the induction hypothesis is required. We may assume $\texttt{dist}(r, \rho_1, f_1)|_{J_1} = \texttt{dist}(b \oplus b' \oplus r, T_1(r, b \oplus b' \oplus$

[2]That is, from the observation regarding the linear algebra approach, given at the end of the proof.

$r, \rho_1), f_1)|_{J_1}$, where T_1 is the function guaranteed by our induction hypothesis. But now, $\texttt{dist}(b', \rho, f)|_J = \texttt{dist}(b, T(b, b', \rho), f)|_J$, where $T(b, b', \rho) = (r, T_1(r, b \oplus b' \oplus r, \rho_1), \rho_2)$, obviously a bijection if b and b' are fixed. The algorithm comp is now easily defined in terms of dist and T: for given J, b', ρ and b, compute $T(b, b', \rho)$. Then the required full set of shares, consistent with b is equal to $\texttt{dist}(b, T(b, b', \rho), f)$. Note that T uses at most $4u$ additions. The proof is concluded by applying Lemma 3.1. All algorithms clearly run in time polynomial in n.

A practical approach to this efficient secret sharing scheme with completion is given by describing it in terms of linear algebra. Let f have u $\wedge$-operators and define l as $\sum_{i=1}^{n} l_i$. Then dist is a linear map

$$\texttt{dist} : GF(2) \times GF(2)^u \longrightarrow GF(2)^l.$$

The first argument stands for the secret bit b while the second argument captures the u coinflips used at the u $\wedge$-operators. The image represents the shares. A matrix describing this map is easily defined in terms of the formula f and the inductive description of dist. □

REMARK 3.1 *The scheme from Theorem 3.1 can be executed m times to get an efficient secret sharing scheme for distributing secret m-bit strings instead of just one secret bit b, if $m = poly(n)$.*

As pointed out in Section 3.2.2, the dual of a monotone Boolean formula is obtained by interchanging the $\wedge$- and $\vee$-operators. Thus, we immediately have the following corollary of Theorem 3.1.

COROLLARY 3.1 *Let $\mathcal{F}$ be a polynomially sized family of monotone boolean formulas. Then there exists an efficient secret sharing scheme with completion $\mathcal{S}$ for $\mathcal{F}^*$.*

Let n, d be positive integers with $d \leq n$ and let K be any finite field with $|K| > n$. Shamir's protocol [107] provides a perfect secret sharing scheme for n, d-threshold functions as follows. Select a random polynomial $f(X) \in K(X)$ of degree at most $d - 1$ with the constraint that $f(0) = s$, where $s \in K$ is the secret that is to be distributed. The i-th share, $1 \leq i \leq n$, is computed as $f(i)$. By Lagrange-interpolation, it can be shown that from any set consisting of at least d such shares, the secret s can efficiently and uniquely be computed. Moreover, no information about s can be inferred from any set of less than d shares. Note that this scheme is also *ideal* in the sense that the size of each share is equal to the size of the secret that is distributed.

About completion in this scheme. Consider a set $J \subset \{1, \ldots, n\}$ with $|J| = m < d$. It can be shown easily that the distribution of shares for J is uniform on K^m and independent from anything else. So, to complete any set of shares $\{s_i\}_{i \in J}$ to be consistent with any given secret s, we just choose a random polynomial f of degree at most $d-1$ such that $f(0) = s$ and $f(i) = s_i$ for $i \in J$. Thus we have the following theorem.

THEOREM 3.2 *For threshold functions, there exist efficient secret sharing schemes with completion.*

3.2.4 Main Results

We are now ready to put all the pieces together, and to prove the theorem promised in the introduction. Let $\mathcal{F}$ be a family of monotone functions and let R be a binary relation as before. We now define the relation $R_{\mathcal{F}}$. Let $p(\cdot)$ be a given polynomial such that for all $(x,w) \in R$, we have $|w| \leq p(|x|)$. For all n, for all $f_n \in \mathcal{F}_n$ and for all k, consider the collection of all tuples $x = (f_n, x_1, \dots, x_n)$ such that $x_i \in \{0,1\}^k$ for $i = 1 \dots n$. Let $w = \{w_j\}_{j \in I} \subset \{0,1\}^*$ be given where I is an arbitrary subset of $\{1, \dots, n\}$ and $|w_j| \leq p(|x_j|)$. Then $R_{\mathcal{F}}$ is the binary relation consisting of all such pairs (x,w) such that $f_n(I) = 1$ and $(x_j, w_j) \in R$ for $j \in I$. If $(x,w) \in R_{\mathcal{F}}$, we will say that w is an (R, f_n)-witness for $(x_1, \dots, x_n)$.

Theorem 3.3 to follow, transforms a Σ-protocol (A,B) for relation R, satisfying honest verifier zero knowledge and special soundness, into a new Σ-protocol (A', B'), satisfying honest verifier zero knowledge and special soundness *with respect to the relation* $R_{\mathcal{F}}$, instead of R. Here, $\mathcal{F}$ is a family of monotone functions such that its dual $\mathcal{F}^*$ has a given and efficient secret sharing scheme (see Section 3.2.3).

First, we give a concrete example of our construction based on Schnorr's protocol (see Section 2.3.2) and a threshold function. Then we outline the general case from Theorem 3.3.

Consider the family $\mathcal{F}$ consisting of a single monotone function f that takes n bits as input, where n is fixed. The output is 1 if and only if the Hamming-weight of the input is at least d, where $1 \leq d \leq n$. Then we can say that $R_{\mathcal{F}}$ consists of all pairs $((x_1, \dots, x_n), (w_1, \dots, w_n))$ such that the x_i's have the same length, the w_i's satisfy $|w_i| \leq p(|w_i|)$, and at least d pairs (x_i, w_i) are elements of R. In Figure 3.1 we give an example based on this setting. Here (A,B) is taken to be Schnorr's identification protocol (see Section 2.3.2). Let G_q be a group of prime order q and let $g \in G_q \setminus \{1\}$ and let $x_1, \dots, x_n \in G_q$. The common input to (A', B') is a description of the group G_q (including q), g, n, d, $x_1, \dots, x_n$. The purpose of A' is to convince B' that A' knows the discrete logarithms for at least d of the n x_i's, without revealing which. For convenience we assume that the prover A' is given the discrete logarithms $\log_g x_i$ $(i = 1 \dots d)$, while the discrete logarithms for the remaining $n - d$ x_i's are not given. Notice that A' simulates conversations (in Schnorr's protocol) for the x_i's with $d+1 \leq i \leq n$. For the other ones, A' behaves like the honest prover in Schnorr's protocol. The function f from Figure 3.1 is the interpolation polynomial from Shamir's secret sharing

$$
\begin{array}{rcl}
\textsf{Prover} & & \textsf{Verifier} \\
(x_i = g^{w_i},\ 1 \le i \le d) & & \\
 & & \\
u_1, \ldots, u_d \leftarrow \mathbb{Z}_q & & \\
c_{d+1}, \ldots, c_n, z_{d+1}, \ldots, z_n \leftarrow \mathbb{Z}_q & & \\
a_i := \begin{cases} g^{u_i} & ,1 \le i \le d \\ g^{z_i} x_i^{-c_i} & ,d{+}1 \le i \le n \end{cases} & \xrightarrow{a_1, \ldots, a_n} & \\
 & & c \leftarrow \mathbb{Z}_q \\
 & \xleftarrow{\quad c \quad} & \\
\text{Solve } \alpha_1, \ldots, \alpha_{n-d} \in \mathbb{Z}_q \text{ s.t. } f(0) = c & & \\
\text{and } f(i) = c_i \text{ for } d{+}1 \le i \le n & \xrightarrow{\alpha_1, \ldots, \alpha_{n-d}} & \\
z_i := \begin{cases} u_i + f(i) w_i & ,1 \le i \le d \\ z_i & ,d{+}1 \le i \le n \end{cases} & \xrightarrow{z_1, \ldots, z_n} & \\
 & & g^{z_i} \stackrel{?}{=} a_i x_i^{f(i)}, \\
 & & 1 \le i \le n
\end{array}
$$

FIGURE 3.1. Example (n, d) WI Proof of Knowledge

scheme (see Section 3.2.3), and is defined as $f(X) = c + \sum_{i=1}^{n-d} \alpha_i X^i$; here c is the challenge from B'. After receiving this challenge, A' computes those α_j's ($j = 1 \ldots n - d$) such that $f(i) = c_i$ for $i = d + 1 \ldots n$ (note that $f(0) = c$), where the c_i's are taken from the simulated conversations. For $i = 1 \ldots d$, the i-th challenge, corresponding to x_i, is computed as $f(i)$. In the final step, A' computes the replies z_i, $i = 1 \ldots d$, and selects the other z_i's from the simulations. All α_i's and z_i's are sent to the verifier B', who checks the conversation.

We now outline the general case. Informally speaking, (A', B') shows to the verifier B' that the prover A' (R, f_n)-knows a witness for their common string $(x_1, \ldots, x_n)$ where $f_n \in \mathcal{F}$. Nevertheless, the verifier B' gets no information as to which *particular* witness A' holds. We get similar results if we assume that (A, B) has the collision-property or satisfies collision-intractability, instead of special soundness. By Proposition 2.5, we may assume that (A, B) has a special simulator M. We briefly outline the techniques we use. Let $x = (f_n, x_1, \ldots, x_n)$ and $w = \{w_j\}_{j \in I}$ be such that $(x, w) \in R_{\mathcal{F}}$ as above. Let I' denote $\{1, \ldots, n\} \setminus I$. Suppose an honest prover A' is given $(x, w) \in R_{\mathcal{F}}$. For $j \in I$, A' has $(x_j, w_j) \in R$. Thus, for such j, A' can successfully (with probability 1) execute the protocol (A, B) on common input x_j, where A' plays the role of A, and B is an honest verifier. In contrast, for $j \in I'$, A' is not given a witness w_j such that $(x_j, w_j) \in R$. Here, all A' can do, is invoke the simulator for (A, B)

and prepare a random accepting conversation.

In the protocol (A', B'), the prover A' will compute a first message a_j for each $j \in I$, as the prover A would have done in the protocol (A, B), given $(x_j, w_j) \in R$. Now for $j \in I'$, the prover A' will prepare a simulated accepting conversation (x_j, a_j, s_j, z_j), where s_j is computed in advance (as explained hereafter) and given as input, together with x_j, to the special simulator M for (A, B).

About the computation of the s_j, when $j \in I'$: by definition of the dual of a monotone function (see Section 3.2.2), we have $f_n^*(I') = 0$. Moreover, we have assumed that $\mathcal{F}^*$ has an efficient secret sharing scheme with completion. To generate the s_j, we take such a sharing scheme for f_n^*, and compute the n shares on input of some arbitrary string (to the `dist` algorithm). We now select the shares s_j corresponding to I' from the full set of n shares, and delete the remaining ones. Note that since $f_n^*(I') = 0$, we can later extend these shares s_j for $j \in I'$ to a full set of shares, consistent with any given string c and with the right distribution (this follows from the completion property).

In the first step of the protocol (A', B'), the prover A' sends all a_j ($j = 1 \ldots n$) computed as discussed above, to the verifier B'. In the second step, the verifier B' sends a random challenge c. With respect to this c, the prover extends the shares s_j, $j \in I'$ so that the full set of shares $s_1, \ldots, s_n$ is consistent with c. Finally, for $j \in I$ the prover A' computes the responses z_j as the prover A would have done given the challenge s_j and the first message a_j. Since A' knows a witness for x_j if $j \in I$, this is no problem. In the last transmission, the prover sends all z_j and s_j. Those z_j where $j \in I'$ are selected from the simulated conversations that A' computed in advance. The verifier checks all conversations and checks whether the full set of shares is consistent with the challenge c (using the algorithm `comp`).

Why should this work? Informally, the idea is as follows. Intuitively, it is clear that for $j \in I'$, the prover A' can effectively answer, as detailed above, one challenge *in the protocol* (A, B), given a fixed first message a_j. If A' can answer two different challenges *in the protocol* (A', B'), given fixed first messages $a_1, \ldots, a_n$, then we must have that $f_n^*(I') = 0$; since the s_j, for $j \in I'$, were chosen in advance, they must be the same in those two conversations (with same first messages, but different challenges) in the protocol (A', B'). But $f_n^*(I') = 1$ would imply that the challenge string they are consistent with is uniquely determined! This contradicts the assumption that the challenges were different in those two conversations. Thus, we have $f_n^*(I') = 0$, and hence, $f_n(I) = 1$. Finally, honest verifier zero knowledge follows from the properties of the simulator M and the properties of secret sharing scheme with completion: the simulator ensures that the first messages a_j, $j = 1, \ldots, n$ reveal no information about the set I, while our sharing scheme makes sure that the joint distribution of the

s_j, $j = 1, \ldots, n$, hides the set I.

Corollary 3.2 shows that the protocol (A', B') is *witness indistinguishable*, and in Theorem 3.4 and its corollaries, we show that, under the assumptions that (A, B) has an invulnerable generator and that the family $\mathcal{F}$ satisfies some additional property, the resulting protocol (A', B') is also *witness hiding.*

THEOREM 3.3 *Let (A, B) be a Σ-protocol for relation R and generator G, satisfying honest verifier zero-knowledge and collision intractability (resp. the collision-property or special soundness). Let $\mathcal{F}$ be a family of monotone functions such that its dual $\mathcal{F}^*$ has an efficient secret sharing scheme $\mathcal{S}^*$ with completion. Then there exists a Σ-protocol (A', B') for relation $R_{\mathcal{F}}$ satisfying special honest verifier zero-knowledge and collision intractability (resp. the collision-property or special soundness).*

PROOF. By Proposition 2.5, we may assume that (A, B) satisfies *special* honest verifier zero knowledge. If (A, B) already has a special simulator, we do not need the transformation from Proposition 2.5. Let M denote the special honest verifier zero knowledge simulator for (A, B). By definition of honest verifier zero knowledge, a simulator will run in probabilistic polynomial time, if $x \in RX$. Nothing is required about what happens if $x \notin RX$. It may be that even then the simulator outputs an accepting conversation. If not, we assume that that it halts in polynomial time, and outputs "?". We assume for simplicity that $\mathcal{S}^*$ distributes 1-bit secrets. Let $t_B(k)$ be the size of the challenges in (A, B).

Let $l_i(n)$ denote the size of the i-th share when $\mathcal{S}^*$ distributes one bit and let $K(n)$ denote $\max_P 1 \leq i \leq n l_i(n)$, the size of the largest share. If any other share has less bits, standard padding techniques (for instance padding with zeroes) are applied to the effect that all shares have this same number of bits. So we may assume that $\mathcal{S}^*$ gives shares of size $K(n)$ bits. Now we define the efficient secret sharing scheme $\mathcal{S}^*_{n,k}$ with completion by taking $t_B(k)$ parallel executions of $\mathcal{S}^*$ to make sure that the new scheme distributes secrets of size $t_B(k)$ bits[3] Note that the size of each share is now $K(n) \cdot t_B(k)$ bits.

Next we will tailor (A, B). The Σ protocol $(A, B)_{n,k}$ is just $K(n)$ parallel executions of (A, B) for each k. The algorithms $a(\cdot)$, $z(\cdot)$, $\phi(\cdot)$ and $M(\cdot, \cdot)$ that constitute (A, B) change accordingly. However, we will use these same notations to refer to the corresponding algorithms for $(A, B)_{n,k}$, leaving any

[3]The protocol description to follow uses the algorithms `dist`, `comp` and `cons`. Any necessary coinflips are left out and are assumed to be understood. To emphasize that we are working with a scheme for $\mathcal{F}^*$ rather than $\mathcal{F}$, we consider f_n^* as part of the input. For convenience, `comp` takes as input a secret c, a set of shares for a set I' with $f_n^*(I') = 0$, and produces the completion, that is, the shares for $\{1, \ldots, n\} \setminus I'$.

coinflips implicit. Observe that $(A, B)_{n,k}$ trivially satisfies special honest verifier zero knowledge and special soundness and that it takes challenges of size $t_B(k) \cdot K(n)$ bits. The size $t_{B'}(k)$ of challenges in the target protocol $(A', B')_{n,k}$ will be equal to $t_B(k)$.

The prover A' is given $(x, w) \in R_{\mathcal{F}}$. Put $x = (f_n, x_1, \ldots, x_n)$ for some n and $f_n \in \mathcal{F}_n$, while for some k we have $|x_i| = k$, $i = 1 \ldots n$. Furthermore, $w = \{w_j\}_{j \in I}$ with $f_n(I) = 1$, $\emptyset \neq I \subset \{1, \ldots, n\}$, and $(x_j, w_j) \in R$ for $j \in I$. Let I' denote $\{1, \ldots, n\} \setminus I$. Note that we have $f_n^*(I') = 0$. Let t denote $t_{B'}(k)$. The common input is x.

Move 1: For $j \in I'$, A' starts by computing $\{s_j\}_{j \in I'} \leftarrow \mathtt{dist}(0, f_n^*)|_{I'}$. Next, A' runs $M(x_j, s_j)$, which results either in an accepting conversation (x_j, a_j, s_j, z_j) or '?'. In the latter case, A' puts $a_j = (\text{'?'}, s_j)$.

For $j \in I$, A' computes $a_j \leftarrow a(x_j, w_j)$ as in the protocol $(A, B)_{n,k}$.

For $j = 1 \ldots n$, A' sends a_j to B'.

Move 2: B' selects c at random from $\{0,1\}^t$ and sends c to A'.

Move 3: For $j \in I$, A' computes $\{s_j\}_{j \in I} \leftarrow \mathtt{comp}(c, \{s_j\}_{j \in I'}, f_n^*)$, and computes $z_j \leftarrow (z(x_j, w_j, s_j), s_j)$ where the algorithm $z(\cdot)$ is as in the protocol $(A, B)_{n,k}$.

For those $j \in I'$ where $a_j = (\text{'?'}, s_j)$, he puts $z_j = \text{'?'}$.

Finally, for $j = 1 \ldots n$, A' sends z_j to B', who accepts if and only if the following conditions are satisfied. (1) $\mathtt{cons}(c, s, f_n^*) = \mathit{accept}$ where $s = (s_1, \ldots, s_n)$ and (2) there is at least one j where no '?' is involved and for all such j the conversation (x_j, a_j, s_j, z_j) is accepting in $(A, B)_{n,k}$.

Completeness of the protocol is proved by inspection of the protocol and by taking into account the completion property of the secret sharing scheme $\mathcal{S}^*$ in particular. As for the *collision-property*, let two accepting conversations with the same first messages be given where the challenges c and c' differ. Consider the first messages $a = a_1, \ldots, a_n$ and $a' = a_1', \ldots, a_n'$. For each j we have $a_j = a_j'$. Note that for some j (but not for all) we may have $a_j = a_j' = (\text{'?'}, s_j)$ with corresponding reply $z_j = z_j' = \text{'?'}$. Call such j degenerate and non-degenerate otherwise. Let J' denote the set of j such that $s_j = s_j'$. Since we have that $c \neq c'$ and that $s_1, \ldots, s_n$ and $s_1', \ldots, s_n'$ are consistent[4]

with c and c' respectively, it must be the case that $f_n^*(J') = 0$. Thus its complement J (that is, $J = \{1, \ldots, n\} \setminus J'$) must satisfy $f_n(J) = 1$ by definition of the dual and the fact that $J \neq \emptyset$. But by the fact that $a = a'$ all

[4]Note that, by definition of $\mathtt{cons}$, we are not relying here on any assumption about *how* these s_j's and s_j''s are generated.

$j \in J$ are non-degenerate. Thus, for $j \in J$ we can immediately invoke the special soundness of $(A, B)_{n,k}$ and we obtain a witness w_j for x_j. *Collision-intractability* or *special soundness*, if this assumed for (A, B) instead of the collision-property, is argued in the same way (in case of special soundness, the verifier must also check whether $f_n \in \mathcal{F}$).

Finally, we argue that the protocol is *special honest verifier zero knowledge.* Let x be given as above. The simulator runs as follows. As before let M denote the special simulator for the protocol $(A, B)_{n,k}$. For any $c \in \{0,1\}^{t'_B}$ compute $\{s_j\}_{i=1}^{n} \leftarrow \mathtt{dist}(c, f_n^*)$. Then run M on input x_j, s_j for each j. We first observe the following property of M. On input (x_j, s_j), where $x_j \in RX$ and s_j is of the proper size, M outputs an accepting conversation x_j, a_j, s_j, z_j with distribution equal to the distribution generated by $(A, B^*(s_j))_{n,k}$ on common input x_j. Here the prover A gets $(x_j, w_j) \in R$ and the verifier B^* always chooses s_j as the challenge. This means that in fact we can simulate perfectly the protocol $(A, B^*)_{n,k}$ where B^* is any verifier who selects the challenge independently of the prover's first message and is given as a black-box (see Lemma 2.4). Secondly, the distribution of the s_j is exactly as in real life, due to the completion property of the secret sharing scheme S^*. Thirdly, if $x_j \notin RX$, the simulator faces the same situation as the real prover: although the verifier B' may learn that some $x_j \notin RX$ and that henceforth the prover A' does not know a witness for that x_j (since it does not exist in this case), B' would have found out by itself, that is, without talking to A' and running the simulator instead. The special honest verifier property of the protocol (A', B') now follows immediately from these observations. □

By Lemma 2.5, we have the following.

COROLLARY 3.2 *The protocol (A', B') is also witness indistinguishable.*

REMARK 3.2 *If t_B is "large enough" as a function of the security parameter k (say for instance t_B is linear in k), then (A, B) and (A', B') are proofs of knowledge. If not, then we can always take an appropriate number of parallel executions.*

REMARK 3.3 *Suppose the secret sharing scheme S^* satisfies the following additional property. Let any $f_n^* \in \mathcal{F}^*$ be given and a subset $J \subset \{1, \ldots, n\}$ with $f_n(J) = 0$. Consider the distribution $\mathtt{dist}(s, f_n^*)|_J$ for any s. If the joint distribution of the shares for J is uniform and independent of anything else, and if (A, B) does not satisfy special honest verifier zero knowledge, then we can do without the quite costly transformation that first maps (A, B) into a new protocol that satisfies special honest verifier zero knowledge. This extra property is met by Shamir's threshold secret sharing scheme.*

We now give conditions on the family $\mathcal{F}$ of monotone functions in order that the protocol (A', B') witness hiding. For any (invulnerable) generator

G of relation R, we let G^n denote the generator that produces an n-tuple of pairs in R by running G independently n times in parallel. G^n proceeds as follows, when it is given any $f_n \in \mathcal{F}_n$. It may discard any subset of the n witnesses such that the remaining ones comply with f_n.

THEOREM 3.4 *Let G be an invulnerable generator for relation R, and suppose that for each $f_n \in \mathcal{F}$ there are at least two minimal sets. Then the protocol (A', B') is witness hiding over G^n and $R_{\mathcal{F}}$.*

PROOF. Let any $f_n \in \mathcal{F}_n$ be given. Now put $N = \{1, \ldots, n\}$ and define $I \subset N$ as the intersection of all sets $J \subset \{1, \ldots, n\}$ that are minimal with respect to f_n (see Section 3.2.2). The condition that for f_n there are at least two minimal sets, is equivalent to the condition that $f_n(I) = 0$. It follows that $i \in I$ if and only if $f_n(N \setminus \{i\}) = 0$. To determine the set I, evaluate f_n at the characteristic vector of $N \setminus \{i\}$ for each $i \in N$. By our assumption on $\mathcal{F}$ (at least two minimal sets for each $f_n \in \mathcal{F}$), it follows that $N \setminus I$ contains at least two elements (the elements in the symmetrical difference of two such sets). We follow the line of reasoning from Theorem. 4.3 of [64]. Suppose we are given a probabilistic polynomial time enemy $\mathcal{A}$ that has non-negligible probability of computing a witness, using the honest prover in the scheme above as a subroutine. We show that $\mathcal{A}$ can be compiled into an algorithm that solves with non-negligible probability random instances x generated by G, thus contradicting the invulnerability of the generator (see [1]). Our compilation now works as follows:

1. Recall that our input is a problem instance x generated by G. We now form an instance $(x_0, w_0) \in R_{\mathcal{F}}$ as follows: choose at random $j \in N \setminus I$, and let $x_j = x$. For all other indices i, run G to produce a solved instance $(x_i, w_i) \in R$. Now put $x_0 = (f_n, x_1, \ldots, x_n)$ and $w_0 = \{w_i\}_{i \neq j}$.

2. Give x_0 as input to $\mathcal{A}$. When $\mathcal{A}$ needs to interact with the prover, we simply simulate the prover's algorithm on input (x_0, w_0) from Theorem 3.3. This can be done because we know witnesses of all instances except x_j, and the fact that $j \notin I$ guarantees that $f_n(N \setminus \{j\}) = 1$.

3. If $\mathcal{A}$ is successful, it outputs a witness w_0' such that $(x_0, w_0') \in R_{\mathcal{F}}$. If so, put $w_0' = \{w_i\}_{i \in J}$, where $J \subset N$ with $f_n(J) = 1$ and $(x_i, w_i') \in R$ for each $i \in J$. Else output something random.

We now show that this compilation finds a witness for x with non-negligible probability. First note that the joint distribution of the x_i's we give to $\mathcal{A}$ is the same as in an ordinary interaction with the prover. Therefore $\mathcal{A}$ is successful with non-negligible probability. We therefore only have to bound the probability that j is in J, the set of witnesses we get from $\mathcal{A}$. Since $f_n(I) = 0$, J must contain at least one index not in I. By witness indistinguishability from Corollary 3.2, $\mathcal{A}$ has no information about which j in $N \setminus I$ we have chosen, and so the probability that $j \in J$ is at least $1/|N \setminus I|$. Hence if $\mathcal{A}$ has success probability ϵ, we have success probability at least ϵ/n, which is non-negligible. □

This result can be interpreted as follows. Suppose that n instances of R are generated according to G, let any $f_n \in \mathcal{F}_n$ be given. Let some any subset of witnesses be discarded, subject to the condition that the remaining ones are consistent with f_n. Let (x, w) the corresponding element from $R_{\mathcal{F}}$. Then the protocol (A', B') has the property that a malicious (bounded) verifier cannot extract a witness w' for x. However, it does not rule out that the protocol could help him to compute witnesses for a small subset of the x_i's that x is composed of. Ideally, we would like to prove that the enemy cannot compute even a single witness. With a stronger assumption on the access structure, this can be done:

COROLLARY 3.3 *If for each $f \in \mathcal{F}$, the corresponding set I is empty, then no probabilistic polynomial time enemy interacting with the honest prover can with non-negligible probability compute a witness for any of the x_i in the input to the protocol.*

PROOF. Let any $f_n \in \mathcal{F}$ be given. Since $I = \emptyset$ implies that $f_n(I) = 0$, there are at least two minimal sets for f_n, and therefore the proof is the same as for Theorem 3.4, except that it follows from the assumption that the index j is always chosen among all indices. Hence if the enemy outputs at least one correct witness for some x_i with $1 \leq i \leq n$, there is a non-negligible probability of at least $1/n$ that this is the witness we are looking for. □

A certain special case of Theorem 3.3 is interesting in its own right:

COROLLARY 3.4 *Let (A, B) be a Σ-proof of knowledge for relation R, satisfying honest verifier zero-knowledge and special soundness. Then for any positive integers n, d with $d \leq n$ there is a Σ-protocol (A', B') satisfying special soundness and special honest verifier zero knowledge in which the prover shows that he knows d out of n witnesses without revealing which d witnesses are known.*

PROOF. Use Theorem 3.3 with, for example, Shamir's secret sharing scheme and a threshold value of $n - d + 1$. Take into account Remark 3.3, which implies that we do not have to convert (A, B) into a new protocol satisfying special honest verifier zero knowledge. □

COROLLARY 3.5 *Let $n = 2$ and $d = 1$. In this case the prover shows that he knows at least 1 out of 2 solutions. For any generator G generating pairs in R, this protocol is witness hiding over G^2.*

PROOF. Since protocols constructed from Theorem 3.3 are always witness indistinguishable, we can use Theorem 4.2 of Feige and Shamir [64]. □

Note that for this corollary, we do not need the assumption that G is invulnerable, as in Theorem 3.4. To build the protocol of Corollary 3.5, we need a 2 out of 2 threshold scheme. Such a scheme can be implemented by choosing random shares c_1, c_2 such that $c_1 \oplus c_2$ equals the secret. Therefore, in the simple case of Corollary 3.5, the protocol constructed by Theorem 3.3 simply becomes a game where the verifier chooses a random s, and the prover shows that he can answer correctly a pair of challenges c_1, c_2, such that $s = c_1 \oplus c_2$. In the prover's final message, he only has to send c_1 because the verifier can then compute c_2 himself. Hence the communication complexity of the new protocol (A', B') is exactly twice that of (A, B), whence the new protocol is just as practical as (A, B).

3.2.5 *Application to Identification and Signatures*

Suppose we have n users, for example employees of a company, such that the i-th user has a public key x_i and secret key $w_i \in w(x_i)$. Suppose also that certain subsets of users are qualified in the sense that they are allowed to initiate certain actions, sign letters on behalf of the company, etc. This defines a monotone function on the set of users. Theorem 3.3 now gives a way in which a subset of users can collaborate to identify themselves as a qualified subset, without revealing anything else about their identities. This makes good sense, if they are to assume responsibility on behalf of the company, rather than personally.

This also extends to digital signatures, since by using a hash function, any three round proof of knowledge as the one produced by Theorem 3.3 can be turned into a signature scheme by computing the challenge as a hash value of the message to be signed and the prover's first message (this technique was introduced in [66]). By this method, a signature can be computed which will show that a qualified subset was present, without revealing which subset was involved. This may be seen as a generalization of the group signature concept, introduced by Chaum and Van Heyst [34]. One aspect of group signatures which is missing here, however, is that it is not possible later to "open" signatures to discover the identities of users involved (see [39]).

3.3 Linear General ZK Proofs

3.3.1 *Introduction*

As an important application of the theory of partial proofs developed in Section 3.2, we now present *communication efficient* zero knowledge interactive proofs and -arguments (see Section 2.2.3) for general NP-languages.

To this end, it is sufficient to give such a proof for any given NP-complete problem, such as circuit-satisfiability, as any NP-membership problem can be efficiently reduced to it. Thus given an NP-language L, and a bitstring x, a circuit can be constructed that is satisfiable (that is, there exists an assignment to the input bits such that the circuit's output bit equals 1) if and only if $x \in L$. Moreover, such construction carries an original witness of membership for $x \in L$ over to a satisfying assignment of the circuit. We restrict our attention to a zero knowledge treatment of circuit-satisfiability.

More precisely, let $\mathcal{C}$ be a polynomially sized[5] family of Boolean *circuits*, containing just AND-, OR- and NOT-operators. Let $C \in \mathcal{C}$, and let $|C|$ denote the size of C. We construct a zero knowledge proof that C is satisfiable, using communication complexity corresponding to the size of $O(|C|)$ *bit commitments*, and with error probability that is negligible as a function of $|C|$.

Informally, a bit commitment scheme is a game between two parties, where one party takes some secret bit as input. The purpose is to compute a certain value (a "commitment") that "hides" this bit, but at the same time "binds" the committing party to this bit. The commitment is "opened" by releasing the bit and auxiliary input that was used to mask the bit. Most known approaches to general zero knowledge proofs use bit commitment schemes (sometimes with extended capabilities) and have the prover commit to an NP-witness w for a given instance $x \in L$, after the prover and verifier have agreed on a suitable representation of L and a witness verification method, based on, for instance, circuit satisfiability or graph Hamiltonicity, or any other NP-complete language. In the protocol the verifier will challenge the prover about the contents of the commitments, in such a way that the prover only passes if these really contain a witness. Zero knowledge is shown by exhibiting a suitable simulator.

The classical methods of Goldreich, Micali and Wigderson [71] and Brassard, Chaum and Crépeau [24] yielding zero knowledge interactive proofs and -arguments for general NP-languages respectively first construct a protocol that allows a prover to cheat with probability $1/2$, which is then iterated k times to achieve the required confidence level of $1/2^k$. These methods would require $\Omega(mk)$ bit commitments to show for instance satisfiability of a Boolean circuit of size m.

The work of Boyar, Brassard and Peralta [22] provides the first approach that improves on these communication complexities. They present zero

[5]The size of a circuit is the number of operators it consists of. A family of circuits is polynomially sized if the size of each circuit in the family polynomial in its number of input bits.

knowledge proofs for circuit satisfiability using a "sub-quadratic number of commitments. Roughly speaking, for large m and k they achieve zero knowledge interactive proofs using $O(\sqrt{m}k)$ bit commitments. Usually, one sets k equal to the size of the input, yielding roughly $O(m^{3/2})$ bit commitments in this case, hence a sub-quadratic number. Note that from this point of view, the results of [22] use a quadratic number of bit commitments. Kilian [83] later extended their results by using the probabilistically checkable proofs (PCP) of [4]. More precisely, a zero-knowledge interactive proof that a circuit of size m is satisfiable is constructed using[6] $O(m^{1+c_1}) + O(\log^{c_2}(m)k$ ideal bit commitments and having error probability 2^{-k}. For interactive arguments similar results are given, using a collision intractable hash function in addition. The latter result was further improved in [84], resulting in an interactive argument with communication complexity $O(lk \log l)$ bits. Here, and in the following, l is the security parameter for the prover. Thus, in order to cheat with probability larger than 2^{-k}, the prover must solve an instance of size l of a hard computational problem, such as finding a discrete logarithm modulo an l-bit prime[7].

Given a family $\mathcal{C}$ of polynomially sized Boolean circuits, one can easily (as detailed later) and efficiently transform each $C \in \mathcal{C}$ to a Boolean formula Φ_C such that Φ_C is satisfiable if and only if C is. Moreover, $|\Phi_C| = O(|C|)$. In our approach, we take any Boolean circuit C, map it to Φ_C and apply the results concerning partial proofs from Section 3.2 to a monotone formula Φ'_C that is constructed from Φ_C, and a bit commitment scheme with properties detailed below. Our approach results in zero-knowledge interactive proofs for circuit satisfiability (and thus for NP) with error probability 2^{-k} and communication complexity corresponding to $O(m) + k$ commitments; for interactive arguments for NP we get communication complexity $O(m) \cdot max(k, l)$ bits (we count commitments for the proof and bits for the argument to facilitate comparison with [83, 84]).

Comparing this to [83], [84] which were the best results so far, we see that for interactive proofs, the term depending on k has been reduced from $O(\log^{c_2} m)k$ to k. For arguments, our result is inferior to [84] when viewed as a function of m, but superior as a function of the security parameters k and l. Note that our interactive argument has no need for a collision-intractable hash function, we only need commitments with the right properties. Hence our cryptographic assumption is potentially weaker than the ones needed in [84].[8]

[6]Here c_1 is any positive constant and $c_2 = O(1/c_1)$.

[7]More precisely, one can show that if a prover can argue a false statement with success probability $\epsilon > 2^{-k}$, then he can solve the hard problem in time $O(1/(\epsilon - 2^{-k}))$.

[8]Although no example is currently known that would support our needs, and not simultaneously imply a collision intractable hash function.

If one adopts the usual convention of setting the security parameter k equal to the input size, our result implies a zero knowledge interactive proof that proves satisfiability of a circuit of size m with error probability 2^{-m} using $O(m)$ *commitments*. Even if an extremely small PCP would exist, the protocol in [83] would use $\Omega(m \log^{c_2} m)$ *commitments* to solve the same problem. To the best of our knowledge, our protocol is the first to achieve "linear zero-knowledge" in this sense. For arguments, we get $O(m^2)$ *bits* using $l = k = m$, where [84] would be $O(m^2 \log m)$ *bits*.

The properties we need from the bit commitment scheme are as follows. First, we require that it allows a *linear proof of contents*. This is a Σ-proof of knowledge of the contents of a commitment satisfying honest verifier zero-knowledge and special soundness. The size of conversations is required to be linear in the size of a commitment. Furthermore, we will assume that negations of commitments can be computed non-interactively given just the commitment. In case of zero knowledge interactive proofs, we need the bit commitment scheme to be unconditionally binding. In the case of interactive arguments, we need the bit commitments to be unconditionally hiding and to possess a "trapdoor" that allows a party who holds that trapdoor to construct commitments that can be opened both as "0" and "1". Finally, there must exist a linear size witness hiding proof of knowledge of the trapdoor.

