Provably Secure and Practical Signature Schemes

Vom Fachbereich Informatik
der Technische Universität Darmstadt
genehmigte

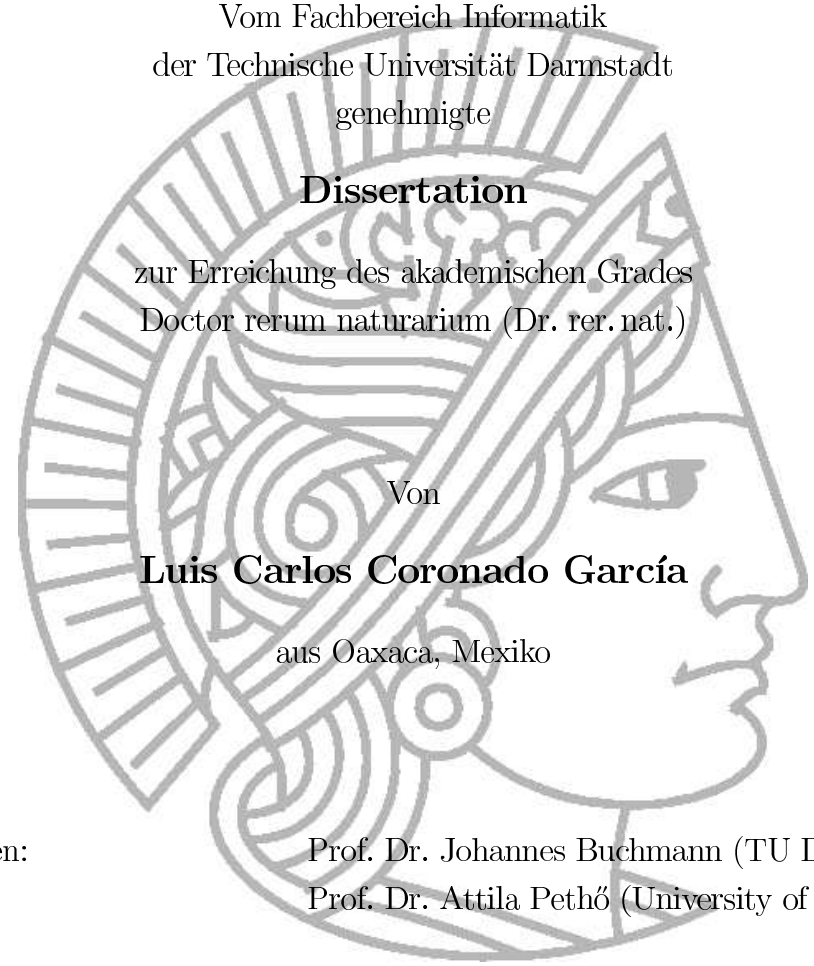
Dissertation

zur Erreichung des akademischen Grades
Doctor rerum naturarium (Dr. rer. nat.)

Von

Luis Carlos Coronado García

aus Oaxaca, Mexiko



Referenten: Prof. Dr. Johannes Buchmann (TU Darmstadt)
Prof. Dr. Attila Pethő (University of Debrecen, Hungary)

Tag der Einreichung : 08. November 2005
Tag der mündlichen Prüfung: 13. Dezember 2005

Darmstadt, 2005
Hochschulkennziffer: D 17

To my beloved Verónica

In Memoriam Josefina García

Acknowledgements

I would like to thank Prof. Dr. Johannes Buchmann, who wisely directed this work. I would also like to thank Prof. Dr. Attila Pethő for accepting the task of the second referee.

I would like to thank my collegues, specially Ulrich for reviewing this work and making helpful assertions, Evangelios and Christoph for helpful comments and fruitful discussions, Ulrike and Marita for their help.

I would like to thank my friends, specially Miguel Alejandro and Maribel.

I would like to thank my friends of the family Santos Mendoza, who are as a second family to me.

I would like to thank my beloved and wounderful wife, Verónica, for her understanding, support, patience and love.

Zusammenfassung

Diese Arbeit leistet Beiträge zu zwei unabhängige Themen: (1) Beweisbar sichere und effiziente Signaturverfahren und (2) die Analyse von Algorithmen zur Multiplikation ganzer Zahlen.

Es ist nach wie vor ungewiss, ob Quantenrechner gebaut werden können, die groß genug sind, um kryptographisch relevante Probleme zu lösen. Shor [Sho94] stellte 1994 einen Quantenalgorithmus vor, der das Faktorisierungsproblem für ganze Zahlen und das Diskrete-Logarithmen-Problem mit polynomiellen Aufwand löst. Signaturverfahren wie RSA, DSA und ElGamal werden mit der Verfügbarkeit von Quantenrechnern offensichtlich unsicher. Immerhin berichten Breyta et al. in [VSS+01] von einer erfolgreichen Implementierung dieses Algorithmus auf einem Quantenregister bestehend aus siben Qubits.

In der vorgelegten Arbeit schlagen wir Signaturverfahren vor, die beweisbar sicher und effizient sind. Die Sicherheit der beschreibenen Verfahren basiert auf der Sicherheit der eingesetzten Hashfunktion und des pseudozufälligen Bitgenerators. Ferner untersuchen und vergleichen wir einige Multiplikationsmethoden. Dieser Vergleich wurde dadurch motiviert, dass einige der am weitesten verbreiteten Signaturverfahren – RSA, DSA und ElGamal – die modularen Exponentiation verwenden, was eine schnelle Multiplikation großer ganzer Zahlen erfordert. Unsere Untersuchung geht über das einfache asymptotische Verhalten der Verfahren hinaus, weil wir die Multiplikationen und Additionen der zu Grunde liegenden Maschinentypen einbeziehen.

Abstract

This work contributes to two independent topics: (1) Provably secure and efficient signature schemes and (2) the analysis of certain integer multiplication algorithms.

It is uncertain whether quantum computers are feasible that are powerful enough to solve non-trivial problems. In 1994, Shor [Sho94] proposed a quantum algorithm for solving both the integer factorization problem and the discrete logarithm problem in polynomial time. This algorithm clearly renders signature schemes like RSA, DSA or ElGamal completely insecure as soon as powerful quantum computers come into existence.

In this thesis we propose signature schemes that are provably secure and efficient. The security of the described schemes relies on the security of hash function and the pseudorandom bit generator used. We also study and compare certain integer multiplication algorithms. The motivation of this comparison is the use of modular exponentiation – which, in turn, needs the multiplication of large integers – in often used signature schemes like RSA, DSA, and ElGamal. Our study is more detailed than just the asymptotic behaviour, since we take multiplication and addition of base-words into account.

Contents

List of Figures

List of Tables

Introduction

Now a days the extended use of electronic media requires mechanisms which assure that the certain electronic information comes from who is supposed to. The legality of contracts and documents in general, requiring autograph signatures for its validity could be transfered to electronic media with the appropriate elements. An autograph signature can be stored in order to ratify any document where such a signature appears. This cannot be the case in the electronic world, since an electronic file can be copied as many times you want. Cryptography is a helpful tool to reach the goal of digital signatures. In the 70's cryptography of public key or asymmetric was introduced. Since then some digital signature schemes have appeared. Roughly speaking, a signature scheme allows to verify whether or not a document has been digitally signed with a certain signing key. The connection between a signing key and a certain person stays out of the scope of this dissertation.

Many of the signature schemes appearing in the literature are based on the integer factorization problem (FP) or on the discrete logarithm problem (DLP). In the middle 90's a quantum algorithm was presented to solve the FP and DLP in quantum polynomial time. The obvious consequence of the existence of quantum