The constants involved in the communication complexities we achieve are small enough for the protocols to be practical in a realistic situation: let n be the number of times the formula Φ reads an input variable. Then the communication complexity of the protocols when using our concrete commitment schemes can be more precisely stated as at most $4n + k + 1$ commitments for the interactive proof and at most $5nl + 5l$ bits for the argument (assuming $k \leq l$). By contrast, the PCP-based methods of [83], [84] hardly have any practical relevance because of the elaborate reductions needed to build a PCP.

REMARK 3.4 *For the case of interactive proofs, we have, like [83], ignored in the statement of results the communication needed to set up the commitment scheme* [9]. *This is reasonable, as the same commitment scheme can be reused in many proofs. For arguments, however, an attractive point is that cheating is only possible if the intractability assumption used is broken* while the protocol is running[10]. *This, however, is only true if a new instance of the commitment scheme is chosen in every run of the protocol.*

[9]In any real implementation, the verifier needs to receive some public parameters of the commitment scheme, and possibly a zero-knowledge proof that they were chosen correctly

[10]In contrast to the situation for proofs, where breaking the assumption at any later time can cause problems

Our communication complexity for arguments therefore includes communication for setting up the commitment scheme. We also remark that in all our protocols, the verifier only sends public coins, so they can be made non-interactive using the Fiat-Shamir heuristic.

3.3.2 Bitcommitments and Linear Proofs of Contents

Bit commitment schemes of the kind we use consist of functions $\texttt{commit} : \{0,1\}^{l_r} \times \{0,1\} \rightarrow \{0,1\}^{l}$ and $\texttt{verify} : \{0,1\}^{l} \times \{0,1\}^{l_r} \times \{0,1\} \rightarrow \{accept,\ reject\}$, whose descriptions are output by a probabilistic polynomial time algorithm G on input 1^l, where l is a security parameter. Here, l_r is polynomially bounded in l. We refer to `commit` and `verify` as the *public key* of the commitment scheme. To commit to a bit b, one chooses r at random from $\{0,1\}^{l_r}$ and computes the commitment C as $C \leftarrow \texttt{commit}(r, b)$. The value r masks the bit b. To verify whether a commitment has been opened correctly, one verifies whether $\texttt{verify}(C, r, b) = accept$.

For interactive proofs, we will need bit commitments to be *unconditionally binding*. This means that the bit b is uniquely determined from the commitment C. Of course we also need the scheme to hide the bit, but the best we can get in this case, is that it is *computationally hiding*: the distributions of commitments to 0 and and to 1, respectively, are computationally indistinguishable: no probabilistic polynomial time algorithm receiving as input a commitment to 0 or 1, can guess the bit b with probability significantly better than $1/2$.

For interactive arguments, we will use bit commitment schemes with dual properties. Unconditionally hiding. This means that the distributions of commitments to 0 and and to 1, respectively, are identical. Now, with respect to the binding property, the best we can achieve is that the scheme is computationally binding. This means that no probabilistic polynomial time algorithm can compute a commitment that can be opened in both ways: it is infeasible to compute $C \in \{0,1\}^{l}$, and $r_0, r_1 \in \{0,1\}^{l_r}$ such that $\texttt{verify}(C, r_0, 0) = \texttt{verify}(C, r_1, 1) = accept$, except with negligible probability.

Unconditionally hiding commitment may in addition be *trapdoor*[24] (also called *chameleon*). For a trapdoor commitment, the generator G outputs in addition a string T called the *trapdoor information*. Given the trapdoor, one can cheat the commitment scheme. More formally, there is a polynomial time algorithm that on input T will produce pairs r_0, r_1 such that $\texttt{commit}(r_0, 0) = \texttt{commit}(r_1, 1) = C$, $\texttt{verify}(C, r_0, 0) = \texttt{verify}(C, r_1, 1) = accept$, and the distribution of C is the same as that of $\texttt{commit}(r, b)$ for random r. We will assume that, on the other hand, given C and any pair r_0, r_1 such that $\texttt{verify}(C, r_0, 0) = \texttt{verify}(C, r_1, 1) = accept$, it is easy to compute T. Note that by the binding property, it is infeasible to compute the trapdoor information T given just the public key of the commitment scheme. Finally, for an unconditionally hiding trapdoor commitment scheme, we require that there exists a witness hiding [64] proof of knowledge of the trapdoor T, with communication complexity linear in l.

Let $C = \texttt{commit}(r, b)$. We say that commitments can be *negated*, if, given just C, one can efficiently compute C' which is a commitment to $1 - b$. Furthermore, if one is given r and b, it is easy to compute r' such that $C' = \texttt{commit}(r', 1 - b)$.

A bit commitment scheme has a *linear proof of contents*, if there is a Σ-protocol (A, B) with the following properties.

1. (A, B) is a *proof of knowledge*, satisfying *special soundness*, that A knows how to open C as a commitment to 1. More precisely, from two conversations that constitute a collision, one can efficiently compute r such that $C = \texttt{commit}(r, 1)$.
2. (A, B) is *special honest verifier perfect zero knowledge*, with simulator M. We assume for simplicity that the simulator always produces an accepting conversation, even if it is given a commitment to 0 (this holds in our concrete examples).
3. The *size* of the conversation is $O(l)$ bits and the challenge size t_B is linear in l. By Lemma 2.8, we may assume that $t_B = l$, while the size of the conversation is still $O(l)$ bits.

From the proof of Theorems 3.5 and 3.6, it will become clear why we require the bitcommitment scheme to have a linear proof of contents. As a toy example, suppose that a prover wants to demonstrate knowledge of a satisfying assignment to the $\Phi = a \oplus b$. Let a bit commitment scheme with negation and linear proof of contents[11] be given. Suppose that the prover has the satisfying assignment $(a, b) = (1, 0)$. Then, the prover provides a commitment C to $a = 1$ and shows that among C and its negation C' (which the verifier can compute on its own), there is at least one commitment to 1. This is done using the techniques from Section 3.2. But then the verifier must conclude that the bits hidden in C and C' respectively, satisfy Φ: one of the bits must be 1 by the properties of the proof and if the other bit were 1 too, this would mean that the prover has broken the commitment scheme! Also by the results in Section 3.2 and the properties of the bit commitments, this approach (which is detailed in Section 3.3.6) can handle any Boolean formula Φ. This then leads to communication efficient honest verifier zero knowledge proofs of satisfiability. Using known transformations, we can turn them into a zero knowledge proofs.

[11]Note that since commitments to 0 are assumed to be indistinguishable from commitments to 1, the simulator M outputs an accepting conversation on input of a commitment to 0, except with negligible probability. Also, if the commitment scheme has the negation property, then there exists a linear proof of contents that A can open a given commitment as 0, as well.

Concrete examples of the commitments schemes defined here are given in Section 3.3.6.

3.3.3 General Approach

We first present a general method for constructing a *communication efficient* perfect *honest verifier* zero knowledge proof (A', B') that a given word x is a member of an NP-language L. Then in the following two sections, we show how to obtain zero knowledge in general for these interactive proofs, resp. arguments.

Let $\mathcal{C}$ be a polynomially sized family of Boolean circuits that verify witnesses for L. First we prove the following.

PROPOSITION 3.1 *Let $\mathcal{C}$ be a polynomially sized family of Boolean circuits. Then there exists an efficient transformation that takes $C \in \mathcal{C}$ to a Boolean formula Φ_C that is satisfiable if and only if C is. Given a satisfying assignment for C, a satisfying assignment for Φ is computed efficiently. Finally, we have that $|\Phi| = O(|C|)$.*

PROOF. Without loss of generality, assume that each circuit consists of AND-, OR-, and NOT-gates only. Let $C \in \mathcal{C}$ and let $|C|$ denote the size of C, that is, the number of gates in C. Suppose that C takes r input bits. First note that if we have a gate in $C \in \mathcal{C}$ with N' input wires, and we assign distinct Boolean variables $a_1, \dots, a_{N'}$ to each of its input wires, and *one* Boolean variable b to the output wires, there exists a Boolean *formula* f that checks the computation at this gate. For instance, suppose we have an AND-gate, then the Boolean formula "$b = a_1 \wedge \cdots \wedge a_{N'}$" does the trick. Since equality-testing of two Boolean variables b_1 and b_2 amounts to "$(b_1 \wedge b_2) \vee (\overline{b_1} \wedge \overline{b_2})$", we can make this Boolean formula f that is satisfiable if and only if the computation at this gate is correct. Number the input bits from 1 to r and the gates from 1 to $|C|$, where the $|C|$-th gate is the output-gate of C. For the i-th gate $(1 \leq i \leq |C| - 1)$, assign the Boolean variable λ_i to each of its output wires. For the j-th input bit, assign the Boolean variable ρ_j to each of its wires. Using the variables just defined, construct the Boolean formula f_i $(1 \leq i \leq |C| - 1)$ that checks the computation at the i-th gate, as detailed above. The Boolean formula $f_{|C|}$ for the output-gate of C, is just the output of that gate in terms of its input variables. The Boolean formula Φ_C is now defined as

$$\Phi_C = \bigwedge_{i=1}^{|C|} f_i.$$

It is clear that Φ_C is satisfiable if and only if C is (note, however, that Φ_C and C do not compute the same function) and that satisfying assignments for C are mapped efficiently to satisfying assignments for Φ_C. Since we may assume that there exist a constant N such that for each $C \in \mathcal{C}$ and for each gate in C there are at most N input wires, we have that $|\Phi_C| = O(|C|)$. Note also that at the cost of a constant expansion of the formula, we may assume that the AND-, and OR-gates have two input bits. □

We may now assume by Proposition 3.1 that we have a family of Boolean formulas Φ that verify witnesses for L, with $|\Phi| = O(|C|)$, $\Phi \equiv \Phi_C$ and $C \in \mathcal{C}$. Given, additionally, a bit commitment scheme with negation and linear proof of contents (A, B), we construct a family $\mathcal{F}$ of *monotone* formulas Φ' from those Φ's, and invoke Theorem 3.3 and Corollary 3.1. The prover P will commit to a witness w for x, after which the prover P and verifier V will run the protocol (A', B'), P running A', and V running B' as subroutines, respectively. P will only be accepted by V if the bits committed to constitute a witness for x, i.e, $x \in L$. This results in interactive proofs and arguments for L that are *honest verifier* zero knowledge. For interactive proofs, we will require the bit commitments to be unconditionally binding, while for arguments we will require them to be unconditionally hiding.

Let an input word $x \in L$ of length k bits be given, and let Φ be a Boolean formula verifying a witness for x. Without loss of generality we may assume that Φ consists of AND-, OR- and NOT-operators only, with the NOT-operators occurring at the inputs[12]. Let m denote the number of different input variables to Φ, and let n denote the number of times that Φ reads an input variable. A monotone formula Φ' is obtained from Φ by removing the negations and by renaming the input variables such that all n references to the inputs refer to different variables. For example, put $\Phi = (a \wedge b) \vee (\neg a \wedge \neg b)$, and note that $n = 4$ and $m = 2$. Now we have $\Phi' = (a \wedge b) \vee (c \wedge d)$, with $n = n' = m' = 4$. In general, Φ' will satisfy $n = n' = m'$.

Let Ψ be any monotone formula on n input variables, and let a set of commitments $D_1, .., D_n$ be given. Then we say that the set of strings $r_1, ..., r_n$ *Ψ-opens* $D_1, ..., D_n$ if $\Psi(\gamma_1, ..., \gamma_n) = 1$, where $\gamma_i = 1$ if and only if $\mathtt{verify}(D_i, r_i, 1) = \mathit{accept}$ $(i = 1 \ldots n)$. Note that by Corollary 3.1, there exists an efficient secret sharing scheme with completion for Ψ^*. This scheme, that of Benaloh and Leichter [14], can take as input a secret bit string of l bits, for any positive integer l. If we assume that Ψ reads each of the n inputs once, the size of each of the n shares output by the scheme is also l bits.

We now incorporate the bit commitments from Section 3.3.2. Let R be the binary relation consisting of all pairs (C, r) such that $\mathtt{verify}(C, r, 1) = \mathit{accept}$. Note that the linear proof of contents (A, B) that comes with our bit commitment schemes, is actually a proof of knowledge for the relation

[12]Pushing the NOT-operators to the inputs does not change the complexity of the formula, and can be done efficiently as well.

R from above. Recall that we have assumed that both the challenge size t_B is equal to l and the size of conversations in (A, B) are linear in l. Here, l is the size of a commitment. Taking into account that (A, B) also satisfies special soundness and special honest verifier zero knowledge, we have by Theorem 3.3 and Corollary 3.1:

PROPOSITION 3.2 *Let $\mathcal{F}$ be a family of monotone Boolean formulas of polynomial size, such that for each n, each $f_n \in \mathcal{F}$ reads each of its n input bits exactly once. Let a bit commitment scheme be given which has a linear proof of contents (A, B). If commitments $D_1, ..., D_n$ of size l and $\Psi \in \mathcal{F}_n$ are given, then there exists an honest verifier zero knowledge Σ-protocol (A', B') showing that A' can Ψ-open $D_1, ..., D_n$. Furthermore, from two conversations of (A', B') of form $(a, c, z), (a, c', z')$, where $c \neq c'$ one can efficiently compute a set of strings that Ψ-opens $D_1, ..., D_n$. Thus (A', B') is a proof of knowledge for relation $R_{\mathcal{F}}$, satisfying special soundness. The communication complexity is l bits plus n times that of (A, B). This corresponds to $O(|\Psi|)$ bit commitments of size l.*

Combining Proposition 3.2 and Corollary 2.1 we have the following.

COROLLARY 3.6 *Suppose that A^* is a prover accepted by the honest verifier B' with probability $\epsilon > 2^{-l}$. Then there exists a probabilistic algorithm Ext that outputs a set of strings that Ψ-opens $D_1, ..., D_n$, running A^* as a rewindable black-box, with expected running time polynomial in T_{A^*} and $1/(\epsilon - 2^{-l})$, where T_{A^*} denotes A^*'s running time.*

We now consider the following honest verifier zero knowledge protocol (P', V') for showing that the Boolean formula Φ is satisfiable.

Step 1 : Let $x \in L$ and let a witness $w = (w_1, .., w_m)$ be given by input bits that satisfy Φ. For $i = 1 \ldots m$, P' now computes commitments C_i for these bits w_i: P' puts $C_i \leftarrow \texttt{commit}(r_i, w_i)$, where r_i is chosen at random from $\{0,1\}^{l_r}$ (notations as in Section 3.3.2). P' sends these C_i's to V'.

Step 2 : For $i = 1...m$, V' computes the negation, a commitment containing $1 - b_i$, C_i' from C_i. Number the positions in Φ where an input bit is used (at the input wires) from 1 to n. For $j = 1 \ldots n$, let $D_j = C_i$, if the bit w_i is used at this position, and let $D_j = C_i'$ if the bit $1 - w_i$ is used.

Step 3 : Using the protocol (A', B') guaranteed by Theorem 3.2, P' now convinces V' that that the bits contained in $D_1, ..., D_n$ satisfy the monotone formula Φ' (that is, $D_1, ..., D_n$ can be Φ'-opened). Here P' plays the role of A' and V' plays the role of B'.

The protocol (P', V') gives rise to the following theorems.

THEOREM 3.5 *Suppose there exists an unconditionally binding bit commitment scheme, in which commitments can be negated, and which has a linear proof of contents. Then any $L \in NP$ has an honest verifier computational zero-knowledge interactive proof system that proves $x \in L$ with error probability at most 2^{-k} using a total of $O(|x|^c)$ commitments, for some constant c. The constant c only depends on L: it is determined by a family of Boolean circuits verifying witnesses for membership of L.*

THEOREM 3.6 *Suppose there exists an unconditionally hiding trapdoor bit commitment scheme, in which commitments can be negated, and which has a linear proof of contents. Then any $L \in NP$ has a honest verifier perfect zero-knowledge interactive argument that $x \in L$, with communication complexity $O(|x|^c) \cdot max(k,l)$ bits, where c is a constant. The constant c only depends on L: it is determined by a family of Boolean circuits verifying witnesses for membership of L. If a prover P^* can cheat with probability $\epsilon > 2^{-k}$ in time T_{P^*}, the prover can break instances of the commitment scheme of size l with expected running time time polynomial in T_{P^*} and $1/(\epsilon - 2^{-k})$.*

PROOF. In case of interactive proofs, we set $k = l$, and execute (P', V'). For interactive arguments, we execute (P', V') s times in parallel, where s is minimal such that $sl \geq k$. The security properties are invariant under parallel composition. *Completeness* is trivial. As for *soundness*, note that if a prover P^* has probability of success greater than 2^{-k}, then there exist $\rho_1, \ldots, \rho_n$ that Φ'-open $D_1, \ldots, D_n$. Thus, $\Phi'(\gamma_1, \ldots, \gamma_n) = 1$ where $\gamma_j = 1$ if and only if $\texttt{verify}(D_j, \rho_j, 1) = accept$. Recall that by definition, each D_j is equal to C_i or C_i' for some i. Let V_i denote the set of indices j, $1 \leq j \leq n$, such that D_j was set equal to C_i, and let V_i' similarly denote those where D_j was set equal to C_i'. Define the set W as the set of indices j, $1 \leq j \leq n$, with $\gamma_j = 1$. We define the bits w_i and w_i', $i = 1 \ldots m$, by setting $w_i = 1$ if $V_i \cap W \neq \emptyset$ and 0 otherwise, and $w_i' = 1$ if $V_i' \cap W \neq \emptyset$ and 0 otherwise. Note that $w_i = w_i' = 1$ implies that we can find j and j' such that $\texttt{verify}(C_i, \rho_j, 1) = \texttt{verify}(C_i', \rho_{j'}, 1) = accept$. If we assume that at least one of w_i and w_i' is equal to 0 for each i, and put $w_i = 1$, $w_i' = 0$ instead in those cases where $w_i = w_i' = 0$, it is easy to see that $w = (w_1, \ldots, w_m)$ satisfies Φ. In case of interactive proofs, where an unconditionally binding scheme is used, the case $w_i = w_i' = 1$ is impossible, so the error probability is at most 2^{-k}. Now for the case of interactive arguments where unconditionally hiding bit commitments are used, the case $w_i = w_i' = 1$ implies that the prover is breaking the binding property of the commitment scheme: by the properties of negation, $\texttt{verify}(C_i, \rho_j, 1) = \texttt{verify}(C_i', \rho_{j'}, 1) = accept$ implies that C_i can be opened as 0 and 1. Since we may view (P', V') as a Σ-protocol satisfying special soundness with challenge size $sl \geq k$, we have similarly to Corollary 3.6 that the prover can break the commitment scheme with expected time running time polynomial in T_{P^*} and $1/(\epsilon - 2^{-k})$, if the success probability ϵ is greater than 2^{-k}. The latter argument also shows that (P', V') is a proof of knowledge in both cases of interactive proofs and arguments. Concerning *honest verifier zero knowledge* simulation, we construct the C_i's as a set of all-0 commitments, and compute the C_i''s from this. We then invoke (s times, in case of the arguments) the honest verifier simulator of (A', B'). This simulation is perfect for unconditionally hiding commitments, and is computationally indistinguishable for unconditionally binding commitments. Finally, the *communication complexities* are argued as follows. Note that we may assume that $|\Phi| = O(|x|^c)$, where the constant c only depends on the language L. We also have $n = O(|x|^c)$, and that the communication complexity of (P', V') is that of (A, B), except that for arguments it is repeated s times. Therefore, in the case of interactive proofs, we need communication corresponding to $O(|x|^c)$ bit commitments of size k (since we have put $k = l$), and for arguments, we need some $\lceil k + 1/l \rceil \cdot O(|x|^c)$ bit commitments of size l. Since we are counting *bits* in this case, the communication is $O(|x|^c) \cdot max(k, l)$ bits. □

3.3.4 Zero Knowledge Interactive Proofs for NP

The problem that (P', V') is only honest verifier zero-knowledge can be solved using a method due to Okamoto [54], namely to let the verifier's challenge be determined by a two party coinflipping protocol. This turns (P', V') into a zero knowledge proof system for L.

Step 1 : The prover P sends the first message of protocol (P', V').

Step 2 : Let an unconditionally binding bit commitment scheme be given (See Section 3.3.2). For $i = 1$ to k do: the prover P commits to a bit p_i, the verifier V chooses a random bit v_i and sends it to P. Finally, the prover P opens the commitment to p_i.

Step 3 : The prover P sends the final message of P' in protocol (P', V'), taking the challenge of the verifier V' to be $p_1 \oplus v_1, ..., p_k \oplus v_k$.

When the verifier is honest, the challenge is uniformly chosen, so the error probability is 2^{-k}. Moreover, the protocol can now be simulated against a general verifier since by rewinding the verifier, the simulator can force the challenge to be a particular value of its choice, and hence the honest verifier simulator we already had is enough to simulate the rest of the conversation. The coinflipping step costs k commitments.

THEOREM 3.7 *Suppose there exists an unconditionally binding bit commitment scheme, in which commitments can be negated, and which has a linear proof of contents. Then any $L \in NP$ has a computational zero-knowledge interactive proof system that proves that some given $x \in L$ with error probability at most 2^{-k} using a total of $O(|x|^c) + k$ commitments, for some constant c. The constant c only depends on L: it is determined by a family of Boolean circuits verifying witnesses for membership of L.*

3.3.5 Interactive Arguments for NP

To build a zero-knowledge interactive argument (P, V) from (P', V'), we use an unconditionally hiding *trapdoor* bit commitment scheme with an efficient witness hiding proof of knowledge of the trapdoor:

Step 1 : The verifier V runs the key generator G, sends the resulting public key for the commitment scheme to the prover P, and keeps the trapdoor T private.

Step 2 : The verifier gives a witness hiding proof of knowledge of the trapdoor.

Step 3 : Protocol (P', V') is executed using the commitment scheme instance just generated.

The idea is taken from [65]. The protocol as shown here has 6 moves, but this can be condensed to 4 moves in the same way as in [65]. The proof of soundness remains essentially the same, but note that in order to fool V, the prover P still has to break the commitment scheme, as follows from the proof of Theorem 3.6. In the case of *trapdoor* commitments we have required that breaking the commitment scheme is essentially as difficult as finding the trapdoor T. However, the verifier's witness hiding proof does not help to do that. Hence the soundness of (P', V') is preserved. Furthermore, the protocol is now zero-knowledge, since the simulator can use the knowledge extractor for the verifier's proof of knowledge to get the trapdoor information[13]. Given the trapdoor, simulation of the rest of the protocol is trivial. The witness hiding proof costs, by assumption on the commitment scheme, a communication complexity that is $O(l)$ bits. We then get the following by inspection of (P, V):

THEOREM 3.8 *Suppose there exists an unconditionally hiding trapdoor bit commitment scheme, in which commitments can be negated, and which has a linear proof of contents. Then any $L \in NP$ has a perfect zero-knowledge interactive argument that $x \in L$, with communication complexity $O(|x|^c) \cdot max(k, l)$ bits, where c is a constant. The constant c only depends on L: it is determined by a family of Boolean circuits verifying witnesses for membership of L. If a prover P^* can cheat with probability $\epsilon > 2^{-k}$ in time T_{P^*}, the prover can break instances of the commitment scheme of size l with expected running time polynomial in T_{P^*} and $1/(\epsilon - 2^{-k})$.*

[13]Although this by itself may not produce the trapdoor with absolute certainty, the simulator can run an exhaustive search for the trapdoor in parallel with the extractor. This will then produce the trapdoor in those exponentially few cases where the extractor fails, while the expected running time remains polynomial

3.3.6 Existence of Bitcommitment Schemes

This section contains material related to practical implementation of our protocols. We present examples of commitments with the properties we need and compute the concrete communication complexities that result.

An Unconditionally Binding Bit Commitment Scheme

For ***unconditionally binding schemes***, note that we could abstract the properties needed as follows. Suppose we are given an NP-language L_0 where language membership is hard to decide on the average. Suppose furthermore that there exists an efficient "inversion" algorithm that takes a word $x_0 \in L_0$ as input and outputs a word $x_\in L_1$, where L_1 is the complement of L, and vice versa. Note that this implies that $L_1 \in NP$. To commit to 0, the prover selects a random instance $x_0 \in L_0$, and to commit to 1, the prover selects a random instance $x_1 \in L_1$. Negation is also taken care of by the inversion algorithm. To open a commitment to 0, the (unbounded) prover just computes a witness w_0 for x_0. To open a commitment x_1 to 1, the prover first maps x_1 to some $x_0 \in L$ using the inversion algorithm and proceeds as before. If we now assume that there exists a Σ-protocol for L_0 that constitutes an honest verifier interactive proof [14] for L_0 with the right communication complexity, then we can easily construct the proof of contents. If we want a polynomially bounded prover to be able to execute the protocol we also have to assume that there exists an efficient algorithm that uniformly samples $x_i \in L_i$, $i = 1, 2$, and outputs x_i together with a witness w_i. We can get around the inversion function if we assume that L_0 itself has an honest verifier zero knowledge Σ-proof. The prover then selects $x_0 \in L$ and $x_1 \in L_1$, presents the pair (x_0, x_1) in random order and proves in zero knowledge that at least one of the two words is in L_0 *and* that at least one of the two words is in L_1. This is possible by the results from Section 3.4.

Concretely, the following scheme, which can be viewed as an instance of the general treatment above, is based on the factoring problem, and is derived from the identification scheme of Guillou and Quisquater [76].

Key Generation : The key generator G for this scheme chooses an l bit integer n as the product of two primes p_1, p_2 of approximately the same length, such that there is an odd prime q that divides $p_1 - 1$, but not $p_2 - 1$ (easily done by choosing q and p_2 first and then looking for a prime of form $2jq + 1$ for some j). Then a constant h is chosen as a random number in Z_n^* which is *not* a q'th power modulo n. The public key is now n, q, h.

[14]Satisfying special soundness, or a weak variant of it where a prover can answer only very few challenges if the input word is not in L_0, see Section 2.4.2

Commitment : The function `commit` is defined as $C = \texttt{commit}(r, b) \leftarrow r^q h^{\delta(b)} \bmod n$ where r is random in Z_n^*. The function δ takes as input a bit b and outputs $(-1)^{1-b}$.

Opening : To open the commitment C, r and b are revealed. This is accepted if and only if $\texttt{verify}(C, r, b) = \mathit{accept}$, where `verify` checks whether $C = r^q h^{\delta(b)} \bmod n$.

Negation : Commitments can be negated: if $C = \texttt{commit}(r, b)$, then anyone can easily compute $C' = C^{-1} \bmod n = c(r^{-1}, 1 - b)$.

Using the fact that 2 and q are relatively prime, it is not hard to show that we can have $hr^q = h^{-1}r'^q \bmod n$ if and only if h is a q'th power, and the scheme is therefore *unconditionally binding.* After choosing the public key, P should convince V in zero-knowledge that h does not have a q'th root. This is easily done using a variant of the protocol from [68] for quadratic non-residuosity - note that it is easy for P to test if a number y has a q'th root by testing if $y^{\phi(n)/q} \bmod n = 1$. Also note we do not count the communication complexity for this proof, since it only has to be done once in a pre-processing phase. The prover can use the same instance many times for proofs concerning different languages even. Finding the bit contained in a commitment C is precisely the problem of distinguishing cosets of the subgroup of q'th powers in Z_n^* (the *q-th residuosity problem,* a variant of the quadratic residuosity problem). This is a well-known problem, believed to be hard, if factoring is hard.

Hence we conjecture that the scheme is *computationally hiding* under the *q'th residuosity assumption*: the distributions of $r^q h \bmod n$ and $r^q h^{-1} \bmod n$ are computationally indistinguishable, when r is random in Z_n^* and the length of q is linear in the length of n, say corresponding to a constant fraction of the length of $p_1 - 1$.

Finally, the following protocol (A, B) is a *linear proof of contents*:

Move 1 : Let $y = C/h \bmod n$, where $C = hr^q \bmod n$ is the input commitment. Now A chooses u at random in Z_n^* and sends $a = u^q \bmod n$ to B.

Move 2 : B chooses c at random from $[0, 2^t - 1]$ (where t is maximal, such that $2^t < q$) and sends it to A.

Move 3 : A sends to B $z = u \cdot r^c \bmod n$, and B checks that $z^q = a \cdot y^e \bmod n$

To verify that the necessary properties are satisfied, is done in very much the same way as in Section 2.3.3.

An Unconditionally Hiding Bit Commitment Scheme

Ideally, we would like to present here an unconditionally hiding bit commitment scheme with the properties we require, based only on abstract Σ-protocols with properties such as special soundness or collision intractability and (special) honest verifier zero knowledge. As it turns out, our model facilitates trapdoor bit commitments. But at present it seems that the other properties needed in the context of our linear zero knowledge proofs, such as linear proof of contents and negation, are hard to get without making a host of new assumptions. Since trapdoor (or chameleon) commitments are useful in other applications as well, we include the construction here, before we show that bit commitments sufficient for our linear zero knowledge arguments can be constructed under the assumption that a family of special one-way group isomorphisms exists (see Section 2.5.2). Thus the discrete logarithm or RSA-assumptions are sufficient to provide bit commitments of the type needed here.

As to *unconditionally hiding* schemes, let (A, B) be a Σ-protocol for relation R and invulnerable generator G. Suppose the protocol satisfies special soundness and honest verifier zero knowledge, and that membership of RX can be decided efficiently. Let M denote the simulator. The verifier V computes an instance $(x, w) \leftarrow G(1^k)$ and sends x to the prover P, who only proceeds if $x \in RX$. To commit to a bit b the prover runs M on input x until the conversation (x, a, c, z) output by M satisfies that least significant bit of c is equal to b. By the honest verifier property of (A, B) this takes an expected number of two trials. By that same property, the value a gives no information about b. The commitment C is equal to a. To open the commitment, the prover sends c and z to the verifier, who checks whether $\phi(x, a, c, z) = accept$. The verifier takes the least significant bit of c as the bit that the prover committed to. Note that if the prover could break the commitment scheme, the invulnerability of the generator G would be contradicted. Now we must build in the *trapdoor*. If it is the case that the protocol (A, B) is a witness hiding proof of knowledge, we are clearly ready. If this is not the case, we simply apply the transformation from Corollary 3.5 first, before going through the definitions above. One of the variations is where one can commit to many bits simultaneously by using a special simulator for (A, B).

We now prove the following proposition.

PROPOSITION 3.3 *Suppose there exists a family $\mathcal{F}$ of special one-way group isomorphisms (see Section 2.5.2) such that membership of $\mathcal{F}$ can be decided efficiently and that for each $(f, K, L, T) \in \mathcal{F}$ it holds that $\log T$ is linear in $k_f = |L|$, then there exists an unconditionally hiding trapdoor bit commitment scheme with negation, a linear proof of contents and a witness hiding proof of knowledge of the trapdoor.*

PROOF. Let $((f, K, L, T, g, h), (w_1, w_2))$ be an element as output by $\hat{R}_{\mathcal{F}}$'s generator $\hat{G}_{\mathcal{F}}$ on input 1^k where k is the security parameter. Recall that $\hat{f}_g(w_1, w_2) = g^{w_1} f(w_2) = h$, where $w_1 \in [0, t)$ and $w_2 \in K$ and f is an isomorphism from K to L. Also recall that T is an odd prime such that for each $\gamma \in L$ we can efficiently compute $\delta \in K$ such that $f(\delta) = \gamma^T$.

Key Generation : The public key of the commitment scheme consists of (f, K, L, T, g, h), while the trapdoor, kept privately by the verifier, is (w_1, w_2).

Commitment : Let the function δ take as input a bit b, and output $(-1)^{1-b}$. Then the function `commit` is defined by

$$C = \texttt{commit}((r_1, r_2), b) = h^{\delta(b)} \hat{f}_g(r_1, r_2),$$

where r_1 is chosen at random from $[0, T-1]$ and r_2 is chosen at random from K.

Opening : To open a commitment C, the values r_1, r_2 and b are revealed. It is accepted if and only if $\texttt{verify}(C, (r_1, r_2), b) = \textit{accept}$, where verification consists of checking whether $C = h^{\delta(b)} \hat{f}_g(r_1, r_2)$.

Negation : Commitments can be *negated*: given $C = \texttt{commit}((r_1, r_2), b)$, then $C' = C^{-1} = \texttt{commit}((-r_1, r_2^{-1}), 1-b)$ contains $1-b$.

Trapdoor : A trapdoor can be any pair w_1, w_2 such that $h = \hat{f}_g(w_1, w_2)$.

A *linear proof of contents* and a *linear witness hiding proof of knowledge of the trapdoor* are provided by Proposition 2.10. Note that we have assumed that T is large enough. It is easily verified that the trapdoor defined above is indeed a trapdoor and that commitments computed using this trapdoor have the right distribution. Note that if the prover can compute commitments that can be opened in both ways, this will contradict the fact that the protocol (A, B) from Proposition 2.10 is witness hiding. Suppose we are given r_i and r_i' $(i = 1, 2)$ such that $h\hat{f}_g(r_1, r_2) = h^{-1}\hat{f}_g(r_1', r_2')$. Let $\alpha, \beta \in K$ be such that $f(\alpha) = g^T$ and $f(\beta) = h^T$. Note that by our assumptions on T such values are computed easily. Now define the following:

$$w_1 = \frac{1}{2}(r_1' - r_1)(T+1) \bmod T$$

$$s_1 = \frac{1}{2}(r_1' - r_1)(T+1) \text{ div } T$$

$$s_2 = (r_2' r_2^{-1})^{\frac{T+1}{2}}$$

$$w_2 = \alpha^{s_1} \beta^{-1} s_2.$$

Then $\hat{f}_g(w_1, w_2) = h$. Thus the scheme is *computationally binding*. To see that it is *unconditionally hiding*, note that for each $b \in \{0, 1\}$ and $r_1 \in$

$[0, T-1]$ there is a unique $r_2 \in K$ such that $\texttt{commit}((r_1, r_2), b)$ is equal to any given commitment. □

For the sake of concreteness we include an example based on exponentiation in a group of prime order g, G_q, where computing discrete logarithms is hard. See Section 2.3.2. For concreteness, we think here of this group as a subgroup of Z_p^*, where p is a prime and q divides $p-1$. But any group of order q would do, such as an elliptic curve group. Note that we have $q = T$.

Key Generation : To generate the public key of the scheme, choose an l-bit prime p, such that the prime q divides $p-1$ (this is easily done by picking q first of length slightly smaller than l and then searching for p). Also select two elements $g_1, g_2 \in Z_p^*$ of order q at random. Define f and $\hat{f}_{g_2}$ by $f(w) = g_2^w$ for $w \in \mathbb{Z}_q$ and $\hat{f}_{g_2}(w_1, w_2) = g_1^{w_1} g_2^{w_2}$ for $w_1, w_2 \in \mathbb{Z}_q$. Next choose w_1, w_2 at random from $\mathbb{Z}_q$ and let $h = g_1^{w_1} g_2^{w_2} \bmod p$. The public key is now p, q, g_1, g_2, h.

Commitment : Let the function δ be as before. Then the function $\texttt{commit}$ is defined by $C = \texttt{commit}((r_1, r_2), b) \leftarrow g_1^{r_1} g_2^{r_2} h^{\delta(b)} \bmod p$ where r_1, r_2 are random in Z_q.

Opening : To open a commitment C, the values r_1, r_2 and b are revealed. It is accepted if and only if $\texttt{verify}(C, (r_1, r_2), b) = \textit{accept}$, where verification consists of checking whether $C = g_1^{r_1} g_2^{r_2} h^{\delta(b)} \bmod p$.

Negation : Commitments can be *negated*: given $C = \texttt{commit}((r_1, r_2), b)$, then by computing $C' = C^{-1} \bmod p$ which is $\texttt{commit}((-r_1, -r_2), 1-b)$, we have a commitment that contains $1-b$.

Trapdoor : A trapdoor can be any pair u_1, u_2 such that $h = g_1^{u_1} g_2^{u_2} \bmod p$.

The scheme has the following *linear proof of contents* (A, B):

Move 1 : Let $y = C/h$, where $C = g_1^{r_1} g_2^{r_2} h \bmod p$ is the input commitment. Now A chooses $f u_1, u_2$ at random in Z_q and sends $a = g_1^{u_1} g_2^{u_2} \bmod p$ to B.

Move 2 : B chooses $c \in \{0,1\}^t$ at random (where t is maximal, such that $2^t < q$) and sends it to A.

Move 3 : A sends to B: $z_1 = u_1 + cr_1 \bmod q$ and $z_2 = u_2 + cr_2 \bmod q$, who checks that $g_1^{z_1} g_2^{z_2} = ay^e \bmod p$

Concrete Communication Complexities

Assume we use one of the two example commitment schemes shown above to implement our protocols. To evaluate their practical potential, we compute the exact communication complexities that result. In general we can say that in the worst case, the formula Φ uses every input bit and its complement exactly once, so that no reuse of commitments is possible. Hence, in the (A', B') protocol, for each input bit we use at most the commitment containing the bit, and the proof of contents done w.r.t. this bit. For both commitment schemes, we have to choose the parameter l such that either factoring or discrete log is infeasible, which means that l should be $700-1000$. To ensure a chance of at most 1 in a billion of cheating it suffices that $k = 30$, so we reasonably assume that $k \leq l$ in any practical situation.

Thus, we have at most $n \leq |\Phi|$ commitments of size l bits and only one iteration of the protocol is necessary. From the description of the scheme based on factoring, we see that the proof of contents requires communication equivalent to two commitments, plus a challenge value that will always be of length at most 1 commitment. One proof of contents therefore needs communication corresponding to at most 3 commitments. From the description of the scheme from Section 3.3.6 based on discrete log, we see that the proof of contents communicates numbers corresponding to at most 4l bits (3 numbers plus 1 challenge). Taking into account that for interactive proofs we have to go through the transformation of Section 3.3.4 ($k+1$ commitments) and for interactive arguments we have to do the proof of knowledge of the trapdoor from Section 3.3.4 ($5l$ bits for the public key and $4l$ bits for the proof), we have:

PROPOSITION 3.4 *Suppose the protocols from Theorem 3.7, resp. 3.8 are executed using the commitment schemes from Section 3.3.6, resp. 3.3.6, then assuming that $l \geq k$, the communication complexities will be at most $4n+k+1$ commitments, resp. $5nl+9l$ bits, where n is the number of times a Boolean formula for verifying an NP witness for L reads an input variable.*

A simple computation shows that with for example about 3 Mbyte of communication, and using $k = 50, l = 768$, n can be up to about 10.000. This might be enough to prove, for instance, that you know a DES key encrypting a given cleartext block to a given ciphertext block.

In a similar case, the short *discreet proofs* of Boyar and Peralta [23] use significantly less communication, about 0.3 Mbyte, but much more computation - a factor of very roughly $k/10 \log n$ more than our protocol[15]. However, it is currently an open problem to prove anything about the soundness and"discreteness" of the proofs from [23], even assuming hardness of factoring or discrete log. Thus the factor 10 in communication seems to be the price we currently have to pay for provable security.

[15]This is due to the fact that many randomly generated commitments must be used for every gate. These commitments can be generated from a short random seed, and do not have to be communicated. They do have to be computed on by both parties, however.

We note that in general, our protocol may be significantly optimized by building ad hoc as small a formula Φ as possible for the problem. Furthermore it may be possible to find a smaller monotone formula computing the same function as the one constructed directly from Φ. For the interactive proofs, k "extra" commitments are needed to make the proof simulatable against a general verifier. Some of these can be saved by committing to a logarithmic number of bits in one commitment, thus doing the two-party coinflip on a logarithmic number of bits simultaneously.

3.4 Monotone Function Closure of Zero Knowledge

3.4.1 Introduction

Consider a language L (a subset of $\{0,1\}^*$), and assume that there exists a zero knowledge interactive proof (P_0, V_0) for membership of L. Let n words $x_1, \ldots, x_n \in \{0,1\}^*$ be given. How do we give an efficient zero knowledge interactive proof that, for instance, half of these words are members of L? In this section we study structural properties of classes of languages that admit a particular kind of zero knowledge proof. More precisely, given any monotone function f (see Section 3.2.2) on, say, n input bits, we will construct a zero knowledge proof (P, V) that there exists a subset A of the x_i's whose elements are all in L *and* that A is consistent with the monotone condition. In other words, the characteristic vector of that set satisfies f. Given some suitable method for parsing any bitstring $x \in \{0,1\}^*$ into n strings $x_1, \ldots, x_n$, we can then define the language $f(L)$ as consisting of those words x whose corresponding x_i's satisfy the conditions above. In fact, (P, V) is a zero knowledge interactive proof for the language $f(L)$. As an example take f as an (n, d)-threshold function. Then our results provide a zero knowledge proof that at least d out of the n x_i's are members of L.

Our transformation can be applied if (P_0, V_0) is a Merlin-Arthur protocol in that the verifier is required to send only random bits as a challenge. Furthermore, we will assume that (P_0, V_0) consists of at most three moves. If (P_0, V_0) is perfect (resp. statistical) *honest verifier* zero knowledge, then the resulting protocol (P, V) a is perfect (resp. statistical) zero knowledge interactive proof for the language $f(L)$. Under the same conditions, we show that the complement language L' of L has a perfect (resp. statistical) *honest verifier* zero knowledge interactive proof for the language $f(L)$. Under some extra condition about L, this proof is also zero knowledge against an arbitrary polynomially bounded verifier. We stated our results in terms of a fixed monotone function f, but in fact we can achieve similar results if this function is replaced by a family $\mathcal{F}$ of monotone functions and if a suitable efficient secret sharing scheme with completion is available (see

Section 3.2). The corresponding composite language is denoted by $\mathcal{F}(L)$, suitable parsing to be understood for the moment.

Please note that our goals here differ from those in Section 3.3 in the following ways. There, we were concerned the *communication complexity* of zero knowledge proofs for *general NP-languages.* We needed computational complexity assumptions to achieve either perfect zero knowledge proofs that are computationally convincing (argument) or to achieve computationally zero knowledge proofs that are statistically convincing (interactive proofs). Here, we do not restrict our attention to NP-languages. But more importantly, we show how to construct zero knowledge proofs for a larger class of languages from zero knowledge proofs for given (smaller) class, *without* using computational complexity assumptions.

The idea of using monotone operations to build new interactive proofs from known ones with certain properties was introduced by De Santis *et al.* in [103] for statistical zero-knowledge proofs and independently in [47] for witness hiding proofs of knowledge (see Section 3.2).

The results in [103] apply to a quite specialized subclass of languages in PZKIP (the collection of languages with a perfect zero knowledge interactive proof), namely random self-reducible languages (*RSR*) (Tompa and Woll [115]). We show that monotone closure applies to a much more general class, namely languages with 3-move Merlin-Arthur proofs that are statistical honest verifier zero knowledge. In addition to RSR, this class contains for example *perfect and statistical non-interactive zero knowledge.* Non-interactive zero knowledge proofs differ from the standard zero knowledge proofs in that no interaction between a prover and a verifier is required. The only assumption needed is that prover and verifier observe a common random string ρ (the "reference string"). A non-interactive zero knowledge proof of some statement consist of a string p, that the prover constructs as a function of the reference string ρ and a "real" proof of the statement. A dedicated verification procedure can be invoked to check if the proof p is consistent with the reference string ρ and the statement. Note that an interactive proof can be obtained by letting the verifier choose the random reference string, send it to the prover and have the prover reply with the (non-interactive) proof. Hence it contains for example the language of*Blum-integers* [77], the language of so called *2-regular integers* [10]. Our results can also be applied to the language of *functions that are almost permutations* [8]. None of these problems seem to be in RSR.

This substantially extends the set of languages known to be in PZKIP and SZKIP (the collection of languages with a statistical zero knowledge interactive proof), but we also believe that it provides a better understanding of the essential properties needed for monotone closure: It holds if the given proof system has a certain form, and not because of algebraic properties of the underlying language, such as random self-reducibility.

Another important point is that it might seem from [103] and [47] that monotone closure holds only for protocols that satisfy a very strong and non-standard requirement for soundness, namely that a cheating prover can never answer more than 1 of the verifier's potential questions (similar to special soundness). We show that this is in fact unnecessary - the standard requirement for soundness suffices. However, if we are given that the zero knowledge proof we start with, satisfies special soundness (such as is the case with the standard zero knowledge proof for quadratic-residuosity), the techniques from Section 3.2 can immediately be invoked to obtain a zero knowledge proof for the composite language.

The key to our results is a combination of the techniques from [103] and [47] (see Section 3.2) with the interactive hashing technique of Naor et al. [90]. Our results do not use unproven complexity assumptions.

3.4.2 Notation and Definitions

The language L is said to be in HV3AM/PZKIP, resp. HV3AM/SZKIP if it has an Interactive Proof system satisfying the following:

1. It is Perfect, resp. Statistical, Honest Verifier Zero-Knowledge.
2. It is an Arthur-Merlin game with at most 3 moves.
3. Completeness is satisfied with probability 1 (the honest prover always succeeds in convincing the verifier about a true statement).

Such a proof system is called an HV3AM/PZKIP-proof, respectively an HV3AM/SZKIP-proof. Some of our results can be shown without the third condition above, but we have included it mainly to simplify the arguments. We know of no concrete protocol that satisfies the first two conditions, but not the third. Note that for interactive protocols of this kind, having at most three moves, there are essentially the following two possibilities. The first possibility is that the protocol is of the Σ-type. The second possibility is as follows. The verifier starts and sends a random challenge, to which the prover subsequently replies. This situation may be seen as corresponding to, for instance, non-interactive zero knowledge made interactive by letting the (honest verifier) choose the random reference string. We model this possibility as a Σ-protocol, by defining the prover's first message as consisting of the empty string. In the second move, the verifier sends a random reference string as the challenge. The prover responds by sending the "non-interactive" proof in the third move. Thus, we may now assume that the protocols we consider are in fact Σ-protocols. We use the notations as they are defined for Σ-protocols.

A *parsing scheme* is a polynomial time algorithm π which takes a k-bit word x as input and outputs integers $n, i_0, i_1, ..., i_n$, where $0 = i_0 < i_1 < ... <$

$i_{n-1} < i_n = k$. Note that $n \leq k$ and that n may depend on x. Having run π on input x we can view x as a concatenation of words $x = x_1|x_2|...|x_n$, where x_j consists of bits $i_{j-1} + 1$ through i_j of x. A parsing scheme is *admissible* if there is a fixed non-constant polynomial q such that the length of any x_i is at least $q(k)$ (note that q must be linear). This is a technicality which is needed in the following to preserve soundness and statistical zero knowledge.

Let $\mathcal{F}$ be a family of monotone functions (see Section 3.2.2). The language $\mathcal{F}(\pi, L)$ is constructed as follows from $\mathcal{F}, \pi$ and L: given a word x, parse it using π to divide x into $x_1, .., x_n$. Define the bit string $B = b_1, ..., b_n$ by $b_i = 1$ if and only if $x_i \in L$. Then $x \in \mathcal{F}(\pi, L)$ if and only if $f_n(B) = 1$. Note that it is easy to choose $\mathcal{F}$ and π such that for any L, $L = \mathcal{F}(\pi, L)$.

Finally, we need an efficient secret sharing scheme with completion for $\mathcal{F}$ (see Section 3.2.3). Recall that this is basically an efficient perfect secret sharing scheme, where any set of shares for a "non-authorized set", with distribution as in "real life", can be extended to a full set of shares that is consistent with any given secret input. Also recall that if $\mathcal{F}$ is a family of polynomially sized monotone boolean formulas, then such a scheme exists for $\mathcal{F}$ and its dual $\mathcal{F}^*$ (see Theorem 3.1 and Corollary 3.1). In particular, this holds if $\mathcal{F}$ consists of threshold functions (see Theorem 3.2).

3.4.3 Overview

We prove the following results:

THEOREM 3.9 *If $L \in HV3AM/PZKIP$, resp. $L \in HV3AM/SZKIP$ and the monotone function family F^* has an efficient secret sharing scheme with completion, then for any admissible parsing π, $F(\pi, L) \in PZKIP$, resp. $F(\pi, L) \in SZKIP$.*

The following is an immediate consequence of Theorem 3.9 and Corollary 3.1.

COROLLARY 3.7 *If $L \in HV3AM/PZKIP$, resp. $L \in HV3AM/SZKIP$ and functions in the family F have polynomial size monotone formulas then for any admissible parsing π, $F(\pi, L) \in PZKIP$, resp. $F(\pi, L) \in SZKIP$.*

We do not know if Theorem 3.9 is strictly stronger than Corollary 3.7, but conjecture that the answer is yes. Our assumption that F^* has an efficient secret sharing scheme with completion implies that functions in F can be computed in time polynomial in k, but it is known [99] that monotone functions, such as perfect matching in a bipartite graph, exist that can be computed in polynomial time, but require exponential size monotone formulas. However, to the best of our knowledge, no efficient secret sharing schemes are known for the functions studied in [99].

Let $HVSZKIP$ be the class of languages with honest verifier statistical zero-knowledge proofs, and let L' be the complement of L (that is, $L' = \{0,1\}^* \setminus L$). We show that

THEOREM 3.10 *If $L \in HV3AM/SZKIP$ and the family F has an efficient secret sharing scheme with completion, then for any admissible parsing π, $F(\pi, L') \in HVSZKIP$.*

COROLLARY 3.8 *Suppose $L \in HV3AM/SZKIP$, and that computing witnesses for membership in L is hard, that the proof system for L is also a proof of knowledge and the family F has an efficient secret sharing scheme with completion. Then for any admissible parsing π, $F(\pi, L') \in SZKIP$.*

We remark that the extra assumptions on L in the corollary are only needed to ensure that we have a unconditionally hiding bit commitment scheme, by [53]. Any other assumption that would ensure this could be substituted, such as the existence of one-way permutations [90] or existence of collision intractable hash functions [91]. All our proofs are constructive and preserve efficiency in the sense that the new provers and verifiers we construct are polynomial time machines that call the prover and verifier of the original proof system for L as subroutines. Finally, it is interesting to note that our construction starts with a protocol that is only honest verifier zero-knowledge, and produces a protocol that is zero-knowledge in general. Thus our results contain parts of those in [53] as a special case.

We briefly indicate our technical approach. Regarding Theorem 3.9, suppose for a moment that L has an interactive proof (P_0, V_0) satisfying the properties stated in the theorem and that, additionally, this proof has the property that there exists a reply to at most one challenge, in case $x \notin L$. Note that Theorem 3.3 faces a situation similar to the current. By straightforward inspection of the proof of that theorem, especially the case of special soundness and honest verifier zero knowledge, it becomes clear that, with the extra condition on (P_0, V_0) from above, we could just carry out the construction from Theorem 3.3 to achieve our purposes here. However, in general we can not assume that (P_0, V_0) satisfies this "special soundness" property. In Section 2.4.2, we showed how we can transform (P_0, V_0), by means of interactive hashing, into an interactive proof (P_0^{IH}, V_0^{IH}) for the same language and with the same zero knowledge properties, such that "special soundness" is obtained. Using this transformation in addition, the techniques from Theorem 3.3 can be applied to obtain the target protocol (P, V) for the language $\mathcal{F}(\pi, L)$.

In the proof of Theorem 3.10, we reverse the roles of the prover and the verifier in the protocol from Theorem 3.9, to get the claimed result for the complement language L'. It is only here that we must allow the prover to be computationally unrestricted. The basic idea is as follows. Since the protocol (P, V) from above itself satisfies special soundness, we make the following observation. Let M be a simulator for (P, V). If $x \notin \mathcal{F}(\pi, L')$, then $M(x)$ may output an accepting conversation x, a, c, z, but the number of values c' different from c, such that x, a, c', z' is an accepting conversation for some z', is negligible. If $x \in \mathcal{F}(\pi, L')$, on the other hand, such z' exists for all c'. Thus, we let the verifier prepare a simulation with M on input x, and let the (unbounded) prover guess c.

3.4.4 The Closure Property for L

We now prove Theorem 3.9. Let a language L in $HV3AM/PZKIP$ or in $HV3AM/SZKIP$, a family $\mathcal{F} = \{f_n\}$ of monotone functions, an efficient secret sharing scheme $\mathcal{S}$ for $\mathcal{F}^*$ (see Section 3.2.3, and an admissible parsing π be given. We only need $\mathcal{S}$ to distribute secrets of one just bit. Without loss of generality, we may assume that the size of each of the shares $s_1, \ldots, s_n$ is equal to $l(n)$ bits, for some polynomial l. We let $s_{i,j}$ denote the j'th bit of the i'th share. Let x be a word parsed as $x_1, ..., x_n$, and let A denote the set of indices i such that $x_i \in L$. Its complement is denoted A'. Our purpose is to exhibit a ZKIP that $f_n(A) = 1$ holds. The description of the protocol (P_0^{IH}, V_0^{IH}) is found in Section 2.4.2.

Overview

We first give an informal description of the proof system (P, V) claimed by the theorem. Let's assume, for clarity, that $l(n) = 1$. The basic idea is that, for $x_i \in L$, the prover and the verifier engage in an execution of the preamble of (P_0^{IH}, V_0^{IH}), while for $x_i \notin L$, they execute (M_0^{IH}, V_0^{IH}). This is done in parallel for all x_i. In all n cases, two challenge values are isolated. In case $x_i \in L$, the prover can answer both of them, while if $x_i \notin L$, he can answer at most one. This means that for all $i \in A'$, the prover is effectively committed to a bit, which challenge (which v-value) he can answer.

The verifier now selects a random bit b_S. The prover must then (on his own) complete all the conversations of (P_0^{IH}, V_0^{IH}) that were started earlier and send the results to V. All these conversations must be accepting. But in addition, we interpret the v-value answered in the i'th instance as the bit s_i in an output of $\mathcal{S}$. This determines a set of shares $s_1, .., s_n$, and V will accept, only if this set of shares is consistent with the b_S he chose.

Intuitively, this is sound since if too many x_i's are not in L $(f_n(A) = 0)$, we have $f_n^*(A') = 1$. This means P will be committed to so many s_i's, that a secret is already uniquely determined before V chooses b_S, so P can only survive with probability $1/2$. It is zero-knowledge because it is easy to simulate if the 1-bit challenge from V is known in advance, thus standard techniques apply.

Proof of Theorem 3.9

Here follows a formal description of the protocol: Initially, both P and V parse the input x using π to obtain n and $x_1, ..., x_n$. Then the following steps are repeated k times, where k is the binary length of x:

1. First P generates a set of shares $\{s_i | \ i \in A'\}$ distributed according to $D_{A'}$. This can be done by choosing a random bit b_r, running $\mathcal{S}$ on input n, b_r and discarding all shares that do not correspond to indices in A'. Now for $i = 1$ to n do the following:

   If $i \in A$, P executes $l(n)$ copies of the preamble of (P_0^{IH}, V_0^{IH}) with V on input x_i.

   If $i \in A'$, then for $j = 1$ to $l(n)$, P and V run (M_0^{IH}, V_0^{IH}) on input input x_i. Here $s_{i,j}$ is P's private input. If any instance of M_0^{IH} produces output ?, then P sends ? and s_i to V. P stores the $l(n)$ answers $r_{i,1}, ..., r_{i,l(n)}$ produced privately.

2. V chooses a random bit b_S and sends it to P.

3. P completes the set of shares $\{s_i | \ i \in A'\}$ to a full set of shares consistent with b_S. For each $i \in A$ he completes the $l(n)$ preambles of (P_0^{IH}, V_0^{IH}) by using for the j'th copy the bit $s_{i,j}$ as the v-value, and using the algorithm of P_0^{IH} to compute a correct answer $r_{i,j}$.

   P now sends to V the complete set of answers $\{r_{i,j}\}$ and the set of shares s_i that were not already sent in step 1.

4. For each i where P did not send ? in step 1, V has now received $l(n)$ complete conversations of (P_0^{IH}, V_0^{IH}) with x_i as input. V checks that all these conversations would lead to acceptance by V_0^{IH}.

   V has also received a complete set of shares $\{s_i | \ i = 1, .., n\}$. He checks that this set is consistent with b_S.

   If all checks are OK, V accepts this iteration, otherwise he rejects and halts.

V accepts if and only if all k iterations are accepted. We now argue that this protocol has the required properties. *Completeness* follows trivially from the completeness of (P_0, V_0), the properties of the simulator M_0 and our assumptions on the sharing scheme $\mathcal{S}$. As to *Soundness*, there is nothing to prove, unless when parsed, the input word is of the form $x_1, ..., x_{n(k)}$ and $f_n(A) = 0$, where the index set A is defined as in the protocol description. Consider an arbitrary prover P^*.

Consider the situation after step 1 of the protocol. For each index $i \in A'$, either the prover has sent $?, s_i$ or for each $j = 1, ..., l(n)$ he can answer one of the two challenges resulting from each execution of (M_0^{IH}, V_0^{IH}). By Proposition 2.2, in the latter case, the challenge that can be answered (and hence $s_{i,j}$) is uniquely determined, except with probability $\delta(k_i)$, where k_i is the length of x_i. Therefore, after step 1, the set of shares $\{s_i | \, i \in A'\}$ that the verifier will accept in step 4 is uniquely determined because of either of the following two reasons. First, from the preambles or second, because the prover sent $?, s_i$ immediately. In the latter case, the set is uniquely determined except with probability at most $\epsilon(k) := \sum_i l(n)\delta(k_i)$. Since the parsing is admissible, all k_i's are polynomial in k, and hence $\epsilon(k)$ is negligible in k.

Now note that $f_n(A) = 0$ implies $f_n^*(A') = 1$, by the definition of dual functions, and that therefore any set of shares $\{s_i | \, i \in A'\}$ can be consistent with at most one value of the secret. Hence the probability that V accepts k iterations of steps 1-4 is at most $(1/2 + \epsilon(k))^k$, which is clearly negligible in k.

Finally, we argue that the protocol is *Zero-Knowledge*. We describe a simulator that works against any verifier V^*. Repeat the following k times:

1. Start by choosing a random bit b and run S on input n, b to get shares $s_1, ..., s_n$.
2. For $i = 1$ to n do the following:

   For $j = 1$ to $l(n)$, start (M_0^{IH}, V_0^{IH}) on input $x_i, s_{i,j}$. If any of the copies produce ? as output, send ? and s_i to V^*. Otherwise, let each copy execute the preamble of (P_0^{IH}, V_0^{IH}) with V^*. Save all answers $r_{i,j}$ produced.
3. Receive b_S from V^*.
4. If $b = b_S$, send to V^* all answers $r_{i,j}$, all shares s_i not yet sent, and stop simulation of this iteration.

   Otherwise, rewind V^* to its state at the start of step 1 and go back to step 1.

To show that this simulation works, consider first the case where the honest verifier simulation of (P_0, V_0) is perfect. Note that we are required to simulate correctly only in cases where the input is in $\mathcal{F}(\pi, L)$. This means that $f_n(A) = 1$ and hence $f_n^*(A') = 0$, using the notation from above. By Proposition 2.3, V^* receives no information about s_i where $i \in A$, and hence since $f_n^*(A') = 0$ receives also no information on b. Hence V^*'s choice of b_S is uncorrelated to b so that the probability that $b = b_S$ is $1/2$, and the complete simulation takes expected linear time.

For correctness of the output distribution, note that for $i \in A'$, the simulator follows completely the prover's algorithm. For $i \in A$, consider the simulation of a single preamble. The a sent initially has exactly the right distribution. In the interactive hashing, both simulator and prover answer consistently with an original choice c with uniform distribution, so all message sent there also have the right distribution. The event that $b = b_S$ is uncorrelated to all messages considered so far (by Proposition 2.3), so the fact that we wait until $b = b_S$ does not bias the distribution actually that is output. Finally, the answers $r_{i,j}$ produced by P resp. the simulator are in both cases random samples of what the original P_0 would produce, given the initial conversation defined in the preamble.

This finishes the perfect zero-knowledge case. In case of statistical zero knowledge, we use the same simulator, except that the oracle we have for making conversations of (P_0, V_0) (the honest verifier simulator) now generates output that is statistically close to the perfect case. Hence the output we produce will also be statistically close to perfect.

3.4.5 The Closure Property for L'

We now prove Theorem 3.10 and Corollary 3.8. The notation is the same as in the previous section. Note, however, that we now require an efficient secret sharing scheme with completion S for $\mathcal{F}$ (and not $\mathcal{F}^*$).

Overview

First the verifier V will (privately) choose a bit b_S and will share it using the secret sharing scheme to get shares $s_1, \ldots, s_n$. Then for each $i = 1 \ldots n, j = 1 \ldots l(n)$ the verifier V and the prover P will run (M_0^{IH}, V_0^{IH}) on input x_i, BUT this time the verifier will play the role of M_0^{IH}. V uses the bit $s_{i,j}$ as private input for the (i,j)'th instance. The verifier V is now able to answer correctly $v = s_{i,j}$. Then the prover must guess the bit b_S, and V rejects if the guess is incorrect.

Note that for i's where $x_i \in L$ there is a correct answer in (P_0, V_0) to any challenge value, so the execution of (M_0^{IH}, V_0^{IH}) tells P nothing about $s_{i,j}$, whereas if $x_i \notin L$, $s_{i,j}$ is most likely uniquely determined from the conversation. Hence, intuitively, the protocol is complete since if enough x_i's are in L' $(f_n(A') = 1)$, P can find enough s_i's to be able to compute b_S. It is sound since if too few x_i's are in L' $(f_n(A') = 0)$, P gets only negligible information about b_S. It is honest verifier ZK, since the honest verifier knows in advance what the prover's answer will be.

Proof of Theorem 3.10 and Corollary 3.8

Initially, both P and V parse the input x using π to obtain n and $x_1, ..., x_n$. Then the following steps are repeated k times:

1. V chooses a random bit b_S and runs S on input n, b_S to get shares $s_1, ..., s_n$.
2. For $i = 1$ to n do the following:

For $j = 1$ to $l(n)$, V and P run copies of (M_0^{IH}, V_0^{IH}) on input x_i (roles reversed!). V's private input is $s_{i,j}$. If any of the copies produce ? as output, send ? and s_i to P.

3. For $i = 1$ to n do the following:

   If $x_i \notin L$, P uses the messages received in the i, j'th preamble executed with x_i as input to determine $s_{i,j}$. Then P knows $\{s_i | \; i \in A'\}$. He uses these shares to determine a secret (bit) b and sends it to V. If the shares are inconsistent or could not be determined, P sends a random bit b.

4. V checks that $b = b_S$ and accepts this iteration if this is case, otherwise he rejects and halts.

V accepts, if and only if all k iterations are accepted. We now argue that this protocol has the required properties: Regarding *completeness*, consider an $i \in A$. Thus, $x_i \notin L$. For such an i, either s_i is sent to V directly or is with overwhelming probability uniquely determined from the preambles executed on x_i, by Proposition 2.2. Therefore P can compute the shares $\{s_i | \; i \in A'\}$ (although this may require unbounded computing power). We assume here that the input is in $\mathcal{F}(\pi, L')$, which means that $f_n(A') = 1$. Therefore b_S is uniquely determined from $\{s_i | \; i \in A'\}$ and hence P can answer correctly, except with negligible probability.

As to *soundness*, there is nothing to prove, unless the input word when parsed is of the form $x_1, ..., x_n$ and $f_n(A') = 0$. An arbitrary P^* may be able to compute the shares $\{s_i | \; i \in A'\}$, but these contain no information on b_S since $f_n(A') = 0$. By Proposition 2.3 the preambles executed on $x_i \in L$ give at most a negligible amount of information on the shares, and therefore P^* cannot guess any b_S with probability significantly different from 1/2. For k iterations the acceptance probability therefore becomes negligible in k.

Finally, *honest verifier zero knowledge* simulation is straightforward: simply follow the honest verifier's algorithm and use the value of b_S (which you know from the verifier's random tape) as the prover's answer b. This is statistically close to the real conversation as the simulation of all messages but the last one (the b) is perfect, and we have shown that the real prover's b-values are correct except with negligible probability.

To show Corollary 3.8, notice that the assumptions imply, by results in [53], that an unconditionally hiding bit commitment scheme exists. Using the method from [95] such a bit commitment scheme can be used to transform any honest verifier statistical zero-knowledge protocol into one that is statistical zero-knowledge in general.

3.4.6 Extensions

We have assumed for simplicity that only one basic language L is involved in our construction. But our method generalizes trivially to proving in statistical zero knowledge statements such as $x_1 \in L_1 \vee \ldots \vee x_n \in L_n$ where $L_i \in HV3AM/SZKIP$, put differently, our results still hold, even if each statement on a subword refers to a different language. It is also possible to mix languages and their complements. For example, the statement $x_1 \in L_1 \vee x_2 \notin L_2$ can be proved in statistical zero knowledge using a straightforward generalization of a corresponding method from [103].

3.5 Linear Secret Ballot Elections

3.5.1 Introduction

We present cryptographic protocols for multi-authority secret ballot elections that guarantee privacy, robustness, and universal verifiability. A special case of the theory developed in Section 3.2 combined with the verifiable secret sharing scheme of Pedersen, reduce the work required by the voter or an authority to a linear number of cryptographic operations in the population size (compared to quadratic in previous schemes). Thus we get significantly closer to a practical election scheme.

An secret ballot election scheme is viewed as a set of protocols that allow a collection of voters to cast their votes, while enabling a collection of authorities to collect the votes, compute the final tally, and communicate the final tally that is checked by talliers. In the cryptographic literature on voting schemes, three important requirements are identified:

Universal Verifiability ensures that any party, including a passive observer, can convince herself that the election is fair. This means that the published final tally is computed fairly from the ballots that were correctly cast.

Privacy ensures that an individual vote will be kept secret from any (reasonably sized) coalition of parties that does not include the voter herself.

Robustness ensures that the system can recover from the faulty behavior of any (reasonably sized) coalition of parties.

The main contribution in this section is to present an *efficient* voting scheme that satisfies universal verifiability, privacy and robustness. The efficiency of our schemes can be summarized as follows, when there are n authorities, m voters, and security parameter k. The total amount of communication will be $O(kmn)$ bits (posted to a "bulletin board"), while the required effort (in elementary operations) for any authority and any voter will be $O(km)$ and $O(kn)$ operations, respectively. For any threshold $t \leq n$, privacy will be assured against a coalition that includes at most $t-1$

authorities, and robustness against a coalition that includes at most $n - t$ authorities.

A fourth property recently stated (see [105, 67]) is that of prevention of *vote-duplication* (one voter copying the vote of another voter without knowing the actual vote). There are various efficient ways to incorporate this property in the schemes we present. We will include a straightforward solution.

3.5.2 Overview of the Approach

The parties in a voting scheme are modelled as probabilistic polynomial time processes. Two means of communication are typically assumed to be available for these parties:

A *bulletin board*, which is a broadcast channel with memory that can be observed and read by all parties. Each party controls her own section of the board in the sense that she can post messages exclusively to her own section, but not to the extent that she can erase or overwrite previously posted messages.

Private channels to support private communication between voters and authorities. For this task any secure public-key encryption scheme is suitable, possibly using the bulletin board to post the corresponding encryptions.

The parties of the voting scheme perform the following steps to execute an election. To cast a vote, each voter constructs a ballot as an encryption of the desired vote, and posts the ballot to the bulletin board. At this point, a proof of validity is also required that convinces all parties that the posted encryption contains a valid vote, without revealing it. The authorities, however, are able to decrypt the ballots (because of extra information received from the voter through a private channel). In the end, the final tally is published together with some auxiliary information to enable universal verifiability: any interested party (a tallier) may "accumulate" the encrypted votes and check the final tally, by holding it against this accumulation and the auxiliary information.

More technically, universal verifiability is achieved by requiring the encryption function to be suitably homomorphic. At the same time a different security property of the encryption function, which is similar to the binding property of commitment schemes, ensures that the authority (assume momentarily to be a single entity) cannot accumulate the individual votes in any other way than the voters actually voted. Such *homomorphic encryption schemes* are available under a wide variety of common cryptographic assumptions.

Central to our results is the way we achieve an efficient proof of validity for ballots. The proof of validity shows that a ballot actually represents a vote, which means that it either represents a yes or a no, and nothing else. To maintain privacy for the voters, the general idea is to use some sort of zero-knowledge proof. The problem is however that zero-knowledge proofs usually require a large number of repetitions before the desired level of confidence is achieved. The efficiency of these proofs to a great extent influences the efficiency of the whole scheme, both in terms of computational effort and in terms of the required bandwidth for each voter.

Our contribution now is twofold. We use a particular efficient homomorphic encryption scheme, based on the discrete logarithm problem (although, as we show, it can be based on other cryptographic assumptions as well). Moreover, by applying results from Section 3.2, the proof of validity is a simple three-move protocol which is *witness indistinguishable* (in fact, witness hiding as well), instead of zero-knowledge as in previous schemes. This leads to a significant reduction of the effort required by the voter, from quadratic in the security parameter to linear, while still hiding the vote.

Clearly, in the scenario above, the authority learns individual votes. This situation is alleviated by having multiple authorities instead of one. The encrypted vote is distributed over the authorities such that fewer than some number of authorities remain ignorant about individual votes. To make sure that the authorities are convinced that the posted shares actually represent the vote cast, verifiable secret sharing is employed. Here, we apply Pedersen's scheme [96], as it fits with the other primitives remarkably well. It is also by this method that robustness is achieved in the sense that only a subset (of a size larger than a certain threshold) of the authorities is required to participate throughout the execution of the scheme in order to compute the final tally.

3.5.3 Related Work

The type of voting schemes considered here was first introduced and implemented in [40, 19, 17]. In these schemes, privacy and robustness are achieved by distributing the ballots over a number of tallying authorities, while still achieving universal verifiability. This contrasts with other approaches in which the ballots are submitted anonymously to guarantee privacy for the individual voters. Such schemes rely on the use of anonymous channels [27], or even some form of blind signatures as in privacy-protecting payment systems (see, for instance, [38]) to achieve privacy. For these approaches it seems difficult however to attain all desired properties, and still achieve high performance and provable security.

The voting schemes of [40, 19, 17] rely on an r-th residuosity assumption. In [104] it is shown that such schemes can also be based on a discrete logarithm assumption (without fully addressing robustness, though), and how this leads to considerable efficiency improvements. In this section we will also address various number-theoretic assumptions.

As with our result, the efficiency improvement in [104] is mainly due to an improved zero-knowledge protocol to show validity of ballots. As noted by Benaloh, such *cryptographic capsules* [16, 17] are at the heart of the problem of electronic elections and also useful for other applications.

3.5.4 Cryptographic Primitives

We use the discrete logarithm problem (see Section 2.3.2). Each family $\mathcal{G}$ for which it is reasonable to assume the intractability of the discrete logarithm problem is suitable for our purpose of constructing efficient and secure homomorphic encryption schemes with corresponding proofs of validity. A well-known family, however, is obtained by choosing large primes p and q at random such that $q \mid p-1$; then G is the unique subgroup of order q in $\mathbb{Z}_p^*$. The discrete logarithm problem for elliptic curves over finite fields is also a candidate for implementation.

Homomorphic Encryption with Efficient Proof of Validity

Let $\mathcal{G}$ be a family of groups of prime order, and generator Gen as in Section 2.3.2. Assume that the discrete logarithm problem for $\mathcal{G}$ is intractable. The encryption scheme below is obtained as an extension of Pedersen's commitment scheme [96] with an efficient proof of validity.

Initialization: The participants, or a designated subset of them, run the generator $Gen(1^k)$ and obtain a description of a group G_q of prime order q, and random group elements g and h. One way to do this honestly, is that the participants agree on a program for Gen first. Then they each run separately $Gen(1^k)$ where the coinflips needed are selected mutually at random, either by observing a common source of physical randomness, or by executing some well-known cryptographic protocol suitable for this purpose.

Encryption: A participant encrypts $v \in \mathbb{Z}_q$ by choosing $\alpha \in \mathbb{Z}_q$ at random, and computing

$$B \leftarrow g^\alpha h^v.$$

Opening: A participant can later open B by revealing both v and α. A verifying party then checks whether $B = g^\alpha h^v$, and accepts v as the encrypted value.

Homomorphic Property: Encryption is homomorphic in the sense that, if B_1 and B_2 are encryptions of v_1 and v_2, respectively, then B_1B_2 is an encryption of $v_1 + v_2 \bmod q$. In general, an encryption of any linear combination of v_1 and v_2 can be obtained from B_1 and B_2; in

Voter			Verifier
$v = 1$	$v = -1$		
$\alpha, r_1, d_1, w_2 \leftarrow \mathbb{Z}_q$	$\alpha, r_2, d_2, w_1 \leftarrow \mathbb{Z}_q$		
$B \leftarrow g^\alpha h$	$B \leftarrow g^\alpha / h$		
$a_1 \leftarrow g^{r_1}(Bh)^{-d_1}$	$a_1 \leftarrow g^{w_1}$		
$a_2 \leftarrow g^{w_2}$	$a_2 \leftarrow g^{r_2}(B/h)^{-d_2}$		
		$\xrightarrow{B, a_1, a_2}$	
			$c \leftarrow \mathbb{Z}_q$
		$\xleftarrow{c}$	
$d_2 \leftarrow c - d_1$	$d_1 \leftarrow c - d_2$		
$r_2 \leftarrow w_2 + \alpha d_2$	$r_1 \leftarrow w_1 + \alpha d_1$		
		$\xrightarrow{d_1, d_2, r_1, r_2}$	
			$d_1 + d_2 \overset{?}{=} c$
			$g^{r_1} \overset{?}{=} a_1(Bh)^{d_1}$
			$g^{r_2} \overset{?}{=} a_2(B/h)^{d_2}$

FIGURE 3.2. Encryption and Proof of Validity of Ballot B

particular, the sign of v_1 can be flipped by computing B_1^{-1}. Note also that $B_1 h$ encrypts $v_1 + 1 \bmod q$.

Proof of Validity: In our voting scheme to follow, it will be the case that a voter posts an encryption of a value $v \in \{1, -1\}$. To demonstrate that the encrypted value is indeed in $\{1, -1\}$, without revealing it, the voter and the verifier execute the efficient *proof of validity* of Figure 3.2. This proof requires only a very small number of modular exponentiations.

THEOREM 3.11 *Under the discrete logarithm assumption, the encryption scheme is binding in the sense that it is infeasible to open a given encryption in two different ways. Furthermore, the proof of validity is a convincing argument that a given encryption is indeed an encryption of a value from the set $\{1, -1\}$, thereby not releasing any information about the actual value.*

PROOF. If any party is able to open an encryption B in two different ways, , which means to present values α, v, α', v' such that $B = g^\alpha h^v = g^{\alpha'} h^{v'}$ with $\alpha \neq \alpha'$ and $v \neq v'$, it follows that $\log_g h = (\alpha - \alpha')/(v' - v)$, which contradicts the discrete logarithm assumption. Furthermore, the protocol in Figure 3.2 is a witness indistinguishable proof of knowledge that the voter knows $\log_g Bh$ or $\log_g B/h$, by Corollary 3.5; here, Schnorr's identification protocol [106] is used as the basic protocol in the construction. Thus the verifier learns that the voter knows α and $v \in \{1, -1\}$ such that $B = g^\alpha h^v$, without obtaining any information about the actual value of v. □

Note that it even follows that the proof of validity is witness hiding. Jumping ahead a little bit, envision that a voter posts an encryption of his vote and that all other participants must verify its validity. As depicted in Figure 3.2, a source of randomness is required in the program for the verifier. For this purpose, one can use some unpredictable physical source of randomness [40], or agree on mutually random bits by cryptographic means. A more practical way, however, making the protocol *non-interactive*, is to apply the well-known Fiat-Shamir heuristic [63]. Their idea is to compute the challenge in a three-move identification scheme as a hash value of the message to be signed and the first message of the prover. So, let $\mathcal{H}$ be a suitable strong cryptographic hash function (thought of as a random oracle). In the non-interactive version of our proof of validity, the challenge c is computed as $c = \mathcal{H}(B, a_1, a_2)$. The set of values d_1, d_2, r_1 and r_2 is denoted by proof(B). Given the values in proof(B), any participant can check the validity of B by verifying that $d_1 + d_2 = \mathcal{H}(B, g^{r_1}(Bh)^{-d_1}, g^{r_2}(B/h)^{-d_2})$.

Verifiable Secret Sharing

To achieve robustness efficiently, non-interactive verifiable secret sharing is employed. We use the scheme of Pedersen [96], as it is based on discrete logarithms as well and fits nicely with our encryption scheme. Being information theoretically secure, this scheme also contributes to privacy in our multiple authority scenario.

3.5.5 Secret Ballot Election Scheme

We now present our main result, a secret ballot election scheme satisfying privacy, universal verifiability and robustness. The participants in the election scheme are n authorities $A_1, \ldots, A_n$ and m voters $V_1, \ldots, V_m$. Privacy and robustness are as follows. No collusion of fewer than t authorities can reveal an individual vote, while the election will be successful when at least t authorities operate properly ($1 \le t \le n$). At the same time we incorporate a simple mechanism to postpone the decision on what to vote until the preparation of the election has been completed. In this way several elections can be prepared beforehand at the beginning of the year, say, and casting a vote in an election then boils down to publishing essentially one bit of information.

Informally the scheme works as follows. Each voter V_i prepares a vote by randomly selecting a number b_i in $\{1, -1\}$. The voter first encrypts b_i by computing $B_i = g^{\alpha_i} h^{b_i}$, where $\alpha_i \in \mathbb{Z}_q$ is chosen randomly, and posts B_i to the bulletin board. Subsequently, b_i is considered as a secret which is to be shared among the authorities. We employ a verifiable secret sharing scheme to prevent voters from disrupting elections by sending false shares to authorities. The efficient scheme by Pedersen [96] is a perfect candidate as it applies exactly to the discrete log setting we are considering. The idea is thus to let the voter act as the dealer in Pedersen's scheme, sending a verifiable share of the secret b_i to each authority using the proper private channels. The voter also posts proof(B_i) to the bulletin board to prove that B_i indeed encrypts a value in $\{1, -1\}$. Later, voter V_i may then cast a vote $v_i \in \{1, -1\}$ by publishing the value $s_i = b_i v_i$. In the end, the aggregate value $T = \sum_{i=1}^{m} v_i$ reduced modulo q such that $-q/2 < T < q/2$ represents the total number of yes-votes minus the total number of no-votes, hence the total number of yes-votes is $(m+T)/2$. For these numbers to be correct the obvious requirement is that $m < q/2$.

We assume that the group G_q and the members g and h are generated as described in Section 3.5.4. In particular, it then follows that $\log_g h$ is not known to any participant.

Ballot Construction

Each voter V_i prepares a masked vote $b_i \in \{1, -1\}$ in the following way.

1. The voter V_i chooses b_i randomly from $\{1, -1\}$, and computes the ballot $B_i = g^{\alpha_i} h^{b_i}$, where α_i is randomly chosen from $\mathbb{Z}_q$. The voter also computes proof(B_i). Finally, the voter determines polynomials G_i and H_i,
$$G_i(x) = \alpha_i + \alpha_{i1} x + \cdots + \alpha_{i,t-1} x^{t-1}$$
$$H_i(x) = b_i + \beta_{i1} x + \cdots + \beta_{i,t-1} x^{t-1},$$
where the coefficients α_{il}, β_{il}, $1 \le l < t$, are chosen at random from $\mathbb{Z}_q$. Also, for these coefficients the voter computes the commitments $B_{il} = g^{\alpha_{il}} h^{\beta_{il}}$.
2. The voter V_i posts $B_i, \text{proof}(B_i), B_{i1}, \ldots, B_{i,t-1}$ to the bulletin board.
3. All participants verify whether ballot B_i is correctly formed by checking proof(B_i).[16]
4. The voter V_i sends the shares $(a_{ij}, b_{ij}) = (G_i(j), H_i(j))$ to authority A_j, using a private channel.

[16]To prevent vote duplication, a bit string specific to voter V_i is also included in the input to the hash function $\mathcal{H}$ in proof(B_i). In case the proof of validity is done interactively, as depicted in Figure 3.2, this bit string could be incorporated in the challenge.

5. Each authority checks the received share (a_{ij}, b_{ij}) by verifying that
$$g^{a_{ij}} h^{b_{ij}} = B_i \prod_{l=1}^{t-1} B_{il}^{j^l}.$$

Vote Casting

To cast a vote, V_i simply posts $s_i \in \{1, -1\}$ such that $v_i = b_i s_i$ represents the desired vote.

Tallying

1. Each authority A_j posts the sum $S_j = \sum_{i=1}^{m} a_{ij} s_i$ and the subtally $T_j = \sum_{i=1}^{m} b_{ij} s_i$.
2. Each tallier checks the share (S_j, T_j) posted by authority A_j by verifying that
$$g^{S_j} h^{T_j} = \prod_{i=1}^{m} \left(B_i \prod_{l=1}^{t-1} B_{il}^{j^l} \right)^{s_i}.$$
3. From t pairs (j, T_j) that correspond to authorities for which the shares (S_j, T_j) are correct, each tallier can compute the final tally T from the formula:
$$T = \sum_{j \in A} T_j \prod_{l \in A \setminus \{j\}} \frac{l}{l-j},$$
where A denotes a set of t correct authorities.

We assume (w.l.o.g.) that in a successful election the shares of every voter have been accepted by all authorities. That is, all verifications by the authorities in the last step of the ballot construction are successful. In case an authority receives a share that does not pass this step, the authority may post the share so that anybody can verify that the share is not correct and that it corresponds to the posted encryption of step 4 of the ballot construction.

THEOREM 3.12 *Under the discrete logarithm assumption, our secret-ballot election scheme satisfies universal verifiability, robustness and privacy.*

PROOF. To prove universal verifiability first note that only correct ballots are counted on account of Theorem 3.11. Further, to prove that the final tally is correct, we reason as follows for each correct authority A_j. Let $G(x) = \sum_{i=1}^{m} s_i G_i(x)$ and $H(x) = \sum_{i=1}^{m} s_i H_i(x)$. By the binding property[17] of the encryptions B_i (see Section 3.5.4), we have:

$$\begin{aligned} g^{G(j)}h^{H(j)} &= g^{\sum_{i=1}^{m} s_i G_i(j)} h^{\sum_{i=1}^{m} s_i H_i(j)} \\ &= g^{\sum_{i=1}^{m} s_i a_{ij}} h^{\sum_{i=1}^{m} s_i b_{ij}} \\ &= \prod_{i=1}^{m} (g^{a_{ij}} h^{b_{ij}})^{s_i} \\ &= \prod_{i=1}^{m} (B_i \prod_{l=1}^{t-1} B_{il}^{j^l})^{s_i}. \end{aligned}$$

By the assumption that the verification in step 2 of the tallying protocol holds for (S_j, T_j), we thus have $g^{G(j)}h^{H(j)} = g^{S_j}h^{T_j}$, which implies $S_j = G(j)$ and $T_j = H(j)$ under the discrete logarithm assumption. As a consequence, the final tally T is indeed equal to $H(0)$ and thus represents the result of the election if the verification in step 2 of the tallying holds for at least t authorities. This deals with universal verifiability and robustness. The privacy property can easily be proved from the fact that the secret sharing scheme and the proofs of validity (see Section 3.5.4) are information-theoretically secure. $\square$

Thus, fewer than t authorities do not obtain any information about individual votes, other than what can be derived from the final tally (and accounting for votes that have been revealed by individual voters). To summarize the *performance* of our scheme, we note the following. The hard operations are the modular exponentiations with full exponents (exponents of expected size $|q|$). Counting these operations, we see that the work for each voter is $O(1)$ for the construction of the ballot (including the proof of validity) plus $O(t)$ for the commitments for the verifiable secret sharing scheme. Each authority has to do $O(m)$ work to check all the shares. So active participants do linear work. Finally, verification of the election (done only by interested talliers) requires $O(m)$ work to check the ballots plus $O(t \log^2 t)$ multiplications to check the shares of the final tally. Note that the parties' work can all be done either before or after the casting of the votes. The actual election simply consists of every voter posting essentially one bit of information to the bulletin board. This represents essentially an improvement of two orders of magnitude over the best schemes known so far (assuming $k = 100$), and enables much of the work required to be done off-line.

[17]The description of our scheme above assumes that the hash function in the Fiat-Shamir style proof(B_i), behaves like a random-oracle. Instead of using this technique, the participants can get the necessary random bits as explained in Section 3.5.4, thus also removing the need for this extra assumption.

3.5.6 Conclusion

We have presented election schemes of provable security and practical efficiency. The computational and communication complexity are essentially linear, instead of quadratic as for previous schemes. Even with a large number of authorities like $n = 10$, a voter need not communicate more than approximately 10.000 bits of information to submit a vote, assuming that $|p| = 512$ bits and $|q| = 160$ bits for our discrete log scheme. (Note that the actual ballot plus its proof of validity require only 1152 bits.) Compare this, for instance, to [104] with in the order of a million bits per vote for the same level of security.

A property of voting schemes, recently identified, that we do not consider here is that of *non-coercibility* [18]. Non-coercibility ensures that no voter will obtain, as a result of an execution of the voting scheme, a receipt that can prove how she voted. The intention of this property is to prevent vote-buying and other tactics of voter persuasion. It is possible that extensions of our ideas can give a scheme which incorporate non-coercibility while maintaining universal verifiability, privacy, robustness, and efficiency.

3.6 Efficiently Combatting Man-in-the-Middle Attacks

3.6.1 Introduction

A (public key) identification scheme (see for instance [63]) is an (interactive) protocol by means of which one party (the prover) proves its identity to another party (the verifier). Securing log-in procedures is a main application of such schemes. An identification scheme consists of an algorithm to generate public-key/private-key pairs, and a protocol for the prover and the verifier. The collection of eligible key-pairs is chosen such that it is infeasible to compute a corresponding private key when only the public key is observed. Typically, the protocol's purpose is to show that the prover "knows" the private key that corresponds to the prover's public key. Most known identification schemes take the form of three move interactive where the verifier is required to send a random bitstring as a challenge. For such methods to be secure, the verifier must not be able to extract this private key from the prover. Formally, this notion of security is captured by considering *adaptive impersonation attacks.* The (probabilistic polynomial time) attacker is given a prover, who has access to a key-pair as produced by the key-generation algorithm, as a black-box. Thus, the attacker only sees the prover's outputs as dictated by the identification protocol and not any of its internal coinflips, private inputs, etc. Next, the attacker is allowed to query the black-box a polynomial number of times, playing the role of a (malicious) verifier. This means that the attacker is allowed to choose the challenges in any way thought suitable to extract information about the private key. In particular, the choice of any next challenge may depend on the entire history of the attack and public key. Next, the attacker is denied any further access to this black-box prover. The identification scheme is called secure against adaptive impersonation attacks if the attacker is still unable to impersonate the prover (execute the prover's part of the protocol, facing an honest verifier).

In [11] a weakness of identification schemes proposed until then was exposed. There, the authors explained how a malicious *man-in-the-middle* $\tilde{V}$ may abuse his conversations with an honest prover P to misrepresent himself as P to yet another verifier V. The attack is not by cryptographic ingenuity. But, simply pretending to be a verifier himself, $\tilde{V}$ actually forwards V's challenges to P and forwards P's replies to V. Thus, while P is under the impression that he is identifying himself to $\tilde{V}$, he is actually identifying himself to V, to the advantage of $\tilde{V}$. A remedy suggested in [11] has the prover and verifier (rather the devices that represent them) isolate themselves physically from the outside world. A Faraday's cage could be a suitable implementation. However, for identification over networks, for instance, this measure seems not to be useful.

We present a simple method to construct identification schemes resilient against adaptive impersonation *and* man-in-the-middle attacks. Though zero-knowledge or witness hiding protocols are known to withstand attacks of the first kind, all such protocols previously proposed suffer from the weakness observed by Bengio et al. [11], since a malicious verifier may simply act as a moderator between the prover and yet another verifier, thus enabling the malicious verifier to pass as the prover. Using a collision intractable (without knowing the private key, it is infeasible to pass the protocol), honest verifier zero knowledge three-move public coin protocol, we build a witness-hiding identification scheme that differs from previous proposals in that an execution of a given proof of identity can only be unambiguously appreciated by the intended verifier. This is achieved by having the prover direct the protocol to the intended verifier's public key. It is subsequently shown that resilience against man-in-the-middle-attacks follows from this approach. Note that required primitive protocol corresponds to an identification scheme secure against passive impersonation and honest verifiers. Directing a proof to an intended verifier has been considered by other researchers in a different context, as we will explain later. Our contribution is to provide a general, secure and efficient immunization against adaptive man-in-the-middle impersonation attacks in identification schemes. Furthermore, we want the immunization to work even if the original identification scheme satisfies only weak security properties.

As an example, Schnorr's scheme based on discrete logarithms [106] or Guillou-Quisquater's scheme based on RSA [76] can be immunized by our construction. But more generally, any one-way group homomorphism or any pair of claw-free trapdoor permutations gives rise to the desired building block. If we would take, for example, Schnorr's scheme [106] as input to our constructions, the resulting identification scheme would have twice the complexity (in terms of computation and communication) of [106]. But we are then able to prove that our scheme is witness-hiding and resilient against man-in-the-middle attacks if computing discrete logarithms is hard.

Conceptually, our method to disable man-in-the-middle attacks is as follows. Let X and Y be two players, where X wishes to identify himself to Y. Suppose now that we have an efficient method by which X could take Y's public key, and his own key-pair (his public key and secret key), and securely prove the statement "I know X's secret key or I know Y's secret key." If this protocol is *witness indistinguishable* (no information is released as to which is the case), only Y can be sure he is talking to X rather than anyone else. For, any other verifier Z would only know that he is talking to X *or* Y. Thus, if X directs his proof to Y as outlined above, the proof is unambiguous only to Y.

So why would this help against man-in-the-middle attacks? By the symmetry of the statement proved and by the asserted witness-indistinguishability of the proof, if Y could abuse his conversation with X to pass as X at Z as the man-in-the-middle would do, he must be able to do so without talking to X. Thus the man-in-the-middle attack reduces to a cryptographic attack. But now we invoke the witness-indistinguishability again to show that if Y's attack would succeed, he could compute X's secret key. This then contradicts our assumption that it is hard to compute the secret key from a random public key. We stress that this approach makes sense only if the keys are sufficiently independently generated. In the extreme case that two verifier keys are identical, it is clear that man-in-the-middle attacks are still feasible. More generally, a proof of security will fail if there is dependence among these keys: if one is chosen as a clever function of the other (such as a random and secret power of a given key based on discrete logarithms), proof given to one verifier may still be "diverted" to another verifier. In our application from Section 3.6.7, dependence of keys is enforced in a natural way.

We note that the same basic idea of proving one of two statements in order to direct a proof to one specific verifier was found independently by Jacobson, Impagliazzo and Sako in [82]. Their main motivation was to make undeniable signature schemes more secure and non-interactive. Their method for building a verifier designated protocol uses a trapdoor bit commitment scheme. In comparison, our method shows that if you start with a protocol of a certain form, then a separate trapdoor bit commitment is not needed. On the other hand, their methods works for some protocols that are not of the form we consider.

It is not so much the concept explained above that we advocate as a significant contribution here. We would like to stress that the concept has been applied implicitly before, prior to [82]. [82] is the first paper applying the ideas to verifier-directed proofs, however. We know of at least one example, namely the protocol of Feige and Shamir [65] for bounded round general zero knowledge proofs. There, the prover commits to a witness for the NP-statement to be proved using an unconditionally hiding *trapdoor commitment scheme*, an instance of which is generated by the verifier. Indeed, the proof conducted there can be seen as showing that the NP-statement is true, or that the prover knows the verifier's trapdoor! To get the designated verifier proofs for general languages, postulated in [82] but not given, we can use the result of [65] and make sure that the verifiers' instances of the trapdoor commitment scheme are independently generated.

In our setting, we restrict ourselves to the problem of identification, and attempt to formulate a very efficient solution to the problem of identification in the presence of an *adaptive man-in-the-middle attacker*. Moreover, we are only interested in solutions that allow for some well-defined and accepted cryptographic intractability assumption to be reduced to the security of the identification scheme.

It is interesting to note that our results apply to a general class of identification schemes which in their normal mode of operation need only satisfy seemingly weak security properties. Namely, zero knowledge with respect to the honest verifier and collision intractability (that is, the scheme is secure against passive impersonation attacks). As a result of our simple and efficient transformation, we obtain the required security level.

Technically speaking, our approach is close to the ones taken in [47, 43]. However, it is not clear from those papers (which may partly be seen as investigations into witness hiding) how we can efficiently obtain security against adaptive *man-in-the middle attackers* in our context. Please note that such was neither clear from [82], since there the focus is on undeniable signatures. Although it appears to be true that their approach using trapdoor bit commitments has a wider applicability than that, their approach does not indicate that immunization of an identification scheme against man-in-the-middle attackers, can be done efficiently and securely *even if* the given scheme is only weakly secure in normal mode of operation, as we discussed above.

Please note that, for example, digital signatures secure against chosen message attacks also lead to identification schemes secure against impersonation and man-in-the middle attacks. The prover would simply sign a message consisting of a random challenge, supplied by the verifier, and the verifier's public key. Although we feel that our schemes could compare favorably in terms of practical value to even such solutions, we like to point out that we aim for a practical identification scheme that is proven secure if some standard cryptographic intractability assumption holds. Seen in this light, digital signatures, for example, with such proven security still come at too high a price in this context. Nevertheless, it may be reasonable here to use them for key-certification. If one is concerned with practical value *and* proven security (relative to a plausible assumption), it may be true that our proposal for identification schemes secure against impersonation and man-in-the-middle attacks comes close to what one could reasonably achieve in this area, due to its conceptual simplicity and efficient implementation.

3.6.2 Relation to Identification Schemes

Let (A, B) be a Σ-protocol for relation R and generator G. Suppose that it is collision-intractable (over R and G) and that it is honest verifier zero knowledge. It is easy to see that such a protocol constitutes an identification scheme *secure against passive attacks*, if (A, B) is collision-intractable and if the length t_B of the challenges is large enough, say linear in the security parameter. A public/private key-pair is $(x, w) \in R$, as generated by G. By Proposition 2.1, we can efficiently extract collisions from a successful passive attacker (that is, one which is given the public x only). But this contradicts our assumptions on (A, B).

Adding honest verifier zero knowledge to our requirements, makes sure that the resulting scheme is *secure against random challenge attacks.* By this we mean that even an attacker which is allowed to query a prover on *random* challenges, cannot *later* pose as that prover (note that we use the previous observation here that collision-intractability implies security against passive attacks.

Security against adaptive attacks means that even though the attacker is allowed to query a prover on *any* challenge of his choice, it can still not later pose as the verifier. The adaptive man-in-the-middle attacker, is one which also has "adaptive access" to a prover X. But this time, the attacker pretends to be verifier Y_1. If it's still infeasible for this attacker to make another (non-malicious) verifier Y_2 accept X's identification, we say that the identification scheme is *secure against adaptive man-in-the-middle attacks.*

3.6.3 Main Result

Let (A, B) be a collision-intractable Σ-protocol for relation R and generator G. Suppose that (A, B) is honest verifier zero-knowledge, with simulator M, and that the challenge length t_B is linear in the security parameter k. Thus, by the remarks above, (A, B) constitutes an identification scheme secure against *random challenge attacks.* Our purpose is to transform (A, B) into a new identification scheme which is secure against adaptive man-in-the middle attacks. This transformation works as follows.

Key Generation

A keypair $(x, w) \in R$, consisting of a public key x and a secret key w, for participant X is generated as

$$(x, w) \leftarrow G(1^k)$$

for an appropriate security parameter k. The public key x is placed in X's public directory. The secret key w is held privately.

Identification of X to Y

Here, participant X will identify itself to participant Y. Let their respective public keys be x and y, and let X's secret key be w. The claimed identification protocol runs as follows.

Move 1: X computes $a \leftarrow a(x, w)$ and $(y, b, d, s) \leftarrow M(y)$. Then X sends the pair (a, b) to Y.

Move 2: Y selects $C \leftarrow \{0,1\}^{t_B}$ and sends C as a challenge to X.

Move 3: X puts $c \leftarrow C \oplus d$ and computes $z \leftarrow z(x, w, a, c)$. Then X sends z, d, s to Y. Finally, Y checks the conversation by verifying whether $\phi(x, a, C \oplus d, z) \overset{?}{=} \textit{accept}$ and $\phi(y, b, d, s) \overset{?}{=} \textit{accept}$. If these verifications are satisfied, X is accepted by Y.

3.6.4 Implementation

Clearly, Section 4.5.5 provides ways for implementing the required primitive for our results in this section, that is, collision-intractable, honest verifier zero knowledge protocol (A, B) For efficiency, *claw-free pairs of trapdoor permutations* (see Section 2.5.3) or *special one-way group isomorphisms* (See Section 2.5.2) are most eligible as a basis for the protocol (A, B).

3.6.5 Security Analysis

We give proof of security under the assumption that the participants' keys are generated as prescribed in the Key Generation protocol. In the next section we give an example application where this condition is satisfied in a natural way. If we abandon the assumption on the key generation, we would have to, for instance, assume a trusted center to which all participants have to demonstrate knowledge of their secret key first, before their public key can be registered.

THEOREM 3.13 *Let (A, B) be a collision-intractable Σ-protocol for relation R and generator G. Suppose that (A, B) is honest verifier zero-knowledge and that the challenge length t_B is linear in the security parameter k.*

Then the identification scheme based on (A, B) from Section 3.6.3 is secure against impersonation and man-in-the-middle attacks.

PROOF. The idea is as follows. First we generate public key x' according to G, and discard the corresponding secret key. We show that, if the protocol were not witness hiding or were not resilient against man-in-the-middle attacks, there exists an efficient algorithm that takes x' as input and outputs a collision for x' in the protocol (A, B). But this would then contradict (A, B)'s collision-intractability.

The following game is easily to be seen as modelling the situation. Let m be polynomial in the security parameter k. We generate m public keys with known secret keys by running G m times. We flip a coin b. If $b = 0$, then we put $x \leftarrow x'$ and assign the m key pairs to $Y_1 \dots Y_m$. If $b = 1$, we select j at random from $\{1, \dots, m\}$, and put $y_j \leftarrow x'$, and assign the m key pairs to $X, Y_1, \dots, Y_{j-1}, Y_{j+1}, \dots Y_m$.

The game consists of two stages.

1. The attacker gets the following prover as a black-box. We define P as the prover who gets x and all public keys y_i as input, plus the secret keys as generated above. P can perform the identification protocol for all pairs (x, y_i). The attacker is allowed to play with P (as a black-box, but not rewindable) for a polynomial amount of time. Then, the attacker gives us a number $j' \in \{1, \dots, n\}$, and hands back P. This models the idea that before the real attack, the attacker may try to extract as much information as needed for winning in the second stage.

2. With probability $(m + 1)/(2m)$, the attacker chose $j' = j$ such that P was not given the secret key for y_j in the beginning or was not given the secret key for x. Let's assume that this event happens (If not, we re-run the previous stage). Next, the attacker gets as input the secret keys for all public keys y_i with $i \neq j$. This models the idea that (possibly via a man-in-the-middle attack), the attacker tries to pass as X to any other verifier intended by X. To make the proof easier, we just give the attacker the secret keys which allow him to perfectly simulate X's behaviour at any other site than Y_j, rather than giving him X as a black-box: if he can't do it *with* the secret keys, than he certainly can't when he is given X as a black-box who only identifies himself at Y_i with $i \neq j$. The attacker wins the game, if he can pass the protocol against the honest verifier on input (x, y_j).

Let's assume that the attacker won with probability $\epsilon > 2^{-t_B}$ (recall that t_B is assumed to be of linear size in k). Then, by Proposition 2.1, we can extract a collision for y_j or for x from the attacker (running it as a rewindable black-box) with expected time polynomial in the running time of the attacker and $1/(\epsilon - 1/2^{t_B})$. Thus, if ϵ is non-negligible, then we can extract a collision from the attacker in expected polynomial time. But, this is a collision for key x' with probability 1/2, since the attacker cannot distinguish between the cases $b = 0$ and $b = 1$ by witness indistinguishability of the protocol (which follows by the properties of the simulator M). This contradicts the assumption that (A, B) is collision-intractable over G. □

3.6.6 *Example*

As an example, we give an implementation based on discrete logarithms. As in Section 2.3.2, let G_q be a group of prime order q such that computing discrete logarithms in G_q is hard. Let g denote a fixed element from $G_q \backslash \{1\}$.

Key Generation

A keypair, consisting of a public key and a secret key, for participant X is generated as

$$(x = g^w, w)$$

where w is chosen at random from $\mathbb{Z}_q$. The public key x is placed in X's public directory. The secret key w is held privately.

Identification of X to Y

Here, participant X will identify itself to participant Y. Let their respective public keys be x and y, and let X's secret key be w. The claimed identification protocol runs as follows.

Move 1: X computes $a = g^z$ and $b = g^s y^{-d}$, where z, s and d are chosen at random from Z_q. Then X sends the pair (a, b) to Y.

Move 2: Y selects C at random from Z_q and sends C as a challenge to X.

Move 3: X puts $c \leftarrow C+d \bmod q$ and computes $r \leftarrow cw+z \bmod q$. Then X sends r, d, s to Y. Finally, Y checks whether $g^r \overset{?}{=} ax^c$ and $g^s \overset{?}{=} by^d$, where c is defined as $C + d \bmod q$ If these verifications are satisfied, X is accepted by Y.

3.6.7 *Applications*

Imagine an organization with n sites to which restricted access is applicable. Some m officials are granted access to some of these sites. When an accessor presents himself at one of these sites, his access rights are checked by verifying his identity. These sites may vary from buildings, to specific sections of buildings, or even to databases or computer systems. The organization keeps a central list of the identities of the officials and their specific access rights. It is assumed that each site has access to this list, either by having a copy of the list at hand, or by consulting the central database.

Let $X_1, \ldots, X_n$ be the collection of participants. The collection of sites with restricted access is denoted $Y_1, \ldots, Y_m$. The organization generates a keyset (x_i, w_i) for each participant X_i, as described in the Key Generation protocol in Section 3.6.3. Each participant X_i is given a tamperresistant smartcard S_i, capable of performing our protocols. The keyset is securely loaded into the cards. Now, for each site Y_j, the organization generates a keyset (y_j, v_j). The secret key v_j is destroyed. We assume that each site is represented too by some device capable of performing the protocols. For each site, the organization prepares a list of the public keys of the officials that are granted access to this site. This list is made available to the site. Please note that the devices for the sites need not store any secret information. One only has to make sure that the data they store is authentic and cannot be modified by unauthorized parties.

When participant X_i wishes to exercise his right of access to site Y_j, he lets his smartcard simply perform the identification protocol with site Y_j as the verifier, on common input (x_i, y_j). By the security properties of the identification scheme, the resulting protocol is secure against adaptive impersonation attacks, but furthermore, no adversary can by means of a man-in-the-middle attack, divert the communication to a different site Y_t, and pass there as X_i, even if X_i has the right of access at site Y_t.

4 Secure and Efficient Digital Signatures

4.1 Introduction

Digital signatures have now become widespread as an electronic alternative to the traditional handwritten signature. They are a key technique for many electronic transaction schemes now in use or proposed, such as electronic commerce schemes. The concept of a digital signature, as well as that of public key encryption, was put forward by Diffie and Hellman in their seminal paper [56] which kicked off the field of public key cryptography. Soon after, Rivest, Shamir and Adleman proposed a practical method, commonly known as RSA, that implements the ideas of Diffie and Hellman. While [56] proposes key exchange protocols based on exponentiation in a finite field, [100] considers protocols based on exponentiation in the ring of integers modulo a composite number. In the former case, security is obtained from the (supposed) difficulty of computing discrete logarithms, while in the latter case factoring integers is assumed to be intractable. For relevant results from the field of computational number theory, please consult [113].

Since then, many papers in the cryptographic literature have been devoted to the design of digital signatures. Roughly, one can distinguish three main directions.

1. Weaker intractability assumptions.
2. Better efficiency.

3. Variations on the concept.

Researchers have identified digital signatures with extended capabilities. First in this line of research, Chaum defined *blind signatures* [28]. This type of signature allows a receiver to obtain a signature on a message such that the message-signature pair is not completely hidden from the actual signer. Chaum used this intriguing concept to build electronic payment schemes where the anonymity of the payer is guaranteed. *Fail-stop signatures* [20, 79, 80] provide a means for the signer to take a forged signature to a court and prove that that it is indeed a forgery. *Undeniable signatures* [32, 33, 21] are like ordinary signatures except that cooperation of the signer is required when verifying a signature. These signatures are zero knowledge in the sense that the string that constitutes the signatures contains no information about its validity. Furthermore, the confirmation of a signature is a zero knowledge proof. A signer can deny a signature that he didn't compute, but can't deny one that he did compute. In some scenarios the fact that the signer must cooperate, may be a shortcoming of these undeniable signatures. This is alleviated in the notion of *designated confirmer signatures* [37], where additionally the verification can be delegated to designated parties. We will only be concerned with ordinary digital signatures.

On the theoretical side, it was important to establish the weakest intractability assumption supporting secure digital signature schemes. In a sequence of results Merkle [89], Bellare-Micali [5], Naor-Yung [91] and finally Rompel [101], it was established that the existence of one-way functions is necessary and sufficient for the existence of secure signatures. This result, though theoretically very important, does not give rise to a practical signature scheme. The construction, which is based on a general *one-way function*, uses a costly "bit-by-bit" signing technique in conjunction with tree authentication [89]. As a result, the size of signatures is $O(k^2 \cdot \log B_\sigma)$ bits, where B_σ is an arbitrary (polynomial) bound on the number of signatures to be made (the "signature bound" and k is a security parameter.

Benefiting from the special properties of *claw-free trapdoor permutations*, the secure scheme presented by Goldwasser, Micali and Rivest [70] achieves signatures of size $O(k \cdot \log B_\sigma)$ bits, with a public key of size $O(k)$ bits. Their scheme also uses a tree structure. Intractability of factoring integers is a sufficient assumption for the existence of the family of functions required for their scheme. Though yielding shorter signatures asymptotically, the size grows rapidly in practice as the number of signatures made increases. An interesting question is whether schemes with this efficiency or better exist under weaker cryptographic assumptions exist than given in [70].

Furthermore, [70] provides formal definitions of security for signatures that have since then been commonly accepted. The strongest notion of security combines the most flexible attack with the weakest target for the attacker:

Running a signer as a black-box, which may be queried in an *adaptive* fashion, the attacker must produce a valid signature on a message whose signature was not requested from the black-box (an *existential forgery*). If no attacker can efficiently perform this task, the signature scheme is called *not existentially forgeable under adaptive chosen message attacks.* Weaker forms of security defined in [70] are obtained by setting a stronger target for the attacker or giving it less control over the signer. For example, *not existentially forgeable under random message attacks* means that the attacker only gets to see signatures on uniformly chosen messages before it must compute an existential forgery. Even, Goldreich, and Micali [59], [60] have shown that signatures with the latter degree of security imply the existence of signatures satisfying the former security level. The transformation, however, has a linear (in the security parameter) blow-up of the size of signatures as a drawback.

Many practical digital signature schemes have been proposed, for instance, El Gamal [61], Feige-Fiat-Shamir [66], Schnorr [106], Guillou-Quisquater [76], Okamoto[93], ISO9796 [81] based on RSA, Digital Signature Algorithm [58], Bellare-Rogaway [7] and Pointcheval-Stern [97]. In these schemes the size of the signatures is $O(k)$ bits. Also, minimizing the amount of (on-line) computation for signer and receiver is an important issue in this type of result.

Although many of them are actually used in practice today, these schemes seem to have the property that their security is hard to analyze. We certainly do not mean to suggest here that their security is dubious. On the contrary, these schemes rely on common cryptographic assumptions, such as the difficulty of factoring or inverting RSA functions, the difficulty of computing discrete logarithms or the collision intractability of certain hash functions, and have so far resisted many years of cryptanalytic efforts.

However, none of these practical schemes has been shown to be secure in the sense of [70] provided that any of these mentioned cryptographic assumptions holds. This implies that, independently of their validity, these necessary and common cryptographic assumptions may still turn out to be insufficient for the security of these signature schemes.

Recently Bellare and Rogaway [6] and [7] (see also [97]) revived and further developed design of cryptographic protocols in the *random oracle model*, first used by Fiat and Shamir in this context. This model provides as a means to analyze the security of a large class of cryptographic schemes such as the practical schemes mentioned previously. Here, a cryptographic function can be modelled as an unrestricted black-box that first selects a truly random function with the desired input/output size (this is sometimes also called an *ideal hash-function*). Then it can be queried by any of the players in the scheme. Note that if the oracle is queried twice at the same input it will give the same output. But moreover, any sequence of input/output pairs will give no information about the output corresponding to a new input. Many of the schemes mentioned above are derived from identification schemes by replacing the challenge with a hash of the message to be signed and the preamble of the conversation. In [97] it is indicated that the security analysis of such schemes in the random oracle model comes close to showing that the underlying identification scheme is zero knowledge from the point of view of an honest verifier, while additionally it must be hard to pass as the prover when only given the public key. Different types of schemes are given in [6] and [7].

Nevertheless, security analysis of signatures in the random oracle model does not provide a proof of security in the sense of [70], since no reduction of any cryptographic assumption is given to the existence of the necessary random oracles (it gives however a better *argument* than with no proof at all).

Thus, based on the above, it is an open problem to design a secure and truly practical digital signature scheme, that may be used in today's or tomorrow's information systems.

Recently, progress has been made in this area. Starting with [57], it can be concluded that proven security based on RSA, moderate amount of computation and provision of any reasonable number of small-sized signatures, can be satisfied. This scheme yields practically much smaller signatures compared to, for instance, [70]. A drawback of their scheme is that all players must have reliable (available when needed and authentic) access to a large random string that is chosen in advance and then fixed. This makes the scheme less suitable for operation on devices that have limited storage capabilities, such as smartcards.

4.2 Our Contributions

The results in this chapter are based on [45], [43], [44] and [49]. Our goal is to construct *secure* (that is, not existentially forgeable under adaptively chosen message attacks) digital signatures with *low communication* complexity (that is, size of signature and public key must be "small"), under cryptographic complexity assumptions as weak as possible.

In Section 4.4 we review the Goldwasser-Micali-Rivest scheme [70] (the "GMR-scheme") and give an efficiency improvement of that scheme that allows to reduce the size of signatures by a factor of $\log l$, where l can take on any value of our choice. The price we pay is an increased size of a signer's key by a factor l and the requirement that all players have access to a shared random string of size lk bits, where k is a security parameter. The GMR-scheme is based on claw-free pairs of trapdoor permutations (see Section 2.5.3), while our scheme is based on special claw-free pairs of trapdoor permutations, defined in Section 2.5.4.

Next, in Section 4.5 we address the question whether the GMR scheme can be realized under weaker cryptographic assumptions than those used in [70], while preserving the size of signatures and public key. We will show that *linear* signature protocols (See also Section 2.2) are sufficient to guarantee, for any polynomial bound B_σ on the number of signatures to be made, the existence of secure signatures of size $O(k \log B_\sigma)$ bits and public key of size $O(k)$ bits. Thus we answer this question affirmatively, as we also show that our primitive can be based on special one-way group isomorphisms (see Section sec:3am.exist.sowgi) as well as claw free pairs of trapdoor permutations (see Section 2.5.3). Note that the latter assumption is not known to imply the former. We give an example of a one-way group homomorphism based on the difficulty of computing discrete logarithms, which is particularly efficient and serves to separate our results from those of [70]. To the best of our knowledge, linear signature protocols provide the weakest cryptographic assumption known to guarantee the existence of signature schemes not existentially forgeable under adaptive chosen message attacks, with signatures[1] of size $O(k \log B_\sigma)$ bits and public key of size $O(k)$ bits.

Providing security against adaptive attacks is an important goal in many cryptographic schemes. In an adaptive attack an enemy may deviate arbitrarily from the protocols and may, for instance, request an honest player to invert a certain cryptographic function controlled by the honest player, on an input of the attacker's choice. It may be the case that a clever choice of such inputs helps the attacker to obtain information that should remain in private possession of the honest player. Transforming cryptographic schemes that are secure against non-adaptive attacks to schemes that withstand adaptive attacks, is a desirable goal. Even more so when the transformation is efficient. Such efficient transformations may be helpful when designing cryptographic protocols for the following reason. It seems easier to first design a scheme that is secure in a weaker sense than desired, especially when one does not have to take into account adaptive attacks, than to design a scheme with the right security properties from scratch. The efficient transformation is then used to map the scheme into a new scheme with the desired security level. In Section 4.6 we show how signature schemes secure against random message attacks can be compiled into signature schemes secure against adaptive chosen message attacks, resulting only in a constant increase of the size of a signature. Thus we improve the transformation given in [60], which has a linear blow-up

Finally, in Section 4.7 we present a practical and secure signature scheme

[1]In both the GMR-scheme as well as in our scheme, the size of the private key is $O(k)$ bits.

based on RSA that is suitable for smartcards. We build on the construction given by [57] and modify their scheme in a number of ways to the effect that the necessity of a large shared random string is circumvented. To the best of our knowledge, our RSA-based signature scheme is the most efficient *and* secure signature scheme known to date.

4.3 Definitions

Formally, a digital signature scheme consists of a key generation algorithm G, a signing algorithm S and a verification algorithm V. A signer generates his key pair (pk, sk) by running G on input 1^k where k is a security parameter. The public key pk is placed in the signer's public directory while the secret key sk is private input to the signer. Given a message m from the message space $\mathcal{M}$, the signer computes a signature σ on m by running S on input (pk, sk) and m. A receiver of the signature verifies the validity of the signature with respect to public key pk by invoking V on input pk, σ and m. The algorithm V returns *accept* if the signature is valid, and *reject* otherwise. The algorithms are required to run in probabilistic polynomial time.

Consider a signer A with public pk. A *forgery* of A's signature is a valid signature with respect to pk on some message m, that is not computed by A. To capture the natural and strongest notion of security, [70] considers the most general attack against a digital signature scheme, *an adaptive chosen message attack*, and combines it with the weakest goal for the attacker, *an existential forgery*.

In this model, a polynomial time attacker $\mathcal{A}$ is given the public key pk, the triple (G, S, V), and the signing algorithm S instantiated with pk and sk, as a black-box. As part of the attack, $\mathcal{A}$ is allowed to request a polynomial number of signatures on messages of its choice. Moreover, $\mathcal{A}$ may choose these messages as a function of previously received signatures, the public key, etc. This models the idea that an attacker may first want a signer to generate signatures on particular messages, say $m_1, \ldots, m_t$, that may later help the attacker to forge a signature. The attacker is successful if it is now able to compute a valid signature $\tilde{\sigma}$ on a message $\tilde{m}$ different from any of the messages that $\mathcal{A}$ requested the black-box s to sign. More technically, this means that $\tilde{m} \neq m_1, \ldots, m_t$. If for any polynomially bounded attacker it holds that this task can only be completed with negligible probability, we say that the signature scheme (G, S, V) is *not existentially forgeable under adaptively chosen message attacks.* If an attacker cannot compute an existential forgery given just signatures on uniformly chosen messages instead of adaptively chosen ones, the scheme is called *not existentially forgeable under random message attacks.*

4.4 Shorter GMR-signatures

We present an efficient signature scheme that is not existentially forgeable under adaptively chosen message attacks. The main feature of our scheme is that any practical number of signatures can be made while the size of the signatures remains relatively small, under the condition that all users have access to a list of shared random strings.

More precisely, let integers l and d be fixed and let k be a security parameter. Given a list of l random k-bit strings shared by all users, at least l^d signatures can be made by each signer in our scheme, where the size of a public key is k bits. The size of a signature does not exceed $(3d-2)k$ bits. The schemes requires a fixed private input (corresponding to the private key) to the signer of size $(l+1)k$ bits, while it needs $(d-1)k$ bits of dynamic storage for employing tree-authentication.

A typical implementation value for the security parameter is $k = 1000$. Our scheme can be based on the difficulty of factoring integers. Let's assume, as an example, that 10^9 is a reasonable bound on the number of signatures a signer may wish to make ("the signature bound"). Let's also assume that signers and receivers have workstations at their disposal so that sufficient memory is available. To reduce bandwidth needed for sending a signature over the Internet, we would like the signatures to be "small." So when we set $d = 3$ and $l = 1000$, signatures have size at most $7 \cdot 10^3$ bits. The size of the shared list and the size of the private input for the signer together is then slightly over $2 \cdot 10^6$ bits, while the public key is only 10^3 bits. Roughly speaking, the GMR-scheme is the special instance of our scheme with $l = 1$. In the example above with a signature bound $B_\sigma = 10^9$, the size of signatures in the GMR-scheme can become 10 times as large.

The first secure signature scheme where such trade-offs between shared randomness and the size of signatures were investigated was proposed by Dwork and Naor [57]. Their scheme is based on RSA, while their method for achieving efficiency relies on special properties of RSA that seem to go beyond the properties of general trapdoor permutations.

With parameters set as above, the scheme of Dwork and Naor offers signatures of $4 \cdot 10^3$ bits. Their public keys and shared lists have the same size as ours. A drawback of our scheme compared to [57] is that the private storage for the signer is larger by a factor of l.

If one accepts the assumption that RSA-inversion is intractable, one might argue that the scheme of [57] is more desirable. However, in Section 4.7, we show how to modify their scheme so as to get rid of the shared list altogether. Thus for secure signatures based on RSA, our scheme of Section 4.7 compares favorably to [57].

Our contribution here is to is to show that a secure signature scheme with efficiency (in terms of the size of signatures) similar to [57] can be based on a general cryptographic assumption that is potentially weaker than an RSA assumption, namely the existence of a *special* family of claw-free trapdoor permutations, which we have shown to exist under the factoring assumption (Proposition 2.13).

As shown in Section 2.5.3, claw-free pairs of trapdoor permutations give rise to a special kind of Σ-protocols, namely signature protocols. Here, it is more convenient however, to use the common notation and terminology for such functions rather than our notation for Σ-protocols.

4.4.1 Notation

Let $f = (f_0, f_1)$ denote a claw-free pair of trapdoor permutations. Since for a fixed value of a, the function $f_{[a]}$ (see also Section 2.5.3) is a permutation of D_f, it makes sense to define its inverse. So, for any $b' \in D_f$, $(f_{[a]})^{-1}(b')$ denotes the pre-image of b' under $f_{[a]}$. The notation $f^{(j)}$ refers the j-th pair of permutations out of some sequence which will be clear from the context. The meaning of the notation $f^{(j)}_{[a]}(b)$ should now be clear from the above if one puts $f^{(j)} = (f^{(j)}_0, f^{(j)}_1)$ and applies the earlier definition of composition. For fixed $a \in \{0,1\}^*$, and for any b' in the domain of the pair $f^{(j)}$, the expression $(f^{(j)}_{[a]})^{-1}(b')$ denotes the pre-image of b' under the permutation $f^{(j)}_{[a]}$.

4.4.2 The Goldwasser-Micali-Rivest scheme

We briefly and informally describe the GMR-scheme from [70]. Let $\mathcal{F}$ be a family of claw-free pairs of trapdoor permutations (see Section 2.5.3). In the GMR-scheme a signer generates two such pairs $f = (f_0, f_1)$ and $g = (g_0, g_1)$ with known trapdoor information, and a random S from f's domain D_f. The public key is equal to (f, g, S). The trapdoor information is private input to the signer. Suppose that the signer expects to make at most 2^d signatures. The message space $\mathcal{M}$ can be, for instance, $\{0,1\}^k$, where k is the security parameter. We assume that elements from the domains D_f and D_g can be efficiently encoded into k bits.

As the first ingredient, f is used to construct a binary *authentication tree* of depth d, the root being S: All nodes are k-bit encodings of elements from D_f. Note that such a tree has exactly 2^d leafs. We give an inductive description of the creation of new nodes, starting at the root S. Let $x \in D_f$ be an internal node (that is, x is not a leaf) that has already been created. Its "left-child" x_L is chosen at random from D_f. Its "right-child" x_R is defined as $x_R \leftarrow f^{-1}_{[x_L]}(x)$. By our assumption on D_f x_L can be given as a k-bit string. Let x_{leaf} be any leaf in this tree, and let $x(0) = S, \ldots, x(d-1)$ denote the ancestors of x_{leaf}, and for each of these ancestors $x(i)$, let $x_L(i)$ and $x_R(i)$ denote their left-, and right-child, respectively.

Of course we have that our given $x_{leaf} \in \{x_L(d-1), x_R(d-1)\}$, and that $x(i) \in \{x_L(i-1), x_R(i-1)\}$, $i = 1, \ldots, d-1$. An *authentication path* $\text{auth}(x_{leaf})$ for x_{leaf} consists of the collection $\{(x(i), x_L(i), x_R(i))\}_{i=0}^{d-1}$. The validity of an authentication path is established by verifying whether $x(0) = S$ and whether, for $i = 0, \ldots, d-1$, $f_{[x_L(i)]}(x_R(i)) = x(i)$.

We can now describe the signature scheme. The signer does not compute the authentication tree in advance, but rather constructs it "on the fly". The most efficient way to do it, from a storage point of view, is to generate it depth first. When reaching the first leaf in the authentication tree, the signer is ready to make the first signature. Let's take a look at the construction of the first signature. Let x_{leaf} be this first leaf that is reached. Now g and D_g come into the play. The signer selects a random x_b (b for "bridge") from D_g and authenticates it by computing $y_b \leftarrow f_{[x_b]}^{-1}(x_{leaf})$. Again, we can say that x_b is given as a k-bit string, by our assumption on D_g. A message $m \in \mathcal{M}$ is finally authenticated by computing $z_b \leftarrow g_{[m]}^{-1}(x_b)$. The signature on m consists (x_{leaf}, y_b, z_b) and the authentication path $\text{auth}(x_{leaf})$. A receiver of this signature establishes the validity of the signature by checking whether $\text{auth}(x_{leaf})$ is valid and whether $f_{[x_b]}(y_b) = x_{leaf}$ and $g_{[m]}(z_b) = x_b$.

The next signature can be made when the second leaf is available, et cetera. In this way, the signer can compute 2^d signatures on any messages $m \in \{0,1\}^k$. One final remark about the construction. As long as some node in the tree can be part of a future authentication path, it should be stored. But as soon as this is not the case anymore, it can be deleted. In this way, the signer need only store a linear number (as a function of d) nodes at any time.

The proof of security for this signature scheme basically shows that given just the trapdoor information for one of the two f and g, we can define a signer that is indistinguishable from one who is given both trapdoors. In particular, the simulator can give out 2^d signatures, just like the real signer above. So when the "simulated" signer is adaptively queried by an attacker, it cannot be biased as to which of the trapdoors the simulation knows. With out loss of generality, assume that the attacker queries the simulator 2^d times, so that "the whole tree is used up". The next step is to show that from an existential forgery *and* the history of the attack (all signatures output by the simulation), we can efficiently derive a claw for either f or g. But since the attacker cannot be biased, with probability $1/2$ this will be a claw with respect to the permutation pair for which the simulator was not given the trapdoor information! This then contradicts the existence of an efficient and successful attacker, since we have assumed that the permutation pairs were claw-free. This type of argument is used throughout this chapter.

4.4.3 The Modification

Obviously, the use of an l-ary authentication tree would result in signatures with an authentication path smaller by a factor of $\log_2 l$. If $l = 1000$ for instance, the size is decreased by a factor of 10, approximately. The first idea that comes to mind when trying to realize this, is using l different claw-free trapdoor permutation pairs f, and try to construct l new nodes out of a given one rather than 2, each of these l new nodes authenticated by a different pair f. We then encounter two problems.

The first is that these f's have potentially different domains D_f. It turns out that even when we take the factoring based implementation of $\mathcal{F}$ (see Section 2.5.3), this is the case and the proof of security would fail for some subtle reasons. So, from which set should the nodes be chosen? These problems are solved once we are given a family of such functions that permute the *same* domain. Such a *special* family is defined in Section 2.5.4, where it is also proved to exist under the assumption that factoring integers is intractable. Let D now denote that common domain. The nodes in the tree are selected at random from D and each internal node has l children. If $f^{(i)}$ is the i-th pair of permutations and if x is an internal node, then its child x_i is authenticated by computing $(f_{x_i}^{(i)})^{-1}(x)$. Leafs x_{leaf} can be used to compute a signature $g_{[m]}^{-1}(x_{leaf})$ immediately.

Now, the second problem arises. How to authenticate all these pairs f? One way would be to view them all as part of the signer's public key. But then its size increases by a factor l, which will results in an enormous directory of public keys if l is large. The solution is found when we borrow the idea of [57] to use a large shared random string. This string is once generated at random in a way trusted by all players and then fixed. The players only need reliable access to the string. The trick is that each signer uses the g-pair from above in conjunction with the shared string to authenticate each of its l pairs f. Furthermore, the shared string is viewed as a collection of l roots for authentication trees. Then a signer starts to construct the first l-ary tree with depth $d-1$ and with root given by the shared random string. This uses at most $d-1$ out of the l pairs f. The signature consists of the authentication path in this tree plus the at most $d-1$ authentication values for the used pairs f. When the first tree is used up, which is the case when l^{d-1} signatures have been made, the signer is free to go on to the next with root as given by the shared random string, and so on. This way, the length of a signature is $O(k \log_l B_\sigma)$ bits instead of $O(k \log_2 B_\sigma)$ as in the GMR-scheme. The size of the public key is just k bits. However, the shared random string will have size lk bits, and each signer needs a little more than lk bits of private storage.

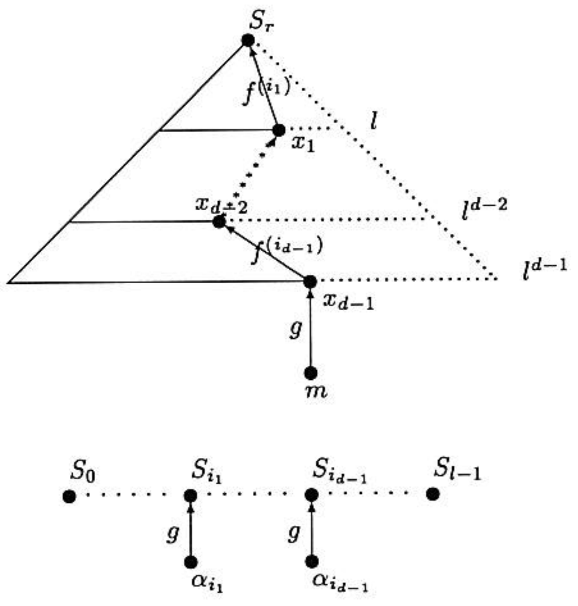

FIGURE 4.1. Overview

4.4.4 Preliminaries

In the signature scheme described in Section 4.4.5, the signers have access to the generator G for family $\mathcal{F}$ of special claw-free trapdoor permutations. If k is the security parameter, the corresponding message $\mathcal{M}(k)$ space is equal to $\{0,1\}^k$, although this choice is arbitrary. Recall from Section 2.5.4 that each permutation pair generated by the special generator $G(1^k)$, has a domain in which the set $\{0,1\}^{\overline{k}}$ can be embedded. The embedded set will then take up a non-negligible fraction of that domain. The value $\overline{k}$ only depends on k. Also, the signers agree on integers l and d such that l^d is larger than or equal to the expected number of signatures to be made by an individual signer. Furthermore, the signers must have access to a list L of shared random numbers and an auxiliary algorithm DFS, both to be explained hereafter.

The Shared List L

Before the signers set up their instances of the signature scheme, they agree on a list L of shared random numbers. This list is to be viewed as part of the system constants. Given the security parameter k and the integer l, the list consists of l bitstrings $S \in \{0,1\}^{\overline{k}}$. The list is written as follows.

$$L = \{S_0, \ldots, S_{l-1}\}.$$

Ways of generating L can vary from observing a common physical source of randomness to executing well-known cryptographic techniques for establishing common random strings. Once this list is generated, all signers and receivers of signatures must have access to this list.

The Algorithm DFS

DFS gradually builds l full l-ary trees with depth $d-1$ and with roots $S_0, \ldots, S_{l-1}$ respectively. DFS selects the nodes at random from the set $\{0,1\}^{\overline{k}}$. The trees are constructed sequentially, in the sense that DFS starts to build the j-th tree after the first $j-1$ trees are completed. Each of the trees separately, is developed in depth-first order. The algorithm has access to the security parameter k and the integers l and d.

The construction is gradual in the following sense. At the i-th call (denoted by DFS(i)) ($i = 1 \ldots l^d$), DFS creates a path $x_1, \ldots, x_{d-2}$ to a new leaf x_{d-1} (a node at depth $d-1$) and outputs these nodes. Notice that with the i-th call a path to a new leaf in the r-th tree with root S_r is output, with $r = i - 1 \text{ div } l^{d-1}$. It will be assumed that the order in which the nodes in the path are output is such that x_1 is a child of S_r and x_j is a child of x_{j-1}, for $j = 2, \ldots, d-1$ Also, DFS(i) outputs indicators i_j, with $j = 1 \ldots d-1$. These indicators point out that x_j is the i_j-th child ($1 \le i_j \le l$) of its ancestor x_{j-1}.

In the mean time, DFS administrates sufficient information so as to be able to carry on with the construction. It only has to store the latest path to a new leaf and the corresponding indicators. Notice that it is sufficient, as far as concerns the indicators, to store i; Write $c \leftarrow (l^{d-1} - l)/(l-1)$ then write $(i-1) \bmod l^{d-1} + 1 + c$ in l-ary representation, say, it is equal to $\sum_{j=1}^{d-1} i_j \cdot l^{j-1} \bmod l^{d-1}$. It follows easily that these i_j are in fact the required indicators.

When a tree is completed, storage regarding this tree (excluding the root) is deleted. Neglecting r and the indicators i_j whose storage is negligible, the storage needed for DFS does not exceed $(d-1)k$ bits at any time.

4.4.5 Description of the Signature Scheme

Initialization: The signer generates $l+1$ independent pairs of claw-free trapdoor permutations

$$g, f^{(0)}, \ldots, f^{(l-1)},$$

with known trapdoor information, by running $G(1^k)$ $l+1$ times. It is assumed that $\alpha_j \in \{0,1\}^k$ is a suitable encoding for $f^{(j)}$, for $j =$

$0 \dots l-1$. The signer places his public key

$$pk \leftarrow g$$

in his public directory. The trapdoor information to all these claw-free permutation pairs is private input to the signer.

Signing: For a schematic overview of the signing procedure, see Figure 4.1. Let a message $m \in (k)$ be given. Suppose this is the i-th execution of this procedure ($i = 1 \dots l^d$). Put $r \leftarrow i - 1 \text{ div } l^{d-1}$. The signer first invokes DFS.

$$(x_1, i_1, \dots, x_{d-1}, i_{d-1}) \leftarrow \text{DFS(i)}.$$

The authentication values β_{i_j} for the $f^{(i_j)}$, he computes

$$\beta_{i_j} \leftarrow (g_{[\alpha_{i_j}]})^{-1}(\kappa_g(S_{i_j})),$$

for $j = 1 \dots d-1$. Here κ_g is the efficient embedding of $\{0,1\}^{\overline{k}}$ into D_g (see Definition 2.6). Next, he computes the authentication values for the nodes x_j as follows.

$$y_1 \leftarrow (f_{[x_1]}^{(i_1)})^{-1}(\kappa_{f^{(i_1)}}(S_r))$$

$$y_j \leftarrow (f_{[x_j]}^{(i_j)})^{-1}(\kappa_{f^{(i_j)}}(x_{j-1})),$$

for $j = 2 \dots d-1$. Finally, he authenticates the message m, by computing

$$z \leftarrow (g_{[m]})^{-1}(\kappa_g(x_{d-1}))$$

The signature $\sigma_{pk}(m)$ is as follows.

$$\sigma_{pk}(m) \leftarrow (r, z, y_1, i_1, \alpha_{i_1}, \beta_{i_1}, \dots, y_{d-1}, i_{d-1}, \alpha_{i_1}, \beta_{i_{d-1}})$$

Verification: The receiver puts

$$\sigma_{pk}(m) \leftarrow (Z, Y_1, i_1, A_{i_1}, B_{i_1}, \dots, Y_{d-1}, i_{d-1}, A_{i_1}, B_{i_{d-1}}),$$

retrieves the S_{i_j} from the shared list L and the signer's public key pk, and interprets the A_{i_j} as claw-free function pairs $h^{(i_j)}$ for $j = 1 \dots d-1$. The receiver proceeds by verifying whether

$$g_{[A_{i_j}]}(B_{i_j}) = \kappa_g(S_{i_j}),$$

for $j = 1 \dots d-1$. Next, he computes $X_{d-1} \in \{0,1\}^{\overline{k}}$ such that

$$\kappa_g(X_{d-1}) \leftarrow g_{[m]}(Z),$$

and recursively computes $X_j \in \{0,1\}^{\overline{k}}$ such that

$$\kappa_{f^{(i_j)}}(X_j) \leftarrow h_{[X_{j+1}]}^{(i_j)}(y_{j+1}),$$

for $j = d-1 \dots 0$. If $X_0 = S_r$, then the signature is accepted.

4.4.6 *Proof of Security*

We prove the following theorem.

THEOREM 4.1 *Suppose that a family $\mathcal{F}$ of special claw-free pairs of trapdoor permutations exists. Let l and d be fixed positive integers and let k be the security parameter. Then the signature scheme, described above, is not existentially forgeable under adaptively chosen message attacks. The size of the public key is k bits, while the size of the secret key is lk bits. The size of the shared list L is lk bits. At least l^d signatures can be made. The size of a signature is at most $(3d-2)k$ bits.*

PROOF. Let a shared list L and a signer with public key g be given and let, as before, $f^{(0)}, \dots, f^{(l-1)}$ denote the l permutation pairs that the signer uses as part of the construction of the authentication-tree. If an attacker is able to forge a signature on a message that the real signer hasn't signed, we can distinguish between two cases.

1. The forgery uses a subset of these $f^{(j)}$'s to authenticate the nodes in the forged signature.
2. The forgery uses at least one permutation pair $\tilde{f}$ different from all $f^{(j)}$'s.

First, we conduct the proof by treating the permutations $f^{(0)}, \dots, f^{(l-1)}$ as part of the public key. Thus, the authentication values β_{i_j} are not regarded as part of the signature in this part of the proof. This corresponds to the first case above. Also, in this case we will only consider a forgery that hooks into the first tree with root S_0. The proof that takes all l trees into account, follows immediately.

If these matters are settled, the proof is finalized by observing that a forgery that uses some permutation pair different from any of the $f^{(j)}$ for $j = 0 \dots l-1$, gives rise to a claw with respect to the permutation pair g, which corresponds to the second case above. The generation of the shared list L will be depending on the simulated signer. This is no problem, since it will be indistinguishable from the list L as generated according to Section 4.4.4.

For simplicity of the proof, it is assumed that the verifier of a signature does a length-check by verifying whether the signature uses exactly $d-1$ nodes (excluding the root). Disposing of this length-check is available at the price of a slightly more technical, but nevertheless correct proof of security.

Without loss of generality, we may assume that the attacker outputs the forgery after exactly l^d calls to the signer: if he can compute a forgery on a message $\tilde{m}$ after fewer calls, we simply redefine the attacker so that he will make additional calls on a message that is not equal to $\tilde{m}$.

The compilation works as follows. Suppose we are given a pair $h = (h_0, h_1)$ from the family $\mathcal{F}$ as generated by the generator G on input 1^k, but not its trapdoor information s_h. We will use the attacker to compute a claw for this pair h. We start by running $G(1^k)$ l times in order to obtain a list of l claw-free pairs of trapdoor permutations with known trapdoor information.

Secondly, we select a random $r \in \{-1, 0, \ldots, l-1\}$. If $r > -1$, we will add the pair h to the list such that it becomes the $(r+2)$-nd entry. The resulting list of $l+1$ pairs consists of $g = (g_0, g_1), f^{(0)} = (f_0^{(0)}, f_1^{(0)}), \ldots, f^{(l-1)} = (f_0^{(l-1)}, f_1^{(l-1)})$, where $h = f^{(r)}$. If $r = -1$, we put h at the head of the list such that $h = g$.

Next we demonstrate that in all cases r with $-1 \leq r \leq l-1$, we can set up a signer for the signature scheme, whose permutation pairs are those in the list of $l+1$ claw-free pairs of trapdoor permutations as defined above, such that one cannot distinguish between a real signer and this "simulated" signer (who doesn't know the trapdoor information for exactly one of the $l+1$ pairs). In particular, it follows from this (perfect) simulation that all cases are indistinguishable from one another.

Running the attacker on this simulated signer, we may expect the attacker to produce a forgery with essentially the same success probability as in real life, by the perfectness of the simulation. Our argument is completed by the observation that from a forgery and the signatures output as a result of the attacker's calls to the simulator, we can derive a claw for at least one of the $l+1$ permutation pairs. By the perfectness of the simulation, this forgery leads to a claw exactly for the pair h, for which we were not given the trapdoor information, with probability $1/l+1$.

As argued above, it is now sufficient to describe our simulation, and demonstrate the perfectness of the simulation and prove that a forgery leads to a claw for one of the $l+1$ permutation pairs.

$r > -1$:

We have to build a full authentication tree of branching degree l and depth $d-1$ while the trapdoor information to $f^{(r)}$ is not given for exactly one r with $0 \leq r \leq l-1$. This tree is now built from the bottom up. Observe that it works for trees of depth 1 as follows.

Choose $x_0, \ldots, x_{l-1} \leftarrow \{0,1\}^{\overline{k}}$. Then, select $y_r \leftarrow D_{f^{(r)}}$ and compute $f^{(r)}_{[x_r]}(y_r)$. If this value is in the image of $\kappa_{f^{(r)}}$, we compute $x \in \{0,1\}^{\overline{k}}$ such that $\kappa_{f^{(r)}}(x) = f^{(r)}_{[x_r]}(y_r)$ and we halt. Otherwise, we keep selecting $y_r \leftarrow D_{f^{(r)}}$ until this is the case. Note that, in any case, x is distributed uniformly over $\{0,1\}^{\overline{k}}$ and is independent of $x_0, \ldots, x_{l-1}$. By our assumptions on the embeddings, the process just described has non-negligible success probability of being completed in polynomial time. After this, for $i = 0, \ldots, l-1$ and $i \neq r$, y_i is computed as $y_i \leftarrow \left(f^{(i)}_{[x_i]}\right)^{-1}(\kappa_{f^{(i)}}(x))$. Note that all values y_i, for $i = 0, \ldots, l-1$, follow deterministically from $x, x_0, \ldots, x_{l-1}$. The value x is to be viewed as the root, and $x_0, \ldots, x_{l-1}$ as its children.

Suppose now, as an induction hypothesis, that we can build such authentication trees with depth $n-1$. Now, given l such trees, we can easily build an authentication tree of depth n by putting the roots of these l trees of depth $n-1$ in the roles of $x_0, \ldots, x_{l-1}$ as above and compute the root in the same way as x was computed above.

All nodes in the authentication trees thus constructed have uniform distribution over $\{0,1\}^k$ and are independent of each other and anything else. The corresponding authentication values follow deterministically.

After a full authentication tree with resulting root S_0, depth $d-1$ and branching l has thus been constructed and the attacker is given the public key and S_0, the simulator can start answering the calls by the attacker, as the trapdoor information to the pair g is given in this case where $r > -1$. As the real signer, the simulator will reveal paths to new leaves (as if) constructed in depth-first fashion.

$r = -1$:

In this case, we are given the trapdoor information for the pairs $f^{(i)}$, for $i = 0, \ldots, l-1$, but not the trapdoor information for the pair g. This time, we will select the root S_0 for the authentication tree as in the description of the scheme and present the resulting public key and S_0 to the attacker.

Next, we construct the authentication tree top down interleaved with the requests from the attacker, as follows. Given a request for a signature on a message m, we simply compute $g_{[m]}(y)$ for $y \leftarrow D_g$ and with non-negligible probability, $g_{[m]}(y)$ is in the image of κ_g and we compute $x \in \{0,1\}^k$ such that $\kappa_g(x) = g_{[m]}(y)$. Finally, we authenticate x as in the description of the scheme. In this way, the simulator is able to answer all calls by the attacker. Again, as in the case $r > -1$, the nodes in the authentication tree have uniform distribution over $\{0,1\}^k$ and are independent of each other and anything else. Also, the corresponding authentication values follow deterministically. As the real signer, the simulator constructs the tree in depth-first fashion.

From the above analysis we conclude that the simulation is perfect and runs in probabilistic polynomial time. Next, we may assume that the attacker outputs a forgery with some probability ϵ. Let the forgery, on a new message $\tilde{m}$, be

$$\tilde{z}, \tilde{y}_1, i_1, \ldots, \tilde{y}_{d-1}, i_{d-1}.$$

From this forgery, we first re-compute the corresponding nodes

$$\tilde{x}_1, \ldots, \tilde{x}_{d-1},$$

as is done in the verification phase of the signature scheme. Let j be the largest index such that $\tilde{x}_1, i_1 \ldots, \tilde{x}_j, i_j$ is a path in the authentication tree constructed by the simulator, starting at the root S_0. Recall that it was assumed that the attacker makes l^{d-1} calls, so the authentication tree is fully constructed. There are two possibilities now. The first is, $j = d-1$. But this means that $\tilde{x}_{d-1}$ is a leaf, and since $\tilde{m}$ is "new" and $\tilde{x}_{d-1}$ must have been used by the simulator to make a signature on a message $m \neq \tilde{m}$ (with corresponding authentication value z), we must have a claw $g_{[m]}(z) = g_{[\tilde{m}]}(\tilde{z}) = \kappa_g(\tilde{x}_{d-1})$ This happens with some probability ϵ_1.

If, on the other hand $j < d$ with probability ϵ_2, we have that the i_{j+1}-st child x_{j+1} of $\tilde{x}_j$ (with corresponding authentication value y_{j+1})) in the tree output by the simulator is different from $\tilde{x}_{j+1}$ by definition of j.

This results in a claw for the pair $f^{(i_{j+1})}$, since we have $f^{(i_{j+1})}_{[x_{j+1}]}(y_{j+1}) = f^{(i_{j+1})}_{[\tilde{x}_{j+1}]}(\tilde{y}_{j+1}) = \kappa_{f^{(i_{j+1})}}(\tilde{x}_j)$.

This part of the proof is completed by observing that, from the perfectness of the simulation, with probability at least

$$\frac{\epsilon_1}{l+1} + \frac{\epsilon_2}{l+1} = \frac{\epsilon}{l+1}.$$

The claw we have obtained will be with respect to the pair h, for which we were not given the trapdoor information. Thus breaking the claw-freeness of the family $\mathcal{F}$.

Recall that the above analysis treated the permutations pairs $f^{(0)}, \ldots, f^{(l-1)}$ as part of the public key. The remaining part of the proof is to show that it is infeasible for the attacker of the full scheme to produce a forgery that uses at least one permutation pair $\tilde{f}$ different from all $f^{(j)}$'s. In the above analysis in case $r = -1$, let α_j be a description of the pair $f^{(j)}$. If we generate the j-th entry of the list L as

$$S_j \leftarrow g_{[\alpha_j]}(\beta_j),$$

for a random $\beta_j \in D_g$ such that S_j is in the image of κ_g for $j = 0 \ldots l-1$, then a forgery that uses a permutation pair $\tilde{f}$ different from all $f^{(j)}$'s immediately leads to a claw for the pair g. If $r > 1$, the list L is generated as in the simulations above. Thus, we conclude that if the attacker can compute a forgery in the signature scheme, with probability ϵ, then we can compile the attacker into an algorithm for finding claws for the family $\mathcal{F}$ with probability $(\epsilon)/(l+1)$ in essentially the same time. □

4.4.7 Alternative Descriptions

There are various implementations of the basic ideas that underly our construction in Section 4.4.5. We mention one alternative. For clarity of exposition, the signature scheme is described in such a way that one authentication tree after another is completed, and that only the leaves of those trees are used for making signatures. However, one can also use the i-th (internal) nodes for making a signature, instead of the i-th *leaf* that is reached. Furthermore, one can develop the trees *level after level* instead of sequentially. Thus, one gains the potential of making an additional (at least) l^{d-1} signatures, whose size is at most $(3d-5)k$ bits. The proof of security is easily adapted to this scenario.

We also note that, if desired, the shared list L can be traded for large public keys and slightly shorter signatures. This is done by simply removing L from the description and requiring that the permutation pairs $f^{(i)}$ are included in the public key. However, the public directory where public keys are registered will become a factor l larger. If public key certificates are used instead, these have to be taken into account as well. It would then be reasonable to have a separate certificate for each pair $f^{(i)}$. The size of the resulting signature might then be equal to the size of a signature in our scheme. Note however, that in order for this public key certificate based option to be as secure as our scheme, the certification authority must also use a secure signature scheme. This scheme must then handle at least $N(l+1)$ signatures, where N is the number of participants. In certificate-based variant, the certifying authority must only generate N certificates. In many applications, the alternative with large public keys may result in a less desirable signature scheme.

4.5 Secure Signatures based on Interactive Protocols

4.5.1 Introduction

Given only a Σ-protocol of a certain type as a primitive, we can build a (non-interactive) signature scheme that is secure in the strongest sense of Goldwasser, Micali and Rivest (see [70]): not existentially forgeable under adaptively chosen message attacks. There are numerous examples of primitives that satisfy our conditions, such as the identification schemes of Feige-Fiat-Shamir, Schnorr, Guillou-Quisquater, Okamoto and Brickell-Mc.Curley ([66], [106], [76], [93], [26]).

More precisely, our primitive is a particularly efficient *signature protocol* (see Section 2.2). Recall that a signature protocol is a Σ-protocol satisfying collision-intractability and special honest verifier zero knowledge. Moreover, the length of a challenges is larger than the length of a first message of a conversation. In the constructions to follow we will additionally require that the protocol is *linear* in the sense that the length of a conversation as well as the length of a challenge satisfies some linear expression in the security parameter k.

A main consequence is that efficient and secure signature schemes can now also be based on computationally difficult problems other than factoring (see [70]), such as the discrete logarithm problem. Here, by efficient we mean signatures of size $O(k \log B_\sigma)$ bits where B_σ is a bound on the number of signatures to be made, and public keys of size $O(k)$ bits.

In fact, the existence of certain one-way group isomorphisms (see Section 2.5.2) is a sufficient assumption to support our construction. Furthermore, our construction can also be based on claw-free pairs of trapdoor permutations (which are not known to imply one-way group homomorphisms). When instantiated for these functions, our scheme is essentially identical to the scheme of [70]. As our general scheme also has the same efficiency as [70] in terms of the size of signatures, our results can be viewed as a generalization of [70]. *Linear signature protocols* can be viewed as the weakest cryptographic assumption known to be sufficient for the existence of secure signatures with size $O(k \log B_\sigma)$ bits and public keys of size $O(k)$ bits.

However, cryptographic protocols are usually instantiated using a specific intractability assumption, such as collision-intractability of a hash function, the difficulty of factoring integers or computing discrete logarithms. Although the general assumptions mentioned above are believed to be unrelated, our claim that we generalize the results of [70] is best argued when we give a specific intractability assumption that separates our results from [70].

To this end, we give a particularly efficient example of our scheme based on discrete logarithms. Other efficient instances of our scheme are based on the difficulty of factoring integers. However, under the difficulty of factoring assumption one can construct a family of claw-free pairs of trapdoor permutations. Since it is believed to be unlikely that one can construct secure trapdoor functions, based on the discrete logarithm problem, the example of our scheme based on discrete logarithms shows that we extend [70].

4.5.2 Notation

Let (A, B) be a signature protocol for relation R and generator G (see Section 2.2.7). Let M denote the special simulator and let k be the security parameter. As before (see Section 2.1), let $a(\cdot)$ denote the algorithm for A to compute the first message of the conversation and let $z(\cdot)$ denote the algorithm to compute the reply given B's challenge. The length of the first message is denoted t_A and the length of the challenge is given by t_B. By definition of a signature protocol we have $t_B > t_A$. With the algorithm ϕ the verifier B checks the validity of a conversation. We will also explicitly use the random string R_A from Section 2.1 in the description of our results. The length of the random tape R_A is denoted by t_R.

4.5.3 Overview of Our Method

Roughly speaking, to compute a signature, a signer will act as both A and B and run their respective algorithms. A signature will consist of a collection of conversations from the protocol (A, B) and all the receiver of the signature will have to do is to check the validity of those conversations.

Initially, a signer will generate two independent instances (x_0, w_0) and (x_1, w_1) of R for the prover A and a first message a_0 with respect to (x_0, w_0). For this task, the algorithms $G(\cdot)$ and $a(\cdot)$ are at the signer's disposal. The public key then consists of the two public strings x_0 and w_0, and the first message a_0, while the private key consists of w_0 and w_1 (and the random bits that are used to generate a_0).

When given a message m to be signed, our signer computes a conversation in (A, B) where m is the challenge, say b_0, m, s_0. But this time the other instance (x_1, w_1) is used. However, this step could be completed by any party, since the special simulator M is available to anyone. Therefore, the signer also interprets b_0 as a challenge and computes the reply z_0 given the first message a_0 and the key pair (x_0, w_0), and thereby effectively authenticates b_0. For this, the signer uses the algorithm $z(\cdot)$. This approach provably thwarts any forgery attempts. The receiver of the signature merely checks the validity of both conversations by using ϕ and verifies whether the first message of the latter conversation is part of the public key.

Of course, this is just part of the whole story. The method above only shows how to make one signature; if this process would be repeated, and the next signature on a different message would be computed, the signer reveals two conversations for the same key pair with the same first message but with different challenge values. This is very much the same as giving away the secret key! To get around this and enable the signer to compute many more signatures, the signer not only authenticates b_0 as above but at the same time authenticates two new first messages a_1 and a_2 with respect to (x_0, w_0). This is done by authenticating the concatenation of b_0, a_1 and a_2 instead of just b_0. Now, a_1 and a_2 will in the two subsequent signatures play the role of a_0. Obviously, this construction can inductively be carried on to the effect that the signer can make any polynomial number of signatures. Note that as more signatures are made, the number of conversations that a signature consist of grows: each of the a-values is authenticated by a previous one, and the final one must be equal to a_0. Since our way of generating these a-values gives rise to a *binary authentication tree* (see Section 4.4.2), the number of conversations in a signature is a logarithmic function of the number of signatures made so far.

In our proof of security will use a simulation technique to set up a simulated signer that looks like a real signer from the point of view of an attacker. However, the simulated signer misses one of the private inputs w_0 and w_1. Our technique makes extensive use of the special simulator M for the protocol (A, B). We then show that a forgery will immediately lead to a collision with respect to one of the two instances x_0 and x_1. But since our simulation is indistinguishable from a real signer, as seen from the outside world, with good probability we get a collision for exactly that instance whose private information our simulator was not given. This is how we show that forging signatures contradicts the assumption that our primitive protocol (A, B) is collision intractable.

The overview of our method hides an important aspect. The length of challenges in the protocol (A, B) might simply be too small to authenticate the necessary information. Recall that in the description given above, we must be able to handle challenges as large as three times the length of a first message.

For example, if we take (A, B) to be Schnorr's protocol [106] (see also Section 2.3), it is clear that the length of any challenge is at most equal to the length of any encoding of the elements in G_q. Since the first message a in that protocol is just a random element from G_q, we have that $t_B \leq t_A$. This is insufficient for our purposes, since in our description above we required that $t_B \geq 3t_A$ (in our formal treatment, $t_B \geq 2t_A$ will be sufficient).

A general way to increase the length of challenges in signature protocols is given by Proposition 2.4. In fact, given any signature protocol and any desired challenge length, we can transform the given protocol to a new signature protocol (for the same relation and generator) that has the desired challenge length while the length of the first message remains the same. We have to be careful, however, since this transformation increases the size of the conversations.

For instance, let's say that our original signature protocol (A, B) has the following properties: $t_A = |x| = k$, and $t_A + 1 = t_B$ where as usual x is the common input and k is the security parameter. We can easily give examples of signature protocols satisfying this. Then to achieve $t_{B'} = 2t_{A'}$ in the resulting protocol (A', B'), we get that the length of the conversations in (A', B') is at least $t_{B'}t_A \geq 2t_A^2 = 2k^2$ bits (recall that $t_A = t_{A'}$ in the transformation). Clearly, this is too much for our purposes here. To alleviate this situation, we define linear signature protocols in the next section.

4.5.4 Linear Signature Protocols

We now define *linear signature protocols*, the weakest cryptographic assumption we have been able to find that supports digital signatures not existentially forgeable under adaptive chosen message attacks, with signatures of size $O(k \log B_\sigma)$ bits and public key of size $O(k)$ bits (see Section 4.5.7).

DEFINITION 4.1 *Let (A, B) be a signature protocol (see Section 2.2) for relation R and generator G such that the length of a conversation is linear* *in the security parameter k. Suppose furthermore that the challenge length t_B satisfies $t_B \geq (1+c)t_A$ for some positive real constant c, where k is the security parameter. Then (A, B) is called a linear signature protocol.*

Recall that a conversation includes the common string. A variant of Proposition 2.4 will provide a way to map a linear signature protocol (A, B) with $t_B \geq (1+c_1)t_A$, for some positive real constant c_1 to a linear signature protocol (A', B') with $t_{B'} \geq (1+c_2)t_{A'}$ for any positive real constant $c_2 \geq c_1$ (recall that $t_A = t_{A'}$ in the construction of Proposition 2.4, and that the length of the common string is invariant). The proof of Proposition 2.4 uses a "chaining technique" to facilitate larger challenges. In fact, per bit of the new and larger challenge in (A', B') it creates a conversation in (A, B).

But we can easily extend this approach and take $c_1 t_A$ bits per link in the chain. In this way, the chain consists of at most a constant number of conversations in (A, B), which is just a constant blow up. So linearity is preserved. We have the following proposition

PROPOSITION 4.1 *Let (A, B) be a linear signature protocol for relation R and generator G. Suppose $t_B \geq (1+c_1)t_A$, for some positive real constant c_1. Then there exists a linear signature protocol (A', B') for relation R and generator G, satisfying $t_{B'} \geq (1+c_2)t_{A'}$ for any positive real constant $c_2 \geq c_1$.*

Thus, when we require in our constructions that a linear signature protocol has challenge length that is a constant number of times larger than the length of the first message (the authentication length), we can invoke Proposition 4.1 to get it right.

4.5.5 Existence of Linear Signature Protocols

Section 2.5 provides a number of cryptographic assumptions under which signature protocols are shown to exist; most notably one-way group homomorphisms, claw-free pairs of trapdoor permutations, the discrete logarithm assumption, the RSA-assumption or the factoring assumption.

More precisely, Proposition 2.12 in Section 2.5.3 can be seen to guarantee *linear* signature protocols based on claw-free pairs of trapdoor permutations, by just choosing the challenge length in that protocol appropriately. Recall that the factoring assumption is sufficient for the existence of these functions.

Proposition 2.8 in Section 2.5.2 yields *linear* signature protocols based on special one-way group isomorphisms; concrete ones are given by the two examples based on discrete logarithms and RSA.

4.5.6 The Algorithm DFS

Our signature scheme from Section 4.5.7 employs a binary authentication tree. The nodes of this tree correspond to random strings that are used as part of the generation of signatures. Since signatures comprise of conversations in a linear signature protocol (A, B), these random strings actually constitute the randomness R_A required for the algorithms of A as described in Section 2.1. Recall from Section 4.5.2 that the length of R_A is denoted t_R (we leave its dependence on the security parameter k implicit in this notation).

Let d be a fixed positive integer and let $\rho_0 \in \{0,1\}^{t_R}$. The algorithm DFS gradually develops, in depth first fashion, a full binary tree of depth d. We set the level of the root to 0. The root ρ_0 of the tree and the depth d are given as input to the algorithm. DFS selects the nodes at random from $\{0,1\}^{t_R}$.

At the i-th call (denoted by $DFS(i)$, $i = 1 \dots 2^d$), DFS outputs the i-th leaf, say ρ_d accompanied by an extended path to the root a_0. This path is extended in the sense that not only the ancestors of the leaf are given, but their siblings as well. So, for instance, at the first call DFS creates d nodes that constitute a path to the first leaf and d siblings of the ancestors of the first leaf. Note that these latter nodes also include the second leaf.

This extended path, including the leaf, is written

$$\rho_0, \rho_1, \rho_1', \dots, \rho_{d-1}, \rho_{d-1}', \rho_d, \rho_d'$$

where the indices tell at which level a node resides. The strings ρ_i, $i = 0 \dots d-1$, reflect the ordinary path from the leaf to the root, while the strings $\rho_1', \dots, \rho_d'$ are the remaining siblings of the nodes on the path from the leaf ρ_d to the root ρ_0.

DFS does not need to store the complete history of the construction. Those parts of the tree that will not appear anymore in future outputs are erased. Roughly speaking, it is sufficient for DFS to store the latest extended path. The required storage capacity of the algorithm is then $O(dt_R)$ bits.

4.5.7 Main Result

Let (A, B) be a linear signature protocol, and let all related notations be as in Section 4.5.2. By Proposition 4.1, we may assume that $t_B \geq 2t_A$. The claimed signature scheme is defined as follows.

Initialization: Let a security parameter k be given, and a positive integer d such that $\log d$ is polynomially bounded as a function of k. The signer runs $G(1^k)$ twice and obtains two instances (x_0, w_0) and (x_1, w_1) in R. Next, ρ_0 is selected at random from $\{0,1\}^{t_R}$. The string

a_0 is computed as $a_0 \leftarrow a(x_0, w_0, \rho_0)$. The public key (x_0, x_1, a_0) is placed in the signer's public directory. The strings w_0, w_1 and ρ_0 are private input to the signer. Assume without loss of generality that the signature bound B_σ is equal to $2^{d(k)}$ and that $d(k) = O(\log k)$. The algorithm DFS gets as input ρ_0 and the integer d (dropping its possible dependence on k).

Signing: The message space $\mathcal{M}$ is set to $\{0,1\}^{t_B}$. Suppose this is the i-th signature ($i = 1 \dots 2^d$), and let $m \in \mathcal{M}$ be the message to be signed. First, the signer puts

$$(\rho_0, \rho_1, \rho'_1, \dots, \rho_{d-1}, \rho'_{d-1}, \rho_d, \rho'_d) \leftarrow DFS(i).$$

The signer performs the following computations.[2]

1. $\sigma \leftarrow \{0,1\}^{t_B}$, $b \leftarrow a(x_1, w_1, \sigma)$, $s \leftarrow z(x_1, w_1, \sigma, m)$,
2. $a_d \leftarrow a(x_0, w_0, \rho_d)$, $z_d \leftarrow z(x_0, w_0, \rho_d, b)$.
3. For $j = 1 \dots d-1$: $a_j \leftarrow a(x_0, w_0, \rho_i)$, $a'_j \leftarrow a(x_0, w_0, \rho'_j)$,
4. For $j = 0 \dots d-1$: $z_j \leftarrow z(x_0, w_0, \rho_j, [a_{j+1}, a'_{j+1}])$, where we use $[a_{j+1}, a'_{j+1}]$ to denote the concatenation of a_{j+1} and a'_{j+1} such that the left part and the right part in the concatenation correspond to the left child and right child of ρ_j in the tree, respectively.

The signature $\sigma(m)$ is now

$$(a_0, a_1, a'_1, z_0, \dots, a_{d-1}, a_d, a'_d, z_{d-1}, ind, z_d, b, s,)$$

where ind is a string of d bits such that the j-th bit indicates how to compose $[a_{j+1}, a'_{j+1}]$ from a_j and a'_j.

Verification: Given the public key (x_0, x_1, a_0), the receiver puts

$$\sigma(m) \leftarrow (A_0, A_1, A'_1, Z_0, \dots, A_{d-1}, A_d, A'_d, Z_{d-1}, ind, Z_d, B, S,),$$

where $A_0 \leftarrow a_0$, and checks whether

1. $\phi(x_1, B, m, S) = accept$,
2. $\phi(x_0, A_d, B, Z_d) = accept$,
3. For $j = 0 \dots d-1$, $\phi(x_0, A_j, [A_{j+1}, A'_{j+1}], Z_j) = accept$.

If all verifications are satisfied, the signature is accepted.

[2]Please note that we make the coinflips to the algorithms $a(\cdot)$ and $z(\cdot)$ explicit as described in Section 2.1.

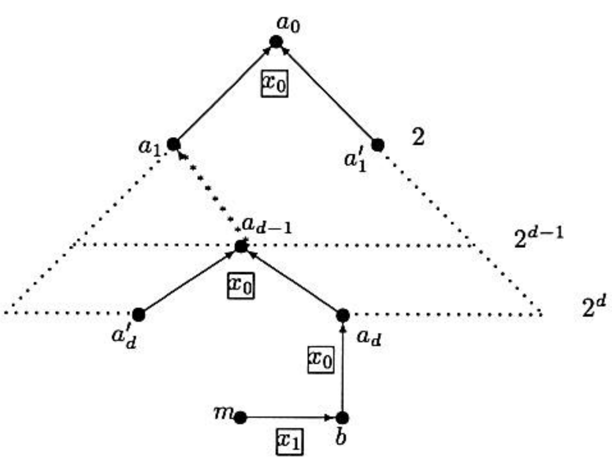

FIGURE 4.2. Secure Signature based on Interactive Protocol

REMARK 4.1 *Storing the a-values instead of the coinflips will reduce the amount of on-line computation needed. But of course, an authentication path can also be computed off-line before the next message to be signed is known to the signer.*

REMARK 4.2 *We develop the authentication tree in depth first fashion to minimize storage. However, breadth-first development of the tree is equally possible. Moreover, our scheme also accommodates the use of the internal nodes for making signatures as is done with the leaves. To this end, the description of DFS and the main result and the proof of security are easily adapted.*

REMARK 4.3 *The result remains valid for general (not specifically linear) signature protocols. However, we may not get the desired efficiency in the general case.*

4.5.8 Proof of Security

The proof of security of the proposed signature scheme is by showing that the existence of a successful attacker of the signature scheme contradicts the collision-intractability of (A, B). To this end, we construct a simulated signer that is indistinguishable from the real signer from the point of view of the attacker. This simulation differs from a real signer in the sense that it is given the witness w for only one of the two instances x. The proof is concluded by the observation that any forgery leads to a collision in (A, B) for at least one of the two instances. By the indistinguishability of the simulation, this is a collision for exactly that instance x for which the simulation was not given a witness, with probability 1/2. Thus an attacker of the signature scheme can be efficiently transformed into an algorithm that computes collisions for (A, B), and we have arrived at the desired contradiction. We now prove the following theorem.

THEOREM 4.2 *Let (A, B) be a linear signature protocol for relation R and generator G. Then the signature scheme constructed from (A, B) is not existentially forgeable under adaptive chosen message attacks. Let k be the security parameter and let B_σ be a bound on the number of signatures to be made. Then the size of each signature is $O(k \log B_\sigma)$ bits, while the public keys has size $O(k)$ bits.*

PROOF. Put $B = 2^d$ for some positive integer d. Let $\mathcal{A}$ denote an attacker who efficiently forges signatures using an adaptive chosen message attack. First, we generate two independent instances $(x_i, w_i) \in R$, $i = 0, 1$, by running the generator G twice. Then we flip a random coin β, and discard the witness $w_{1-\beta}$. The simulated signer receives as input (x_β, w_β) and $x_{1-\beta}$, and is defined as follows (coinflips are implicit in our description). Let M be the special simulator for (A, B).

$\beta = 0$:
First, a_0 is generated as $a_0 \leftarrow a(x_0, w_0)$. The rest of the simulation is as in the description of the signature scheme, except that b and s are computed by $(x_1, b, m, s) \leftarrow M(x_1, m)$.

$\beta = 1$:
In this case, the authentication tree is constructed bottom-up. First, all 2^d b-values are generated by computing $b \leftarrow a(x_1, w_1)$ 2^d times independently. Then for each b-value a leaf a_d is generated together with an authentication value z_d by computing $(x_0, a_d, b, z_d) \leftarrow M(x_0, b)$. Now, given two adjacent leaves a_d, a'_d which should have the same ancestors (suppose that a_d is left from a'_d), their parent a_{d-1} is generated together with the authentication value z_{d-1} by computing $(x_0, a_{d-1}, a_d|a'_d, z_{d-1}) \leftarrow M(x_0, a_d|a'_d)$. This process is inductively repeated until we have a full binary tree of depth d. Let a_0 denote the resulting root. Since in this case w_1 is known to the simulation, it can compute $s \leftarrow z(x_1, w_1, b, m)$ for any of the b-values as generated above and for any m.

Now the attacker $\mathcal{A}$ comes in the play. After the public key (x_0, x_1, a_0) has been made available to $\mathcal{A}$, it is allowed to carry out the adaptive chosen message attack on the simulated signer. From inspection of the simulated signer, it is clear that it can fulfill any of the at most 2^d requests form the attacker. Note that in the case $\beta = 1$, the simulation pre-computed the tree, but only reveals the nodes in the order that a real signer would. Indistinguishability between the cases $\beta = 0$ and $\beta = 1$ follows immediately from Lemma 2.4; the distribution of all a, z, b and s values is exactly the same as when generated by a real signer, due to the properties of the simulator M for (A, B) (see Section 2.4.3).

Next, we assume without loss of generality that the attacker makes exactly 2^d requests to the simulation. Since we are assuming that $\mathcal{A}$ is successful, it will output some forgery

$$(\tilde{a}_0, \tilde{a}_1, \tilde{a}'_1, \tilde{z}_0, \ldots, \tilde{a}_{d-1}, \tilde{a}_d, \tilde{a}'_d, \tilde{z}_{d-1}, ind, \tilde{z}_d, \tilde{b}, \tilde{s}),$$

on some message $\tilde{m}$ whose signature has not been requested from the simulation. Note that we must have $\tilde{a}_0 = a_0$. For convenience, we assume that a verifier in the signature scheme always does a length-check that a signature consists of exactly d values a. This is not necessary for the security of our scheme, but it simplifies some of the arguments to follow. Let T denote the full binary tree of depth d tree consisting of the a values as generated by the simulation, and let H denote the collection of the 2^d signatures output by the simulation.

Now consider the largest integer j such that $\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_j$ is a path in T Since $\tilde{a}_0 = a_0$ and T has depth d, we have $0 \leq j \leq d$. Then, since we have assumed that the simulator output the maximal number of 2^d signatures, there exists $\sigma(m) \in H$ for some $m \neq \tilde{m}$ where

$$\sigma(m) = (a_0, a_1, a'_1, z_0, \ldots, a_{d-1}, a_d, a'_d, z_{d-1}, ind, z_d, b, s)$$

and $a_i = \tilde{a}_i$ for $i = 1 \ldots j$. Suppose that $j < d$, then (by the maximality of j) we have $\tilde{a}_{j+1} \notin \{a_{j+1}, a'_{j+1}\}$. Then $[\tilde{a}_{j+1}, \tilde{a}'_{j+1}] \neq a_{j+1}, a'_{j+1}]$. Thus, since $\tilde{a}_j = a_j$, we immediately have a collision $(x_0, a_j, [a_{j+1}, a'_{j+1}], z_j)$ and $(x_0, a_j, [\tilde{a}_{j+1}, \tilde{a}'_{j+1}], \tilde{z}_j)$. In case $j = d$, we must have that $\tilde{a}_d = a_d$ is a leaf in T. If $b \neq \tilde{b}$, we have a collision (x_0, a_d, b, z_d) and $(x_0, a_d, \tilde{b}, \tilde{z}_d)$. Finally, if $b = \tilde{b}$, we must have a collision (x_1, b, m, s) and $(x_1, b, \tilde{m}, \tilde{s})$.

By the perfectness of the simulation, the probability that we get a collision with respect to $x_{1-\beta}$ is $1/2$. Thus the existence of a successful attacker of the signature schemes contradicts the collision-intractability of the protocol (A, B). The claims about the size of signatures and of the public key follow by inspection of the protocol, and by taking into account Proposition 4.1. □

4.5.9 Example

Let (A, B) be the signature protocol DL^* from Section 2.5.2. Suppose elements from G_q can be encoded using $u = O(|q|)$ bits. We can now carry out the construction of Section 4.5.7. As a toy example, put $d = 3$. Thus, the authentication tree will eventually have 15 nodes, and the signer can make 8 signatures in this small example.

Let t be the smallest integer such that $t|q| \geq 2u$. To set up an instance of the signature scheme, the signer generates two independent instances of (A, B), $(x, w) \equiv ((x_1, w_1), \ldots, (x_t, w_t))$ and $(\overline{x}, \overline{w}) \equiv ((\overline{x}_1, \overline{w}_1), \ldots, (\overline{x}_t, \overline{w}_t))$, with $x_i = g^{w_i}$ and $\overline{x}_i = g^{\overline{w}_i}$ for $i = 1, \ldots, t$. The w_i and $\overline{w}_i$ are chosen at random from $\mathbb{Z}_q$. The root of the authentication tree, a_0, is computed as $a_0 = g^{\rho_0}$, where ρ_0 is chosen at random from $\mathbb{Z}_q$. All 14 other nodes a_i are computed as $a_i = g^{\rho_i}$ for random $\rho_i \in \mathbb{Z}_q$, $i = 1 \ldots 14$. The 8 b-values are computed as $b \leftarrow g^{\sigma}$ for random $\sigma \in \mathbb{Z}_q$. The initialization phase is completed when the public key of the signer, $(x, \overline{x}, a_0)$, is placed in the public directory.

If we number the nodes in the order in which they are generated, the top has number 0, the first level (from left to right) contains number 1 and 2, the second level number 3, 4, 9 and 10, the third level number 5, 6, 7, 8, 11, 12, 13 and 14. So, for instance, as part of the first signature, the signer generates nodes $1, 2, 3, 4, 5, 6$, where a_5 is the first leaf. The second leaf, a_6, is used in the second signature. Now assume that the signer is making the 5-th signature. Note that after the fourth signature, for instance, there is no need to store nodes 3, 4, 5, 6, 7 and 8, since these will not occur on the extended authentication path of any of the forthcoming leaves. So at this point, the tree only consists of nodes 0, 1 and 2. To make the 5-th signature, the signer generates nodes 9, 10, 11 and 12 as described above. Note that node 11 is the 5-th leaf. This gives rise to new values ρ_9, ρ_{10}, ρ_{11} and ρ_{12}. The values ρ_0, ρ_1 and ρ_2 are still in memory. The path to the root 0 from node 11 consists of the nodes 9 and 2, while nodes 1, 10 and 12 are the other children of the nodes on this path.

Let l be the largest integer such that $2^l \leq q$, and let message $m \in \{0,1\}^{t \cdot l}$ be the message to be signed, where $m = m_1 | \ldots | m_d$ and the m_i are l-bitstrings, to be interpreted as members of $[0 \ldots 2^l)$. For instance, a_9 and a_{10} are authenticated by computing z_2 as $z_9 = \rho_2 + \mu_1 \overline{w}_1 \cdots + \mu_t \overline{w}_t$, where $\mu_1 | \cdots | \mu_t = a_9 | a_{10}$. The μ_i are l-bitstrings, to be interpreted as members of $[0, \ldots, 2^l)$. In the same way, a_1 and a_2 are authenticated from a_0, and a_{11} and a_{12} from a_9. Summing up, this results in authentication values z_0, z_2 and z_9. Now b is generated and authenticated by z_{11} (from a_{11}) as above. Similarly, s is computed from σ and m. Finally, z_0, a_1, a_2, z_2, a_9, a_{10}, z_9, a_{11}, a_{12}, z_{11}, b and s are forwarded to the receiver, who performs the obvious verifications.

Previously, the only known way to get a signature scheme provably secure based on discrete log was to use the method from [52] to build a collision intractable hash function and then use Merkle's construction. This would require an exponentiation for each bit processed in the hashing, and moreover we would need as a part of the signature a full preimage under the hash function to authenticate 1 bit. Therefore we would get signatures of length $O(k^2 \log B_\sigma)$ bits and would need $O(k^2 \log B_\sigma)$ exponentiations to make a signature.

4.6 Security Amplifications for Signatures

As a by-product of their investigations into so-called on/off-line signatures, Even, Goldreich and Micali [59] showed that the existence of signature schemes secure against known message attacks implies the existence of schemes secure against adaptively chosen message attacks. Unfortunately, this transformation leads to a rather impractical scheme. We exhibit a similar security amplification, which takes the given scheme to a new signature scheme that is not even existentially forgeable under adaptively chosen message attacks. Additionally, however, our transformation will be practical: The complexity of the resulting scheme is twice that of the original scheme. As argued in Section 4.1, such transformations may be an aid to the design a practical and secure signature schemes.

As in [70], and in many other cryptographic schemes, their approach works with two independently generated instances of the signature scheme S that is given as input. The resulting keys constitute the keys for the instance of $\overline{S}$.

Let m be a message of length n. The first instance is used to authenticate the concatenation of $2n$ bit-strings, chosen uniformly at random by the signer. Bit-wise, the message m is used to select n of these strings which are finally authenticated, one-by-one, using the second instance of S. For each new message, this procedure is repeated. As a result of this bit-wise signing technique, the complexity of the transformed scheme becomes, roughly, the complexity of S times the number of bits that are signed. Therefore the transformation from [59] is not suitable to serve as a basis for security amplifications of practical signature schemes.

Consider a signature scheme that is secure when an attacker just sees signatures on random k bits messages. Our approach is to take a subset $\overline{\mathcal{M}}$ of this message space $\mathcal{M}$ that contains just a negligible fraction of the elements but contains an exponential number of elements. For instance, take the subset of strings where the last $k/2$ bits are set to 0. A signer in the new scheme first generates two independent instances of the original scheme. To sign a message $m \in \overline{\mathcal{M}}$ in the new scheme, the signer computes a random pair $m_1, m_2 \in \mathcal{M}$ such that $m_1 \oplus m_2 = m$, and signs m_1 with respect to the first instance and m_2 with respect to the second instance. The intuition behind the security of this scheme, is that an adaptive attacker cannot get around the random split of the message and will still only see signatures (in the original scheme) on random messages. The idea behind the smaller message space is that it prevents the attacker from making forgeries out of cross-combinations of signatures.

4.6.1 The Transformation

Let S be any signature scheme and let $\mathcal{M}(k)$ denote the message space and $|\mathcal{M}(k)|$ its size corresponding to k. Moreover we assume that $\mathcal{M}(k) = \{0,1\}^{t(k)}$, where $t(k)$ is some non-constant polynomial in k. Let $\overline{\mathcal{M}}(k)$ denote a subset of $\mathcal{M}(k)$ that consists of a negligible[3] large fraction $\rho(k)$ of $\mathcal{M}(k)$. For instance, $\overline{\mathcal{M}}(k)$ could consist of all bit-strings of length t, with the last $t/2$ bits set to zero. Using S as a building block, a new scheme $\overline{S}$ is constructed as follows:

Initialisation
Let the security parameter k be given. To generate an instance of $\overline{S}$, the signer runs G twice, yielding two key-pairs (pk_1, sk_1) and (pk_2, sk_2). The public-key pk for the instance of $\overline{S}$ will be (pk_1, pk_2), and the secret key sk will be (sk_1, sk_2).

Signing
Let $m \in \overline{\mathcal{M}}(k)$ be the message to be signed. The signer chooses a random pair (m_1, m_2), with $m_1, m_2 \in M(k)$, such that $m_1 \oplus m_2 = m$, and computes $\sigma_i(m_i)$ for $i = 1, 2$, where $\sigma_i(m_i)$ denotes a signature in S, with respect to the key-pair (pk_i, sk_i). The signature, $\overline{\sigma}(m)$, in $\overline{S}$ is $(m_1, m_2, \sigma_1(m_1), \sigma_2(m_2))$.

Verification
To verify a signature $\overline{\sigma}(m)$ on $m \in \overline{\mathcal{M}}(k)$ with respect to pk, the receiver checks whether $m_1 \oplus m_2 \overset{?}{=} m$, and whether $\sigma_i(m_i)$ is a valid signature in S with respect to pk_i for $i = 1, 2$.

It is now shown that if S is secure against a known message attack, then $\overline{S}$ is secure against a chosen message attack.

We first need a lemma which says that it is very unlikely that $\sigma_1(m_1)$ and $\sigma_2(m_2')$ corresponding to signatures on two different messages, $m, m' \in \overline{\mathcal{M}}(k)$, can be combined to a valid signature in $\overline{S}$.

Consider the following game involving two players A and B. Player B submits any member $m^1 \in \overline{\mathcal{M}}(k)$ to A, and A returns a random pair (m_1^1, m_2^1), with $m_1^1, m_2^1 \in M(k)$, such that $m_1^1 \oplus m_2^1 = m^1$. They repeat this procedure, say, r times. This results in a sequence

$$(m^1, m_1^1, m_2^1), \ldots, (m^r, m_1^r, m_2^r),$$

such that $m_1^j \oplus m_2^j = m^j$ for $j = 1 \ldots r$. B is allowed to choose the values of m^j adaptively. B wins if he can find a pair (m_1^u, m_2^v) such that $m_1^u \oplus m_2^v \in \overline{\mathcal{M}}(k)$ and $u \neq v$ and $1 \leq u, v \leq r$.

[3]A non-negative function $f : \mathbb{N} \to \mathbb{R}$ is negligible iff $\forall c > 0\, \exists n_0 \in \mathbb{N}\, \forall n \in \mathbb{N} : n > n_0 \Rightarrow f(n) \leq n^{-c}$.

LEMMA 4.1 *In the game described above, B's probability of winning the game is at most $r(r-1)\rho(k)$.*

PROOF. Define for $1 \leq u, v \leq r$ and $u \neq v$, the stochastic variable $X_{u,v} = m_1^u \oplus m_2^v$. The probability that $X_{u,v} \in \overline{\mathcal{M}}(k)$ is clearly fully determined by A's uniform coin flips, and therefore equal to $\rho(k)$. As there are $r(r-1)$ pairs (u, v), B will win with probability at most $r(r-1)\rho(k)$. □

Now consider the signature scheme $\overline{S}$ described above. Let $\mathcal{A}$ be any probabilistic polynomial time algorithm that executes an adaptively chosen message attack on $\overline{S}$, and let $\mathcal{A}$'s signature requests be on messages

$$m^1, \ldots, m^{r(k)} \in \overline{\mathcal{M}}(k),$$

with $r(k)$ polynomially bounded. The signer then returns

$$(\sigma_1(m_1^1), \sigma_2(m_2^1)), \ldots, (\sigma_1(m_1^{r(k)})), \sigma_2(m_2^{r(k)}),$$

as required.

PROPOSITION 4.2 *If the signature scheme S is not existentially forgeable under known message attacks, the attacker $\mathcal{A}$ has only negligible probability of outputting a signature $\tilde{\sigma}(\tilde{m})$ in S, where $\tilde{m} \neq m_i^j$ for $j = 1, \ldots, r(k)$, and $\tilde{\sigma}(\tilde{m})$ is a valid signature with respect to pk_i, with $i = 1$ or $i = 2$.*

PROOF. We use standard simulation techniques. Suppose $\mathcal{A}$'s probability of success is non-negligible (in k). Let a signer A in S, with public key pk, be given. We will use the attacker $\mathcal{A}$ to conduct a successful known message attack on signer A, thus contradicting the assumption on S.

Generate an instance (pk', sk') in S. Choose i at random in $\{1, 2\}$, and put $pk = pk_i$ and $pk' = pk_{3-i}$. Now present the resulting key (pk_1, pk_2) for $\overline{S}$ to the attacker. The signer A with public key pk_i, used as a subroutine in the simulation, will output signatures on randomly chosen messages. More specifically, the simulation works as follows.

1. Receive message $m \in \overline{\mathcal{M}}(k)$ from the attacker.
2. Receive a signature $\sigma_i(m_i)$ from A, where A chooses $m_i \in M(k)$ uniformly at random and i is as defined earlier.
3. Compute $m_{3-i} = m_i \oplus m$ and $\sigma_{3-i}(m_{3-i})$. Forward $\overline{\sigma}(m)$ to the attacker.

As the attacker cannot distinguish this simulation from a true signer in $\overline{S}$, the probability that $\tilde{\sigma}(\tilde{m})$ is a forgery of A's signature is half $\mathcal{A}$'s success probability. This is still non-negligible. □

THEOREM 4.3 *Let S be any signature scheme that is not existentially forgeable under known message attacks. Then the signature scheme $\overline{S}$ is not existentially forgeable under adaptively chosen message attacks.*

Proof. Let $\tilde{m} \in \overline{\mathcal{M}}(k)$ and let $\tilde{\sigma}(\tilde{m}) = (m_1, m_2, \sigma_1(m_1), \sigma_2(m_2))$ be a forgery in $\overline{S}$ on a new message, obtained after an adaptively chosen message attack. By Proposition 4.2, except with negligible probability $\sigma_1(m_1) = \sigma_1(m_1^u)$, and $\sigma_2(m_2) = \sigma_2(m_2^v)$, for some u, v with $1 \leq u, v \leq r(k)$ and $u \neq v$ (notation as in Proposition 4.2). So we must have that $m_1^u \oplus m_2^v = \tilde{m}$. However, by Lemma 4.1, this has only negligible probability. □

4.6.2 Minimality of the Construction

Naor [92] gave a counterexample in case one chooses $(pk_1, sk_1) = (pk_2, sk_2)$ in the transformation. This proves that our construction is minimal in some sense. For the history dependent case (where each new signature has a dependence on previous executions), taking the GMR-scheme [70] and Section 4.4 and applying the proposed transformation, leads to an insecure scheme.

Let f_s be a pseudo-random function [12] with seed s, and let S denote the signature scheme we start from. Suppose S signs messages of size k bits. Let sk be the private input to the signer in S. First we make S into a new signature scheme S' that is also "not existentially forgeable under random message attacks", as follows.

The private input to the signer in S' is sk, s. Given a k-bit message m, let m' denote m without its least significant bit m_k. In S', the signer now sends $S(m), f_s(m') \oplus (sk \cdot m_k)$. If this scheme S' is taken as input to the proposed transformation, the attacker just asks for a signature on the message 0...01. From the resulting signature, the attacker is able to compute the private input sk.

4.7 Secure and Practical RSA-based Signatures

For most digital signature schemes used in practice, such as ISO9796/RSA or DSA, it has only been shown that certain plausible cryptographic assumptions, such as the difficulty of factoring integers, computing discrete logarithms or the collision-intractability of certain hash-functions are necessary for the security of the scheme, while their sufficiency is, strictly speaking, an open question.

A clear advantage of such schemes over many signature schemes with security proven relative to such common cryptographic assumptions, is their efficiency: as a result of their relatively weak requirements regarding computation, bandwidth and storage, these schemes have so far beaten proven secure schemes in practice.

Our aim is to contribute to the bridging of the gap that seems to exist between the theory and practice of digital signature schemes. We present a digital signature that offers *both* proven security and practical value.

More precisely, under an appropriate assumption about RSA, the scheme is proven to be not existentially forgeable under adaptively chosen message attacks. We also identify some applications where our scheme can be conveniently implemented using dedicated smartcards that are available today.

Our exposition is organized as follows. In Section 4.7.1, we outline the technical ideas behind our design. The formal presentation of the scheme can be found in Section 4.7.2. The latter section left open the choice of some parameters. This is resolved in Section 4.7.3, which is followed by a discussion of the performance of our scheme in Section 4.7.4. The proof of security is given in Section 4.7.5. In Section 4.7.6, we give optimizations of the proposed scheme that cut the storage requirements even further.

4.7.1 Basic Ideas

Conceptually, our signature scheme may be viewed as a cross between [70] and [57], together with modifications enabling their synthesis. These modifications are facilitated by our signature protocols RSA^* from Section 2.5.2. Recall that these these provide an important example of a special one-way group isomorphism. Let l and d be integers. In [57], all players in the signature scheme must have access to two lists. The first list contains l primes. Depending on the particular RSA-assumption one wishes to make, these could be, for instance, the first l odd primes, or l random primes. The second list consists of l random k-bit strings. Here, k is a security parameter and l is an integer with $l \geq k$. Our first objective is to remove the necessity of the list of random numbers.

In [57], the system constants are as follows. Let L denote the list of primes $\{p_0, \ldots, p_{l-1}\}$, L' the list of l random l-bit strings $\{x_0, \ldots, x_{l-1}\}$ and let a denote a random l-bit string, to be used as the root of all authentication trees. Let a signer be given an RSA-modulus n together with its factorization. The public key consists of n and y_{root}. The latter is to be the root of an l-ary authentication tree of depth d. The factorization of n is private input for the signer.

The "basic authentication step" in [57] is

$$y \leftarrow (\alpha \cdot \prod_{i=0}^{l-1} x_i^{\beta_i})^{\frac{1}{p_j}} \bmod n$$

where α is an already authenticated value, $\beta = \beta_0 | \cdots | \beta_{l-1}$ is an l-bit string to be authenticated, and p_j is a prime from the list L that has not been used before in connection with α. Instead, our basic authentication step is

$$y \leftarrow (\alpha \cdot h^{\beta})^{\frac{1}{v_j}} \bmod n,$$

where h is a member of $\mathbb{Z}_n^*$ and part of the signer's public key. Furthermore, e_j is the smallest integer such that $v_j \equiv p_j^{e_j} > n$. Here the values that can be authenticated are elements of $\mathbb{Z}_n^*$. This removes the list L' and the condition that $l \geq k$. However, implementing this idea only results in a scheme that we can prove secure against random message attacks. Such a scheme can be efficiently transformed to a scheme that is secure against active attacks, as is desired here, by means of a technique described in [44]. The loss of efficiency is a factor of two (twice as much computation, signature size twice as large). But we can do better in this case, if we add one prime q with a special purpose to the list: it is only used when a message m, agreed upon between the signer and a receiver, is to be authenticated, as follows

$$z \leftarrow (\alpha \cdot h^m)^{\frac{1}{w}} \bmod n,$$

where α is a "freshly" generated leaf in the authentication tree and e is the smallest integer such that $w \equiv q^e > n$. This relates to the idea [70] of applying sufficiently independent functions to the actual signing process and the construction of an authentication tree, respectively.

To minimize the storage needed for the list of primes, we can take L to consist of l consecutive primes. Then, only the first prime and all consecutive differences are stored. In Section 4.7.6, two other techniques are given for further improvements of the efficiency of the scheme.

4.7.2 Description of the Scheme

In a preprocessing-phase, a security parameter k is determined, as well as integers l and d. Next, a list L consisting of $l+1$ distinct primes is generated by invoking an algorithm $H(1^k, 1^l)$, say $L = \{q, p_0, \ldots, p_{l-1}\}$. Ways of choosing H are discussed in the next section.

Furthermore, we assume that we are given a probabilistic polynomial time generator G that, on input 1^k, outputs a triple (n, r, s), where r and s are primes and $n = r \cdot s$ is a k bits integer. It is assumed that G is defined such that it is infeasible to factor n, when only n as generated by G is given as input. Finally, we must have that q and the p_i are co-prime to $\phi(n)$. Given n and L, define e as the smallest integer such that $q^e > n$ and e_i as the smallest integer such that $p_i^{e_i} > n$ for $i = 0 \ldots l-1$. In the following, w denotes q^e and v_i denotes $p_i^{e_i}$, for $i = 0 \ldots l-1$.

We start with an informal overview of the scheme. The signer has as public key an RSA-modulus n, $h \in \mathbb{Z}_n^*$ and $x_0 \in \mathbb{Z}_n^*$. Here, n is generated by $G(1^k)$ and h and x_0 are chosen at random from $\mathbb{Z}_n^*$ by the signer. In a possible variation of the scheme, x_0 and h are chosen mutually at random and are the same for all signers. In any case, h and x_0 must be chosen at random to avoid weak keys.

As always, his knowledge of the factorization (r, s) of n enables the signer to compute $X^{\frac{1}{u}} \bmod n$ for any $X \in \mathbb{Z}_n^*$ and any integer u such that $\gcd(u, (r-1)(s-1)) = 1$. The public key consists of the triple (n, h, x_0). The factorization of n is private input to the signer.

The algorithm DFS, which is used in the formal description of our scheme, gradually develops a full l-ary tree of depth d by selecting the nodes at random from $\mathbb{Z}_n^*$. The tree is constructed in depth-first fashion. Although not explicitly given as input to DFS, it is assumed that it has access to l, d, x_0 and n. The value x_0 serves as the root of the tree. The i-th time DFS is invoked, denoted DFS(i) ($i = 1 \ldots l^d$), it creates a path to a new leaf x_d and outputs this path, say, $x_1, \ldots, x_d$ (the root x_0 being understood). This sequence is ordered such that x_{j-1} is the parent of x_j ($j = 1 \ldots d$).

Furthermore, for each node x_j in this sequence, DFS(i) also outputs an indicator i_j ($j = 1 \ldots d$) in such a way that i_j is assigned to x_j if and only if x_j is the i_j-th child of x_{j-1}. The amount of storage needed for this procedure (apart from l, d, x_0 and n) does not exceed the amount of storage needed for $d-1$ pairs consisting of a node and an indicator.

By invoking DFS, the signer gradually constructs, in a depth first fashion, an l-ary authentication tree with depth d: each time a new signature is required he constructs a path to a new leaf. All nodes x are members of $\mathbb{Z}_n^*$, given by their smallest non-negative representative modulo n. The message space is equal to the set $\{0,1\}^k$, which we will also identify with the set of non-negative integers smaller than 2^k.

In Figure 4.3, the signer is making his i-th signature, on a message $m \in \mathbb{Z}_n^*$. So, in particular x_d is the i-th leaf he reaches. The part of the tree on the right side of the path $x_0, \ldots, x_{d-1}, x_d$ is not yet constructed. Since x_1 happens to be the i_1-st child of x_0, the signer authenticates x_1 with respect to the prime p_{i_1} by computing $y_1 \leftarrow (x_0 \cdot h^{x_1})^{\frac{1}{v_{i_1}}} \bmod n$. Similar rules apply to the authentication of the remaining nodes in this path. In particular, it so happens to be in our example that x_d is the i_d-th child of x_{d-1}. Thus x_d is authenticated by computing $y_d \leftarrow (x_{d-1} \cdot h^{x_d})^{\frac{1}{v_{i_d}}} \bmod n$. Finally, the message m is signed by computing $z \leftarrow (x_d \cdot h^m)^{\frac{1}{w}} \bmod n$. Notice that the prime q is only used when the "actual signature" is computed, while the other primes in the list L are used exclusively in the process of constructing the authentication tree. The signature on m consists of the y_j and indicators i_j, ($j = 1 \ldots d$) and z.

Concerning the storage needed for the signer, notice that the part of the tree left from the path $(x_0, \ldots, x_{d-1}, x_d)$ can be deleted. Actually, x_d itself can also be removed. In order to carry on with the depth-first construction of the tree, it is sufficient to store $x_0, \ldots, x_{d-1}$ and the indicators to their parents. This storage amounts to at most $(d-1)(k + \log l)$ bits (the root x_0 is part of the public key).

A receiver of this signature gets only the message m, authentication values y_j, the indicators i_j $(j = 1 \ldots d)$ and z. So, what about the nodes? These are re-computed as follows. On input of the public key, the list L, m and z he recomputes x_d as $x_d \leftarrow z^w \cdot h^{-m} \bmod n$. Recursively, the receiver re-computes x_{j-1} from x_j, y_j and i_j in a similar fashion $(j = d \ldots 1)$. The last node x_0 he thus computes should be equal to the actual x_0, which is part of the public key. If so, the signature is accepted. We point out that in many tree-structured signature schemes, it is sufficient to send the authentication values and have the verifier re-compute the nodes, instead of defining these as part of the signature. It is easily seen why this does not affect the security at all: briefly, if the verifications in the "reduced" scheme hold, one gets a signature in the original scheme (on the same message, of course) by simply incorporating the re-computed nodes. More formally, the description of the new signature scheme is as follows.

Preprocessing:
A security parameter k, integers l and d are determined. Next, the system constant $L = \{q, p_0, \ldots, p_{l-1}\}$ consisting of $l + 1$ distinct primes is generated by invoking $H(1^k, 1^l)$. Define e as the smallest integer such that $w \equiv q^e > n$, and e_i as the smallest integer such that $v_i \equiv p_i^{e_i} > n$, for $i = 0 \ldots l - 1$. For possible choices of H, see Sections 4.7.3 and 4.7.6.

Initialization:
The signer runs $G(1^k)$ and obtains a triple (n, r, s) such that q and the p_i are co-prime to $\phi(n)$. Next, he chooses h and x_0 at random in $\mathbb{Z}_n^*$. His public key pk is now the pair (n, h, x_0), while his secret key sk consists of the pair (r, s).

Signing:
Let m be a k bit message. The i-th signature, where $1 \leq i \leq l^d$, is computed as follows. First, the signer runs the algorithm DFS to get $(x_1, i_1, \ldots, x_d, i_d) \leftarrow \mathrm{DFS}(i)$. Next, he computes (for $j = 1 \ldots d$) $y_j \leftarrow (x_{j-1} \cdot h^{x_j})^{\frac{1}{v_{i_j}}} \bmod n$. Finally, he computes $z \leftarrow (x_d \cdot h^m)^{\frac{1}{w}} \bmod n$. The signature σ on m consists of the values $z, y_1, i_1, \ldots, y_d, i_d$.

Verification: Verification is done as follows. The receiver of a signature puts

$$\sigma = (Z, Y_1, i_1 \ldots, Y_d, i_d),$$

and, on input of $pk = (n, h, x_0)$, m and σ, he computes $X_d \leftarrow Z^w \cdot h^{-m} \bmod n$. Finally, he computes $X_{j-1} \leftarrow Y_j^{v_{i_j}} \cdot h^{-X_j} \bmod n$ $(j = d \ldots 1)$. If $X_0 \equiv x_0 \bmod n$, the signature is accepted.

REMARK 4.4 *For a more convenient exposition of the scheme, we have chosen to let the signer only use the leaves for signing. However, the scheme is*

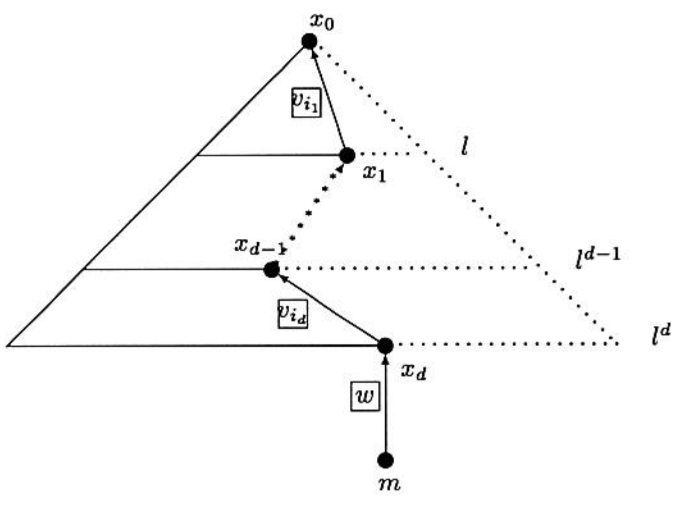

FIGURE 4.3. Secure and Practical RSA Signature

easily adapted so as to allow for a more extensive use of the authentication tree. In this modified scheme, each freshly constructed node can immediately be used for making a signature. The proof of security is easily adapted to fit with this modification.

4.7.3 Generating the List of Primes L

In order to minimize the storage needed for the system constants (L, k, l and d), it is convenient to set L to any $l+1$ consecutive primes greater than 2. In this case, only the first prime, the differences between consecutive primes and the exponents e and e_i are stored. As an example, one could take L to consist of the first $l+1$ (odd) primes.

It must be emphasized, however, that the correctness of the scheme is independent of the particular ways of generating L. Also, the proof of security is not affected by such choices. What is dependent on the choice of L, is the particular assumption we have to make about RSA-inversion. See Section 4.7.5.

4.7.4 Performance of the Scheme

A signer can make at least l^d signatures (see also Remark 4.4) such that the size of each signature does not exceed $(d+1)k$ bits (neglecting the $d \log l$ bits needed to indicate the path). A public key has size $3k$ bits.

Concerning the amount of computation needed, signing requires two full RSA-exponentiations and one modular multiplication on-line. A path to the current leaf can be authenticated by pre-processing, using $2d$ full-RSA exponentiations and d modular multiplications. A receiver of a signature will have to perform $2(d+1)$ full RSA-exponentiations and $d+1$ modular multiplications.

For the gradual depth-first construction of the authentication tree, the signer stores at most $(d-1)(k+\log l)$ bits at any time. Secure storage in the strongest sense (storage not accessible or alterable by "the outside world") is only needed for the secret key (k bits) and the relevant nodes of the latest path in the tree, which is at most $(d-1)k$ bits. The public list L only has to be securely stored in a weaker sense: the signer must have certainty that L is authentic.

Please note, however, that by storing the last signature up to the leaf, including all preceding authentication values, the generation of the next signature will cost just on-line RSA-operation in general (observe that this applies to each l consecutive signatures that having the same authentication path). Thus, if l is large, storage/computation trade-offs of this kind may reduce the required computation for the signer to virtually only one RSA-computation.

4.7.5 Proof of Security

The proof of security works for any choice of the list L. However, the particular assumption we make about the difficulty of RSA-inversion depends on this very choice in the following way. We require the following of the algorithm H.

ASSUMPTION 4.1 *Let k be a security parameter and let l be of polynomial size in k. Let L be generated by $H(1^k, 1^l)$ and let n be an RSA-modulus as generated by $G(1^k)$ and let x be a random member of $\mathbb{Z}_n^*$. Then there is no probabilistic polynomial time algorithm that has non-negligible probability of computing $x^{\frac{1}{\alpha}} \bmod n$ with $\alpha \in L$, on input L, n and x.*

Under this assumption, we can prove that the signature scheme is not existentially forgeable under adaptively chosen message attacks.

THEOREM 4.4 *Under Assumption 4.1, the signature scheme presented in Section 4.7.2 is not existentially forgeable under adaptively chosen message attacks.*

PROOF. We are given integers l and d, a list $L = \{q, p_0, \ldots, p_{l-1}\}$ consisting of $l+1$ distinct primes and an RSA-modulus n. Let w and v_i be defined as in Section 4.7.2, for $i = 0 \ldots l-1$. We assume that n is generated according to $G(1^k)$, but we are not given the factorization. Also, we assume that q and the p_i are co-prime to $\phi(n)$ and that L is generated according to $H(1^k, 1^l)$. The proof is by contradiction. We show that existence of a successful attacker implies that we can compute $X^{\frac{1}{\alpha}} \bmod n$, given a random $\alpha \in L$, and a random $X \in \mathbb{Z}_n^*$. Which contradicts Assumption 4.1.

Let $\alpha \in L$. First, we show that we can set up a "simulated" signer, who as input $h \in \mathbb{Z}_n^*$ and $h^{\frac{1}{\beta}} \bmod n$ for all β in L different from α, but is yet indistinguishable from a signer who proceeds as in Section 4.7.2 after he is given h, n *and* its factorization. To this end, we consider two cases separately and focus mainly on the differences with Section 4.7.2. Finally, we run the attacker against this simulated signer and obtain the desired contradiction.

Technically, the simulation runs as follows. In case $\alpha = w$, the root x_0 is computed as $x_0 \leftarrow a_0^{v_0 \cdots v_{l-1}} \bmod n$, for randomly chosen a_0 from $\mathbb{Z}_n^*$. The value a_0 is stored. All nodes x, excluding the leaves, are computed as $x \leftarrow a^{v_0 \cdots v_{l-1}} \bmod n$, where a is chosen at random from $\mathbb{Z}_n^*$. The value a is stored. If any x is the f-th child of his parent $x_* = a_*^{v_0 \cdots v_{l-1}} \bmod n$, the authentication value y is computed as $y \leftarrow a_*^{v_0 \cdots v_{f-1} \cdots v_{f+1} \cdots v_{l-1}} \cdot (h^{\frac{1}{v_f}})^x \bmod n$. After the i-th signature on a message m, the i-th leaf x is computed as $x \leftarrow a^w \cdot h^{-m} \bmod n$ where a is chosen at random from $\mathbb{Z}_n^*$. Next, the simulated signer reveals the path to the i-th leaf, together with all authentication values, and the authentication value $z = a$ of the message m.

In case $\alpha \neq w$, say, $\alpha = v_j$, the authentication tree has to be constructed from the bottom up. We first show how this is done for $d = 1$. We select the j-th child at x at random from $\mathbb{Z}_n^*$. The parent x_* is then computed as $x_* \leftarrow b^{v_0 \cdots v_{l-1}} h^{-x} \bmod n$, where b is chosen at random from $\mathbb{Z}_n^*$. The value b is stored. The authentication value y of x is computed as $y \leftarrow b^{v_0 \cdots v_{j-1} v_{j+1} \cdots v_{l-1}} \bmod n$. Finally, the remaining $l-1$ children of x_* are selected at random from $\mathbb{Z}_n^*$. Let x' be the f-th child ($f \neq j$). Then its authentication value y' is computed as $y' \leftarrow b^{v_0 \cdots v_{f-1} v_{f+1} \cdots v_{l-1}} \cdot (h^{\frac{1}{v_f}})^{x'-x} \bmod n$. When we have constructed $l-1$ other such trees with $d = 1$, the same procedure can be used to combine them into a tree with $d = 2$, by letting the roots play the role of the leaves as above. By induction, we can build an l-ary tree of any depth d.

One choice has been left open in the present case. The leaves x of the target tree of depth d must be chosen as $x \leftarrow b^w \bmod n$, for random b in $\mathbb{Z}_n^*$. With the i-th signature request, the simulated signer can reveal the path to the i-th leaf, together with all authentication values, and the authentication value $z \leftarrow b \cdot (h^{\frac{1}{w}})^m \bmod n$.

It is clear that in both cases each node in the tree has a uniform distribution and is independent of anything else. All other values follow deterministically. Thus this simulation cannot be distinguished from the real signer.

In the next step of our proof, we run the attacker against the simulated signer and show that we can compute $X^{\frac{1}{\alpha}} \bmod n$, for random $\alpha \in L$, and a random $X \in \mathbb{Z}_n^*$. Here, we have essentially the same success-probability as the attacker. Recall that n and L were generated by G and H respectively.

We proceed as follows. We choose a random $\alpha \in L$, a random $X \in \mathbb{Z}_n^*$ and a random ρ from $\mathbb{Z}_n^*$. Put $h \leftarrow X^{\prod_{\beta \in L/\{\alpha\}} \beta} \cdot \rho^{\prod_{\beta \in L} \beta} \bmod n$. Next we feed L, n, h, and $h^{\frac{1}{\beta}} \bmod n$ for all β in L different from α to the simulated signer and run the simulation (note that h is also distributed as in "real life") Next, we run the attacker against this simulator. Assume that after l^d calls to the simulated signer, the attacker outputs a forgery:

$$\tilde{m}, \tilde{z}, x_0, \tilde{x}_1, i_1, \tilde{y}_1, \ldots, \tilde{x}_d, i_d, \tilde{y}_d.$$

This is a signature[4] on a message $\tilde{m}$ that has not been signed by the simulator in the course of the attack. Now, let T denote the full-tree of depth d and branching l that the simulated signer has output in the course of the attack. Define j to be the largest integer such that $x_0, \tilde{x}_1, i_1, \ldots, \tilde{x}_j, i_j$ is a path in T.

If $j = d$, then $\tilde{x}_d$ is a leaf. So, there exists a signature

$$m, z, x_1, i_1, y_1, \ldots, x_d, i_d, y_d,$$

output by the simulated signer, such that $\tilde{x_d} = x_d$. By the assumption on $\tilde{m}$, we have $\tilde{m} \neq m$. So, we have

$$(y_d \cdot \tilde{y_d}^{-1})^w = h^{m-\tilde{m}} \bmod n$$

Assume without loss of generality that $m > \tilde{m}$. Now, since the parameters are chosen such that any message is numerically less than $w = q^e$, we can extract $h^{1/q} \bmod n$ from this information as follows: Put $m - \tilde{m} = a \cdot q^b$, where $(a, q) = 1$ and $0 \leq b \leq e - 1$. Let integers f, i be such that $a \cdot f = 1 + i \cdot q^{e-b}$. Then $h^{1/q} = (y_d^f \cdot \tilde{y_d}^{-f} \cdot h^{-i})^{q^{e-b-1}}$.

If, on the other hand, $j < d$, then $\tilde{x}_j$ is a node in T at depth j and $\tilde{x}_{j+1}$ is not a child of $\tilde{x}_j$ in T. Let x_{j+1} denote the i_{j+1}-th child of $\tilde{x}_j$ in T. Then clearly, by assumption on j, $x_{j+1} \neq \tilde{x}_{j+1}$. Thus,

$$(y_{j+1} \cdot \tilde{y}_{j+1}^{-1})^{v_{i_{j+1}}} \equiv h^{x_{j+1} - \tilde{x}_{j+1}} \bmod n,$$

with $x_{j+1} - \tilde{x}_{j+1} \neq 0 \bmod v_{i_{j+1}}$. From this value, $h^{\overline{\frac{1}{p_{i_{j+1}}}}} \bmod n$ is extracted as above in the case $j = d$. We conclude that the forgery allows us to compute $h^{\frac{1}{\alpha}} \bmod n$ for some $\alpha \in L$. By the construction of h, it follows, by

[4]In the verification, the receiver of the signature checks if the signature consists of d nodes. We can remove this "length-check"-condition at the expense of a slightly more technical proof than presented here.

the same calculations as above, that we can efficiently derive $X^{\frac{1}{\alpha}} \bmod n$ from this value.

From the perfectness of the simulation the probability that $\alpha = \beta$ is $1/(l+1)$. Thus, if the attacker has non-negligible success- probability, then we can extract random α-th roots also with non-negligible probability, for $\alpha \in L$. □

Note that if a signer deviating from the signer's algorithm, should deliberately compute two messages that have the same signature, a receiver can easily compute a multiple of the order of h, which may allow that receiver to forge or even factor the signer's modulus.

In the Section 4.7.3 we have suggested to make a particular choice that minimizes the storage of L, namely of having L consist of $l+1$ consecutive primes. Furthermore, for reasons of simplicity, we have suggested that these are the first $l+1$ primes of size $k+1$ bits.

4.7.6 Optimizations

In this section, we describe a number of provably secure methods for decreasing the required size of the exponents in the list L (See also Section 4.7.1).

Using Multiple Values of h

In this variation, the signer generates two values h as described in Section 4.7.2, h_1 and h_2. Let β be some k bits string that has to be authenticated. The signer splits β into two blocks β_1 and β_2 of size $k/2$ bits each and computes the authentication value for β as follows.

$$y \leftarrow (\alpha \cdot h_1^{\beta_1} h_2^{\beta_2})^{\frac{1}{p}} \bmod n,$$

for some appropriate exponent p and node α. This cuts the required size of the exponents by a factor of two. The expenses are an increase of the size of the public key by k bits. As noted before, the value of h may be chosen mutually at random between the signers. This also holds for this method, and as such it would mean an increase of k bits of the system constant. This method preserves the security properties of the scheme, and can be used in conjunction with the other methods presented.

Using a Hash-Function

Let $\mathcal{H}$ be a collision-resistant hash-function that maps arbitrary sized input strings to strings of size $k_* << k$. All values that need to be authenticated in the signature scheme (nodes in the tree and the messages), are to be hashed down to k_* bits first. Also, the root of the authentication tree as part of the public key, can be replaced by a hash of that root.

The effect is that the required size of the exponents is now k_* bits instead of k bits. The security statement now also requires that $\mathcal{H}$ is collision-resistant. This method can be used in conjunction with any of the other methods presented here.

Nederlandse Samenvatting

Van oudsher hield de cryptologie zich voornamelijk bezig met het ontwikkelen van "geheimschriften" (cryptografie) en het bestuderen van de betrouwbaarheid daarvan (cryptanalyse). Na WO II, vooral onder invloed van Claude Shannon's informatie theorie [109, 110], is de cryptologie een op mathematische leest geschoeide wetenschap geworden. Een tweede grote doorbraak vond plaats in 1976 met de publicatie [56] van het artikel *New Directions in Cryptography* door Whitfield Diffie en Martin Hellman. Zij formuleerden het idee van *public key cryptografie.* Indachtig aan de public key cryptografie zoals die zich nadien heeft ontwikkeld, heeft Oded Goldreich [75] het vakgebied als volgt omschreven:

"*Cryptografie* kan men beschouwen als de wetenschap die zich bezighoudt met het beperken van de effecten die teweeg gebracht kunnen worden door oneerlijke dan wel zich misdragende partijen".

Indien er bij de lezer zich een vergelijking met de rechtspraktijk opdringt, wijs ik er op dat de cryptografie beoogt het succesvol ontplooien van zulke bezigheden op voorhand onmogelijk dan wel ondoenlijk te maken.

Strikt genomen hebben de begrippen "cryptografie", "cryptanalyse" en "cryptologie" nog steeds betrekking op repectievelijk ontwerp, studie van veiligheid en de combinatie daarvan (zij het nu in een veel bredere context), maar voor het gemak zal ik hier "cryptografie" als de verzamelnaam nemen. Belangerijke begrippen in de cryptografie zijn *authenticiteit* en *geheimhouding* van *informatie.* Het laatste begrip heeft hier een zeer ruime betekenis: gegevens van welke inhoudelijke aard dan ook, zoals die over een computer-netwerk verzonden en ontvangen kunnen worden, worden als "informatie" aangemerkt. In deze beschouwing wordt een computer-netwerk opgevat als een collectie van computers of "processoren" die afzonderlijk kunnen functioneren. Communicatie, dat wil zeggen, uitwisseling van informatie, wordt mogelijk gemaakt door een netwerk van verbindingen tussen deze processoren.

Laten we aannemen dat dit netwerk volledig is, dat wil zeggen, tussen elk tweetal processoren bestaat er een verbinding. Stel dat een zekere processor A een "bericht" m ontvangt, dat wil zeggen, een pakketje informatie gezonden over de verbinding tussen processor A en een andere processor B. Processor A beschouwt m als *authentiek*, indien A met zekerheid kan vaststellen dat m inderdaad van B afkomstig is. Indien de verbindigen tussen processoren op generlei wijze beveiligd zijn, is het onjuist om authenticiteit vast te stellen op grond van de "fysieke" oorsprong van het bericht: de verbinding tussen A en B. Immers, het bericht zou ook afkomstig kunnen zijn van een mogelijk kwaadwillende partij C die zich toegang heeft verschaft tot de onbeveiligde verbinding tussen A en B, en een bericht aldaar heeft ingevoerd en verzonden, als ware dit afkomstig van B. Laten we nu aannemen, afgezien van de vraag of A de authenticiteit van m heeft vastgesteld, dat B het bericht inderdaad heeft verzonden, en wel als "klare tekst". Dan mogen we aannemen dat partij C het bericht had kunnen afluisteren. Van *geheimhouding* van m, dat wil zeggen, de inhoud van het bericht is slechts bekend aan zender en ontvanger, kan dan a priori geen sprake zijn.

De *public key* cryptografie, waarvan de principes in 1976 door Diffie en Hellman zijn geformuleerd, biedt een elegante wiskundige oplossing voor het authenticiteits- en het geheimhoudings-probleem. Voordien waren er reeds wiskundige methoden om authenticiteit en geheimhouding te waarborgen, in feite gebaseerd op bilaterale "geheime afspraken" tussen elk tweetal partijen die onderlinge veilige communicatie nodig achten. Deze zal ik eerst gedeeltelijk toelichten.

Een in de praktijk nog steeds gebruikt voorbeeld van een dergelijke "vercijferingsmethode" die geheime communicatie mogelijk maakt, is de *Data Encryption Standard* (kortweg DES), die in het begin van de jaren zeventig door IBM is ontwikkeld. Het bestaat uit een publiekelijk bekend vercijferings- en ontcijferings-algoritme (een algoritme is een berekeningsvoorschrift). Elk tweetal partijen in het netwerk kiest samen een random (willekeurige) geheime sleutel. Hiertoe dienen zij, voor de duur van de totstandkoming van deze geheime, gedeelde sleutels, over een veilig kanaal te beschikken, bijvoorbeeld door gebruik te maken van een betrouwbare koerier, verzegelde enveloppen of anderzijds fysiek beveiligde kanalen. Deze geheime sleutel stelt een dusdanig grote hoeveelheid random informatie voor, dat het juist gokken ervan door een derde partij praktisch gezien geen kans van slagen heeft. Door de grootte van de sleutel is het tevens praktisch beschouwd onmogelijk dat dezelfde sleutel gekozen zal worden door meer dan een tweetal partijen. Een partij A versleutelt een klare tekst bericht m bestemd voor B door de functie DES te berekenen op m *en* hun gedeelde geheime sleutel $k(A,B)$. De vercijfering c ontvangen hebbende, roept B de functie DES aan, met als invoer $k(A,B)$ en c, om de klare tekst m te verkrijgen. De veilige werking ervan berust op de grootte van de sleutels en op het feit dat het DES-algoritme zo ontworpen is dat, hoewel vercijfering en ontcijfering gegeven de sleutel snel uitgevoerd kunnen worden, het "kraken" van vercijferingen zonder wetenschap van de geheime sleutel waarmee de onderliggende klare tekst is versleuteld, praktisch gezien ondoenlijk is. Op DES en andere zogeheten *common key* cryptografische methoden kunnen ook *message authenticity codes* gebaseerd worden als oplossing van het authenticiteits-probleem.

De *public key cryptografie* van Diffie en Hellman vindt zijn oorsprong in hun pogingen cryptografische systemen te ontwerpen die geheime afspraken tussen elk tweetal partijen (zoals noodzakelijk in de common key cryptografie) overbodig maken: niet alleen als een besparing op de benodigde hoeveelheid werk om het systeem op te zetten, maar vooral ook omdat de fase waarin deze afspraken gemaakt worden grote veiligheidsrisico's met zich mee kan brengen. Immers, in die fase beschikken de partijen nog niet over een betrouwbare cryptografische methode.

De sleutel tot de oplossing wordt gegeven door zogeheten *trapdoor one-way functies*; deze functies zijn eenvoudig te evalueren voor elk argument uit hun domein, maar het berekenen van de inverse van deze functies op een willekeurige waarde uit hun beeld is ondoenlijk, *tenzij* men de trapdoor kent, dat wil zeggen, de geheime sleutel. Als er voldoende veel van zulke trapdoor one-way functies bestaan, zullen in de visie van Diffie en Hellman alle partijen *afzonderlijk* een willekeurige functie met corresponderende trapdoor kiezen. Laten we zeggen dat partij A de functie f_A gekozen heeft met s_A als trapdoor. Dan plaatst A een beschrijving van f_A, A's *public key*, in een publiekelijk toegankelijk bestand, dat op een betrouwbare manier bijgehouden wordt. A bewaart s_A als geheime sleutel. Wanneer B een geheim bericht m naar A wil verzenden, berekent B, na A's public key in het bestand opgezocht te hebben, de vercijfering c als $f_A(m)$, en verstuurt het over de (onveilige) verbinding tussen A en B. De eigenschappen van de gebruikte functies garanderen dan dat alleen A bericht c kan ontcijferen, en wel door $m = f^{-1}(c)$ te berekenen met gebruikmaking van de geheime sleutel s_A. Tot zover het geheimhouding-probleem.

Een andere zeer belangrijke bijdrage van Diffie en Hellman is de *digital signature*: de digitale equivalent van de handgeschreven handtekening. Een handtekening moet aan twee eisen voldoen: de handtekening die een bericht vergezeld, moet de authenticiteit van het bericht waarborgen, en verder moet deze authenticiteit door elke partij te controleren zijn. Nu wordt aan de laatste eis in alledaagse pen-en-papier praktijken, in de meest strikte zin beschouwd, niet immer voldaan. Digitale handtekeningen daarentegen, voldoen aantoonbaar aan beide eigenschappen. Een digitale handtekening systeem werkt eveneens volgens het public key principe. Laat f een functie zijn waarvoor het ondoenlijk is om deze te evalueren op enig argument uit zijn domein, *tenzij* men de beschikking heeft over een geheime trapdoor s. Laat de handtekening op een bericht m de waarde $\sigma = f(s, m)$ zijn en laat g een functie zijn die voor een gegeven σ, m en f bepaalt of σ inderdaad het f-beeld van m is. Dan bestaat de verificatie van een handtekening uit het toepassen van g op σ, m en f. Gegeven een grote verzameling van zulke functies f en g, kiest iedere partij A een willekeurig paar (f, g) met bijhorende trapdoor s. De public key van A bestaat dan uit f en g, terwijl de trapdoor s de geheime sleutel vormt. Tegenwoordig spelen digitale handtekeningen een cruciale rol in de praktijk, met name in die electronische informatie systemen waar integriteit van informatie van groot belang is.

Nadat Ron Rivest, Adi Shamir en Leonard Adleman [100] een realisatie van trapdoor one-way functies hadden gegeven (de "RSA-functies"), gebaseerd op de computationele moeilijkheid van het factorizeren van samengestelde gehele getallen met alleen grote priemfactoren, heeft het cryptografische onderzoek een enorme vlucht genomen. Zonder op deze plaats technische details te vermelden, de crux van hun idee is dat het relatief eenvoudig is om twee willekeurige, verschillende, priemgetallen van, zeg, ieder 100 cijfers decimaal te selecteren en het product daarvan te berekenen, terwijl wanneer men *slechts dat product* gegeven is, het ondoenlijk is om de priemfactoren te berekenen. In de public key cryptografie wordt de graad van veiligheid van een systeem ontleend aan de computationele moeilijkheid van getaltheoretische problemen zoals factorizatie en discrete logaritmen, of het bestaan van functies met voor de cryptografie relevant eigenschappen. Een veiligheidsanalyse dient aan te tonen dat indien een cryptografisch systeem effectief gekraakt *zou* worden, men tevens een effectieve methode in handen *zou* hebben om die zeer moeilijke getaltheoretische problemen op te lossen. De huidige stand van zaken is dat indien men de probleem instanties groot genoeg kiest (als voorbeeld, het factorizeren van willekeurige samengestelde getallen van, zeg, meer dan 200 decimale cijfers is tot op heden ondoenlijk), elk bestaand wiskundig algoritme buitensporig veel tijd vergt om deze op te lossen. Sterker nog, het lijkt erop dat deze problemen intrinsiek moeilijk zijn.

Naast het onderzoek naar public key encryptie/decryptie en public key digital signatures, waarvoor in de jaren tachtig de theoretische fundamenten zijn gelegd, ontwikkelde de (theoretische) cryptografie zich in de breedte met het onderzoek naar diverse klassen *one-way functies*, (functies die gemakkelijk te berekenen zijn, maar ondoenlijk om te inverteren), *pseudorandomness* (functies waarvoor het ondoenlijk is de geproduceerde uitvoer van een willekeurige uitvoer te onderscheiden, en dat terwijl de grootte van de invoer slechts een fractie van de grootte van de uitvoer betreft), *anonimiteit in electronische transactie systemen* (bijvoorbeeld anonimiteit van de betaler in een digitaal betalings systeem), en zo verder. Andere fundamentele onderzoeksgebieden binnen de cryptografie zijn *secure multi-party computations* en *zero knowledge protocollen.*

Deze laatste twee begrippen zal ik nu nader toelichten, daar zij naast *digital signatures* een grote rol spelen in dit proefschrift. Losjes gezegd, in een *multi-party computation* berekenen een aantal partijen samen een gegeven functie (waarbij elk van de partijen z'n eigen invoer heeft) zodat de uitkomst aan iedereen bekend geraakt en de berekingswijze bestand is tegen deelnemende partijen die "vals spelen", terwijl de afzonderlijke invoer van elk van de partijen voor de andere partijen verborgen blijft. Een belangrijk voorbeeld is de *electronische verkiezing.* In zo'n systeem kunnen, bijvoorbeeld, alle daartoe gerechtigde Nederlanders bij de Tweede Kamerverkiezingen een electronisch "stembiljet" invullen en aan de autoriteiten sturen. Maar dan wel op zo'n manier dat individuele stemmen voor de autoriteiten verborgen blijven. Slechts de totale verkiezingsuitslag kan gepubliceerd worden, en *zelfs* door iedereen op juistheid gecontroleerd worden. Eveneens vanwege de gebruikte cryptografische methode, kan het vervalsen van de verkiezings uitslag ondoenlijk dan wel onmogelijk gemaakt worden.

Een *zero knowledge protocol* is een interactief vraag-en-antwoord spel tussen een "prover" (een partij die bewijst) en een "verifier" (een partij die verifieert). De prover beweert dat een zekere wiskundige stelling geldig is, maar wenst het bewijs van die stelling niet aan de verifier te tonen, wellicht omdat de prover bevreesd is dat de verifier met de eer gaat strijken. In ieder geval wil de prover de verifier ervan overtuigen dat de stelling klopt. Maar dan wel op zo'n manier dat de verifier daaruit geen informatie verkrijgt zodat die op zijn beurt een derde nu eveneens van de waarachtigheid van de stelling kan overtuigen. En natuurlijk wil de verifier niet overtuigd geraken van een stelling die onjuist is. Zero knowledge protocollen bieden de mogelijkheid om een skeptische verifier te overtuigen, zonder dat deze enige informatie over het bewijs verkrijgt. Deze zero knowledge protocollen spelen een een belangrijke rol in complexe cryptografische systemen, met name op plaatsen waar een bepaalde partij een andere partij ervan moet overtuigen dat, bijvoorbeeld, een zekere gegeven waarde op een van tevoren afgeproken manier is berekend, zonder de precieze waarden die ten grondslag hebben gelegen aan de berekening prijs te geven. Anderzijds, vindt zero knowledge, naast het daaraan gerelateerde *witness hiding*, een directe praktische toepassing in toegangscontrole van, bijvoorbeeld, computernetwerken (merk op dat een aanpak simpelweg gebaseerd op passwords geen hout snijdt wanneer de toegangscontrole over een onveilige lijn tot stand moet komen, en dat dan in ieder geval de te volgen procedure interactief dient te zijn om de mogelijkheid van *replay-attacks* uit te schakelen).

Dit proefschrift poogt de theoretische cryptografie en de cryptografische praktijk dichter bij elkaar te brengen. In de theorie worden zeer elegante oplossingen gegeven voor bovenstaande fundamentele cryptografische vraagstellingen. Deze oplossingen hebben dan de eigenschap dat men de veiligheid volgens een wiskunding bewijs kan herleiden tot de veiligheid van onderliggende cryptografische primitieven zoals bepaalde typen one-way functies of specifieke zeer ingewikkelde getaltheoretische problemen. Een nadeel is echter dat de resulterende methoden vaak te veel werk (berekening, tijd, opslag of communicatie) vergen om praktisch te kunnen zijn.

Aan de andere kant, methoden die direct zijn toegesneden op de praktijk (en dus "handzaam" zijn), missen meer dan eens duidelijkheid wat betreft hun veiligheid: de wiskundige reducties die moeten aantonen dat een gegeven praktisch systeem veilig is indien, bijvoorbeeld, factorizeren of het berekenen van discrete logaritmen inderdaad fundamenteel moeilijke problemen zijn, blijft vaak achterwege, en wel omdat het in die gevallen volstrekt onduidelijk [5] is hoe zo'n reductie zou moeten verlopen.

Teneinde deze discrepantie in een aantal belangrijke gevallen te doen verwijnen, worden in Hoofdstuk 2 zogenaamde *Σ-protocollen* ingevoerd: Merlin-Arthur "spelen" met drie zetten waarbij de verifier slechts random coinflips ("toevalsgetallen") geacht wordt te verzenden. Typisch cryptografische eigenschappen die hiervan geëist zullen worden zijn (een zekere vorm van) *zero knowledge* en *collision-intractability*. Deze laatste eigenschap houdt in dat het moeilijk moet zijn om voor zekere functies afgeleid van de Σ-protocollen (en onder zekere condities) twee verschillende invoer waarden te berekenen die dezelfde beeld waarde hebben. Σ-protocollen zullen de bouwstenen vormen voor de resultaten in de latere hoofdstukken, maar niet voordat er een theorie over is ontwikkeld, die leert hoe men op praktische en betrouwbare wijze kan "stapelen" met de gegeven bouwstenen, en hoe men deze abstracte bouwstenen praktisch kan realizeren.

Gebaseerd op Σ-protocollen, worden in Hoofdstuk 3 eerst *partial proofs* (partiële bewijzen) ingevoerd. Als voorbeeld, met een partieel bewijs kan een prover een verifier ervan overtuigen dat van, zeg, 100 gegeven stellingen er ten minste 50 waar zijn, zonder de verifier enige informatie te geven *welke* 50 waar zijn. Veel algemener gezegd, is het mogelijk om zero knowledge, collision-intractable Σ-protocollen die betrekking hebben op één stelling (willekeurig gekozen uit een zekere verzameling stellingen) over te voeren in Σ-protocollen (met dezelfde cryptografische eigenschappen) die betrekking hebben op n willekeurige stellingen waarop een willekeurige *monotone* conditie (dat wil zeggen, een voorschrift dat gegeven kan worden in termen van logische AND- en OR-operatoren) is gelegd. Indien een familie van zulke (polynomiaal begrensde) condities gegeven, kan de transformatie nog

[5] Dat wil zeggen, in theoretische zin. Van veel praktische cryptografische systemen is het tevens sinds lange tijd volstrekt onduidelijk hoe men ze efficiënt zou moeten breken.

steeds doorgang vinden, maar dan onder de aanname dat een zeker type efficiënt *secret sharing* schema beschikbaar is. Indien de probleem-instanties (de "stellingen") waarop de Σ-protocollen betrekking hebben, op de juiste manier uit een collectie "zeer moeilijke problemen" gekozen worden, dan is het resulterende scheme *witness hiding*: een malicieuze verifier kan de bewijzen niet aan de prover onttrekken. Er is een subtiel verschil tussen zero knowledge en witness hiding: in het laatste geval kan de verifier geen informatie onttrekken die deze verifier zou helpen later zelf als de prover op te treden, terwijl in het eerste geval de verifier helemaal geen informatie leert (buiten de juistheid van de stelling). Echter, in menig cryptografisch protocol volstaat witness hiding als garantie voor de veiligheid ervan. Tevens zijn witness hiding protocollen in het algemeen efficiënter dan zero knowledge protocollen.

Deze partial proofs zullen daarna de basis vormen van zero knowledge proof systeem voor het *circuit satisfiability*-probleem. Het resultaat bereikt de laagste bekende communicatie-complexiteit voor dit probleem, namelijk een lineair aantal *bit commitments* (een bit commitment wordt berekend door een functie waarvan de beeld waarde (het commitment) het bijbehorende origineel verbergt, terwijl het ondoenlijk is om twee originelen te vinden die door deze functie in hetzelfde beeld overgevoerd worden) als functie van de grootte van het onderhavige circuit. Voorgaande oplossingen vergden tenminste een logaritmische factor (in de grootte van het circuit) meer aan communicatie. Aangezien circuit satisfiability een NP-compleet probleem is (in een NP-probleem dient men voor een gegeven verzameling L en een gegeven element x te beslissen of dat element in die verzameling zit, terwijl gegeven is dat, indien $x \in L$, er een kort bewijs daarvan *bestaat* [6]), levert deze methode een communicatie-efficiënt (in termen van de grootte van een circuit dat vervulbaar is precies wanneer een gegeven x in de gegeven NP-taal L zit)) zero knowledge proof systeem voor alle "wiskundige" stellingen die als een NP-bewering op te vatten zijn. Onder NP-beweringen vallen bijvoorbeeld alle stellingen van het type "Laat y een gegeven waarde zijn en f een efficiënt berekenbare fuctie. Er bestaat een x zodanig dat $f(x) = y$)." In complexe cryptografische systemen komt het vaak voor dat een zero knowledge proof voor zo'n type stelling gegeven moet worden (waarbij f dan een one-way functie voorstelt). In praktische situaties kan de efficiënte methode die in dit proefschrift gepresenteerd wordt, aanzienlijke besparingen op de benodigde hoeveelheid communicatie opleveren.

Verdere resultaten in Hoofstuk 3 betreffen efficiënte zero knowledge proofs voor monotone afsluitingen van talen (verzamelingen bitstrings) die een membership-proof (een bewijs dat een gegeven element in de verzameling zit) van het type Σ-protocol toelaten. Steunend op sommige van de voor-

[6]Alleen de existentie is a priori gegeven, niet hoe men het bewijs zou moeten construeren.

gaande resultaten, wordt Hoofdstuk 3 afgerond met het presenteren van een klasse *digitale identificatie* methoden die bestand zijn tegen zogeheten *adaptive impersonation attacks*, de meest krachtig aanval die een malicieuze partij zou kunnen uitvoeren op een identificatie methode (binnen het polynomiale tijd berekenings model zoals dat in de public key cryptografie wordt gehanteerd), en het tot op heden meest efficiënte *electronische verkiezingssysteem.*

Hoofdstuk 4 gaat uitsluitend over *digitale handtekeningen.* Σ-protocollen worden hier aangewend om digitale handtekeningen te ontwikkelen die bestand zijn tegen *existential forgery under adaptively chosen message attacks*, het hoogste veiligheids niveau (wederom binnen het standaard polynomiale tijd berekenings model) van digitale handtekeningen. Enerzijds worden handtekeningen ontwikkeld die een zelfde communicatie complexiteit hebben als voorgaande methoden met relatief lage communicatie complexiteit, maar dan onder zwakkere "intractability" hypothesen (aannamen met betrekking tot de moeilijkheid van zekere computationele problemen). Het proefschrift wordt afgesloten met een digitale handtekening systeem gebaseerd op RSA. Nu waren er reeds zulke systemen, maar de bijzonderheid van dit schema is gelegen in het feit dat dit het eerste schema is dat zowel bewezen veilig is (in de strikte zin van resistentie tegen adaptively chosen message attacks, en onder aanname dat de RSA-functies veilig zijn) *en* praktisch genoeg is om, bijvoorbeeld, op een smartcard (plastic kaart met daarop een "intelligente" chip) uitgevoerd te kunnen worden.

Curriculum Vitae

Ronald J. F. Cramer, February 3rd 1968, Haarlem, The Netherlands.

Education:

1980–1986:
Gymnasium β (grammar school) at the Eerste Chr. Lyceum, Haarlem.

1986–1992:
M. Sc. in Pure Mathematics (Algebra & Geometry) from Leiden University.

Academic Positions:

1992–1995:
Junior researcher at CWI (Center for Mathematics & Computer Science), Amsterdam.
1995–1997:
Researcher at CWI.

[9] M. Bellare and and O. Goldreich: *On Defining Proofs of Knowledge*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 390–420.

[10] M. Blum, A. De Santis, S. Micali and G. Persiano: *Non-Interactive Zero Knowledge*, SIAM J. Computing, vol. 20, 1991, pp. 1084–1118.

[11] S. Bengio, G. Brassard, Y. Desmedt, C. Goutier and J.J. Quisquater: *Secure Implementation of Identification Systems*, Journal of Cryptology, 1991 (4): 175–183.

[12] M. Blum and S. Micali: *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*, SIAM Journal on Computing, vol. 13, No. 4, pp. 850–864, November 1984.

[13] J. Bos and D. Chaum: *Provably Unforgeable Signatures*, Proceeding of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 1–14.

[14] J. Benaloh and J. Leichter: *Generalized Secret Sharing and Monotone Functions*, Proceeding of Crypto '88, Springer Verlag LNCS, vol. 403. pp. 25–35.

[15] J. Benaloh and M. de Mare: *One-Way Accumulators: A Decentralized Alternative to Digital Signatures*, Proceedings of Eurocrypt '93, Springer Verlag LNCS, vol. 765, pp. 274–285.

[16] J. Benaloh: *Cryptographic capsules: A disjunctive primitive for interactive protocols*, Proceedings of Crypto '86, Springer Verlag LNCS, vol. 263, pp. 213–222.

[17] J. Benaloh: *Verifiable Secret-Ballot Elections*, PhD thesis, Yale University, Department of Computer Science Department, New Haven, CT, September 1987.

[18] J. Benaloh and D. Tuinstra: *Receipt-free secret-ballot elections*, Proceedings of STOC '94, ACM, pp. 544–553,

[19] J. Benaloh, M. Yung: *Distributing the power of a government to enhance the privacy of voters*, Proceedings of 5th ACM Symposium on Principles of Distributed Computing (PODC '86), pp. 52–62, New York, 1986, ACM.

[20] G. Bleumer, B. Pfitzmann and M. Waidner: *A Remark on a Signature Scheme where Forgery Can Be Proved*, Proceedings of Eurocrypt '90, Springer Verlag LNCS, vol. 473, pp. 441–445.

[21] J. Boyar, D. Chaum, I. Damgård and T. Pedersen: *Convertible Undeniable Signatures*, Proceedings of Crypto '90, Springer Verlag LNCS, vol. 537, pp. 189–205.

[22] J. Boyar, G. Brassard and R. Peralta: *Subquadratic Zero-Knowledge*, Journal of the ACM, November 1995.

[23] J. Boyar and R. Peralta: *Short Discreet Proofs*, Proceedings of Eurocrypt '96, Springer Verlag LNCS, vol. 1070, pp. 131–142.

[24] G. Brassard, D. Chaum and C. Crépeau: *Minimum Disclosure Proofs of Knowledge*, JCSS, vol.37, pp. 156–189, 1988.

[25] G. Brassard, C. Crépeau and M. Yung: *Everything in NP Can Be Argued in Perfect Zero Knowledge in a Bounded Number of Rounds*, Proceedings of 16th ICALP, Spinger Verlag LNCS, vol. 372, 1988, pp. 123–136.

[26] E. F. Brickell and K. S. McCurley: *An Interactive Identification Scheme Based on Discrete Logarithms and Factoring*, Journal of Cryptology, 5(1), pp. 29–39, 1992.

[27] D. Chaum: *Untraceable electronic mail, return addresses, and digital pseudonyms*, Communications of the ACM, 24(2):84–88, 1981.

[28] D. Chaum: *Blind Signatures for Untraceable Payments*, Proceedings of Crypto '82, Plenum, pp. 199–203.

[29] D. Chaum: *Security without Identification: Transaction Systems to Make Big Brother Obsolete* , Comminications of the ACM, vol. 28, no. 10, October 1985, pp. 1030–1044.

[30] D. Chaum, C. Crépeau, I. Damgard: *Multi-Party Unconditionally Secure Protocols*, Proceedings of STOC '88, pp. 11–19.

[31] D. Chaum, I. Damgård and J. van de Graaf: *Multiparty Computations ensuring Privacy of each Party's Input and Correctness of the Result*, Proceedings of Crypto '87, Springer Verlag LNCS, vol. 293, pp. 87–119.

[32] D. Chaum and H. van Antwerpen: *Undeniable Signatures*, Proceedings of Crypto '89, Springer Verlag LNCS, vol. 435, pp. 212–216.

[33] D. Chaum: *Zero Knowledge Undeniable Signatures*, Proceedings of Eurocrypt '90, Springer Verlag LNCS, vol. 473, pp. 458–464.

[34] D. Chaum and E. van Heyst: *Group Signatures*, Proceedings of Eurocrypt 91, Springer Verlag LNCS, vol. 547, pp. 257–261.

[35] D. Chaum: *Achieving Electronic Privacy*, Scientific American, August 1992.

[36] D. Chaum, T. P. Pedersen: *Wallet Databases with Observers*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 89–105.

[37] D. Chaum: *Designated Conformer Signatures*, Proceedings of Eurocrypt '94, Springer Verlag LNCS, vol. 950, pp. 86–91.

[38] L. Chen: *Witness Hiding Proofs and Applications*, PhD thesis, Aarhus University, Computer Science Department, Aarhus, Denmark, August 1994.

[39] L. Chen and T. P. Pedersen: *New group signature schemes*, Proceedings of Eurocrypt '95, Springer Verlag LNCS, vol. 950, pp. 171–181.

[40] J. Cohen and M. Fischer: *A robust and verifiable cryptographically secure election scheme*, Proceedings of FOCS '85, pp. 372–382.

[41] J. Cowie, B. Dodson, R. M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery and J. Zayer: *A World Wide Number Field Sieve Factoring Record: on to 512 bits*, to appear in the Proceedings of Auscrypt '96, Springer Verlag LNCS.

[42] R. Cramer and T. Pedersen: *Improved Privacy in Wallets with Observers*, Proceedings of Eurocrypt '93, Springer Verlag LNCS, vol. 765, pp. 329–343.

[43] R. Cramer and I. Damgård: *Secure Signature Schemes based on Interactive Protocols*, Proceedings of Crypto '95, Springer Verlag LNCS, vol. 963, pp. 297–310.

[44] R. Cramer, I. Damgård and T. Pedersen: *Efficient and Provable Security Amplifications*, to appear in Proceedings of 4th Cambridge Security Protocols Workshop, April 1996, Springer verlag LNCS (to appear).

[45] R. Cramer: *On Shared Randomness and the Size of Secure Signatures*, CWI technical report CS–R9530, April 1995.

[46] R. Cramer and I. Damgård: *Efficient Identification Schemes Secure against Impersonation and Man-in-the-Middle Attacks*, preprint, October 1995.

[47] R. Cramer, I. Damgård and B. Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Proceedings of Crypto '94, Springer verlag LNCS, vol. 839, pp. 174–187. Also in CWI Quarterly (Special Issue on Cryptography), 8 (2):111–127, June 1995.

[48] R. Cramer, M. Franklin, B. Schoenmakers and M. Yung: *Linear Secret ballot Election Schemes*, Proceedings of Eurocrypt '96, Springer verlag LNCS, vol. 1070, pp. 72–83.

[49] R. Cramer and I. Damgård: *New Generation of Secure and Practical RSA-based Signatures*, Proceedings of Crypto '96, Springer verlag LNCS, vol. 1109, pp. 173–185.

[50] I. Damgård and R. Cramer: *Linear Zero Knowledge – A Note on Efficient Zero Knowledge Proofs and Arguments*, Basic Research in Computer Science (BRICS), RS–96–7, Aarhus University, April 1996.

[51] I. Damgård and R. Cramer: *On Monotone Function Closure of Perfect and Statistical Zero Knowledge*, CWI technical report, CS–R9618, April 1996.

[52] I. Damgård: *Collision Free Hash Functions and Public-Key Signature Schemes*, Proc. of Eurocrypt '87, Springer Verlag LNCS, vol. 304, pp. 203–216.

[53] I. Damgård: *Interactive Hashing can Simplify Zero-Knowledge Protocol Design Without Computational Assumptions*, Proceedings of Crypto '93, Springer verlag LNCS, vol. 773, pp. 100–109.

[54] I. Damgård, O. Goldreich, T. Okamoto and A. Wigderson: *Honest Verifier vs. Dishonest Verifier in Public Coin Zero Knowledge Proofs*, Proceedings of Crypto '95, Springer Verlag LNCS, vol. 963, pp. 325–338.

[55] National Bureau of Standards. *Data encryption standard.* Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington DC, January 1977.

[56] W. Diffie and M. Hellman: *New Directions in Cryptography*, IEEE Transactions on Information Theory IT-22 (6): 644–654, 1976.

[57] C. Dwork and M. Naor: *An Efficient Existentially Unforgeable Signature Scheme and its Applications*, Proceedings of Crypto '94, pp.234–246.

[58] National Institute of Technology and Standards: *Specifications for the Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication, US. Department of Commerce, 1993.

[59] S. Even, O. Goldreich and S. Micali: *On-Line/Off-Line Digital Signatures*, Proceedings of Crypto '89, Springer Verlag LNCS, vol. 435, pp.263–275.

[60] S. Even, O. Goldreich and S. Micali: *On-Line/Off-Line Digital Signatures*, Journal of Cryptology (1996) 9: 1–19..

[61] T. ElGamal, *A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*, IEEE Transactions on Information Theory, IT-31 (4): 469–472, 1985.

[62] L.Fortnow: *The complexity of perfect zero-knowledge*, Adv. in Comp. Research, vol. 5, 1989, pp.327-344.

[63] A. Fiat and A. Shamir: *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Proceedings of Crypto '86, Springer Verlag LNCS, vol. 263, pp. 186–194

[64] U. Feige, A. Shamir: *Witness Indistinguishable and Witness Hiding Protocols*, Proceedings of STOC '90, pp. 416–426.

[65] U. Feige and A. Shamir: *Zero-Knowledge Proofs of Knowledge in Two Rounds*, Proceedings of Crypto '89, Springer Verlag LNCS, vol. 435, pp. 526–544.

[66] U. Feige, A. Fiat and A. Shamir: *Zero-Knowledge Proofs of Identity*, Journal of Cryptology 1 (1988) 77–94.

[67] R. Gennaro: *Achieving independence efficiently and securely*, Proceedings 14th ACM Symposium on Principles of Distributed Computing (PODC '95), ACM, 1995.

[68] S. Goldwasser, S. Micali and C. Rackoff: *The Knowledge Complexity of Interactive Proof Systems*, SIAM J.Computing, Vol. 18, pp. 186-208, 1989.

[69] S. Goldwasser, S. Micali and A. Yao: *Strong Signature Schemes*, Proceedings of STOC '83, ACM, pp.431–439.

[70] S. Goldwasser, S. Micali and R. Rivest: *A Digital Signature Scheme Secure Against Chosen Message Attacks*, SIAM Journal on Computing, 17(2): 281–308, 1988.

[71] O. Goldreich, S. Micali and A. Wigderson: *Proofs that yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design*, Proceedings of FOCS '86, pp. 174–187.

[72] O. Goldreich, S. Micali and A. Wigderson: *Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems*, Journal of the ACM, 38(1): 691–729, 1991.

[73] O. Goldreich, S. Micali and A. Wigderson: *How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority*, Proceedings of STOC '87, ACM, pp. 218–229.

[74] O. Goldreich: *Two Remarks Concerning the GMR Signature Scheme*, Proceedings of Crypto '86, Springer Verlag LNCS, vol. 218, pp. 104–110.

[75] O. Goldreich: *Foundations of Cryptography–Fragments of a Book*, available from `http://www.wisdom.weizmann.ac.il/people/homepages/oded/oded.html`, 1995.

[76] L. Guillou, J.J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proceedings of Eurocrypt '88, Springer Verlag LNCS, vol. 330, pp. 123–128.

[77] J. vd Graaf and R. Peralta: *A Simple and Secure Way to Show the Validity of your Public Key*, Proceedings of Crypto '87, Springer Verlag LNCS, vol. 293, pp. 128–134.

[78] G. H. Hardy, E. M. Wright: *An Introduction to the Theory of Numbers*, fifth edition, 1979, Oxford Science Publications.

[79] E. van Heijst and T. Pedersen: *How to make Efficient Fail-Stop Signatures*, Proceedings of Eurocrypt '92, Springer Verlag LNCS, vol. 658, pp. 366–377.

[80] E. van Heijst, T. Pedersen and B. Pfitzmann: *New Constructions of Fail-Stop Signatures and Lower Bounds*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 15–30.

[81] *Information Technology – Security Techniques – Digital Signature Scheme Giving Message Recovery*, ISO/IEC Standard 9796, first edition, International Standards Organization, Geneva.

[82] M. Jacobson, R. Impagliazzo and K. Sako: *Designated Verifier Proofs and their Applications*, Proc. of Eurocrypt '96, Springer Verlag LNCS, vol. 1070, pp. 143–154.

[83] J. Kilian: *A note on Efficient Proofs and Arguments*, Proceedings of STOC '92, pp. 723–732.

[84] J. Kilian: *Efficient Interactive Arguments*, Proceedings of Crypto '95, Springer Verlag LNCS, vol. 963, pp. 311–324.

[85] L. R. Knudsen: *Block Ciphers – Analysis, Design and Applications*, Ph. D thesis DAIMI PB–485, November 1994, Aarhus University, Denmark.

[86] A. K. Lenstra and H. W. Lenstra, Jr. (eds.): *The Development of the Number Field Sieve*, Lecture Notes in Maths. vol. 1554, Springer Verlag, 1993.

[87] A. J. Menezes: *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, ISBN 0-7923-9368-6, 1993.

[88] R. C. Merkle: *A Certified Digital Signature*, Proceedings of Crypto '89, Springer Verlag LNCS, vol. 435, pp. 234–246.

[89] R. C. Merkle: *A Digital Signature Based on a Conventional Encryption Function*, Proceedings of Crypto '87, Springer Verlag LNCS, vol. 293, pp. 369–378.

[90] M. Naor, R. Ostrovsky, Venkatesan and M.Yung: *Zero-Knowledge Arguments for NP can be Based on General Complexity Assumptions*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 51–58.

[91] M. Naor, M. Yung: *Universal One Way Hash functions and their Cryptographic Applications*, Proceedings of STOC '89, pp. 33–43.

[92] M. Naor: personal communication, April 1996.

[93] T. Okamoto: *Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 31–53.

[94] H. Ong and C. P. Schnorr: *Fast Signature Generation with a Fiat-Shamir-like Scheme*, Proceedings of Eurocrypt '90, Springer Verlag LNCS, vol. 473, pp. 432–440.

[95] R. Ostrovsky, R. Venkatesan and M. Yung: *Interactive Hashing Simplifies Zero-Knowledge Protocol Design*, Proceedings of Eurocrypt '93, Springer Verlag LNCS, vol. 765, pp. 267–273.

[96] T. P. Pedersen: *Non-interactive and information-theoretic secure verifiable secret sharing*, Proceedings of Crypto '91, Springer Verlag LNCS, vol. 576, pp. 129–140.

[97] D. Pointcheval and J. Stern: *Security Proofs for Signature Schemes*, Proceedings of Eurocrypt '96, Springer Verlag LNCS, vol. 1070, pp. 387–398.

[98] B. Pfitzmann: *Fail-Stop Signatures Without Trees*, Hildesheimer Informatik-Berichte 16/94, Universität Hildesheim, Juni 1994.

[99] R. Raz and A. Wigderson: *Monotone Circuits for Matching Require Linear Depth*, J. ACM, vol. 39, 1992, pp. 736–744.

[100] R. Rivest, A. Shamir and L. Adleman: *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of ACM, 21 (1978), pp. 120–126.

[101] J. Rompel: *One-Way Functions are Necessary and Sufficient for Secure Signatures*, Proceedings of STOC '90, pp. 387–394.

[102] A. De Santis, G. Di Crescenzo, and G. Persiano: *Secret Sharing and Perfect Zero-Knowledge*, Proceedings of Crypto '93, Springer Verlag LNCS, vol. 773, pp. 73–84.

[103] A. De Santis, G. Di Crescenzo, G. Persiano and M. Yung: *On Monotone Formula Closure of SKZ*, Proceedings of FOCS '94, pp. 454–465.

[104] K. Sako and J. Kilian: *Secure voting using partially compatible homomorphisms*, Proceedings of Crypto '94, Springer Verlag LNCS, vol. 839, pp. 411–424.

[105] K. Sako and J. Kilian: *Receipt-free mix-type voting scheme—a practical solution to the implementation of a voting booth.*, Proceedings of Eurocrypt '95, Springer Verlag LNCS, vol. 921, pp. 393–403.

[106] C. P. Schnorr: *Efficient Signature Generation by Smart Cards*, Journal of Cryptology, 4 (3): 161–174, 1991.

[107] A. Shamir: *How to Share a Secret*, Communications of the ACM 22 (1979) 612–613.

[108] A. Shamir: *The Generation of Cryptographically Strong Pseudo-Random Sequences*, Proceedings of Crypto '81, U. C. Santa Barbara Dept. of Elec. and Computer Eng. , Techn. Report 82-04, 1982.

[109] C. E. Shannon: *A Mathematical Theory of Communication*, Bell Syst. Techn. Journal, vol. 27, pp. 379–423 and 623–656, July and October 1948.

[110] C. E. Shannon: *Communication Theory of Secrecy Systems*, Bell Syst. Techn. Journal, vol. 28, pp. 656–715, October 1949.

[111] P. Shor: *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings of FOCS '94, pp. 124–134.

[112] V. Shoup: *On the Security of a Practical Identification Scheme*, Proceedings of Eurocrypt '96, Springer Verlag LNCS, vol. 1070, pp. 344–353.

[113] G. J. Simmons (ed.): *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, ISBN 0–87942-277-7, 1992.

[114] D. R. Stinson: *Cryptography - Theory and Practice*, CRC Press, 1995.

[115] M. Tompa and H. Woll: *Random Self-Reducibility and Zero-Knowledge Proof of Information Possession*, Proceedings of FOCS '87, pp. 472–482.

[116] A. Yao: *How to generate and exchange secrets*, Proceedings of FOCS '86, pp. 162–167.

Index

B_σ, 124
DL^*, 50, 149
G_q, 29
RSA^*, 50, 155
$RSA_{n,v}$, 32
RX, 20
R_A, 20
R_B, 20
Σ-protocols, 19
$\mathbb{Z}_n$, 29
$\mathbb{Z}_n^*$, 29
$\hat{f}_g$, 51, 92
DFS, 133, 145, 157
$\phi(\cdot)$, 20
$a(\cdot)$, 20
$c(\cdot)$, 20
q-th residuosity, 90
t_A, 20
t_B, 20
$z(\cdot)$, 20
`comp` 64
`cons` 64
`dist` 64
`rec` 64

accepting conversation, 21
attack
  adaptive chosen message, 125, 128, 136, 153, 160
  man-in-the-middle, 115
  passive, 36
  random challenge, 118
  random message, 125
authentication length, 20
authentication path, 131

binary relation, 20
bit commitment, 78, 81
  negation of, 82
  trapdoor, 81
  unconditionally binding, 81
  unconditionally hiding, 81
  with linear proof of contents, 82
  with linear witness hiding proof of trapdoor, 82
black-box, 21
  rewindable, 21

challenge, 20

claw-free pairs of trapdoor permutations, 53, 119
  based on factoring, 55
  special, 56, 136
  special, based on factoring, 57
collision, 22
collision-intractable, 23
collision-property, 22
completeness, 22
complexity
  asymptotic, 12
  communication, 12
  cryptographic, 11
  round, 12
conversation, 20
  length of, 21
cryptographic protocol, 7
cryptology, 3

digital signatures, 123
  definition of, 128
  Goldwasser-Micali-Rivest, 130
  history of, 123
  security of , 128
discrete logarithms, 29

existential forgery, 125

GMR-scheme, 130

hash function, 45
homomorphic encryption, 108
  proof of validity, 109

identification scheme, 52, 114
  definition of, 114
  security of, 114
infeasible, 9
initial message, 20
instance generator, 23
interactive
  argument, 28
  hashing, 37
  proof, 27
intractability assumption, 11
intractable, 9
invulnerable generator, 23

knowledge extractor, 27

modular design, 13
monotone function, 63
  Boolean formula, 63
  dual of, 63
  threshold, 64

one-time pad, 1
one-way function, 4
  trapdoor, 4
one-way group homomorphism, 47, 144

partial proofs, 62
proof of knowledge, 27
proof of security, 10
public key cryptography, 3

reduction, 10
reply, 20
rewindable black-box, 24
Rivest-Shamir-Adleman scheme, 31
RSA, 31, 155

secret ballot election, 105
  linear, 105
  privacy of, 105
  robustness of, 105
  universal verifiability of, 105
secret sharing, 64
  efficient, 64
  for monotone formulas, 65
  for threshold functions, 68
  perfect, 65
  with completion, 64
security amplification, 151
signature bound, 124
signature protocol, 28
  linear, 143
  quasi, 28
special one-way group isomorphism, 48, 49, 91, 155

special soundness, 22

Turing machine, 9, 19
  interactive, 10, 20
  polynomial time, 20
  probabilistic, 10, 19

witness, 20
  extractor, 25
  hiding, 25, 26
  indistinguishable, 26

zero knowledge, 24
  honest verifier simulator, 24
  interactive argument, 28, 88
  interactive proof, 28, 87
  simulator, 24
  special honest verifier, 24
  special honest verifier simulator, 24
zero knowledge proof
  general, 78
  honest verifier, 24
  linear general, 78
  of circuit satisfiability, 78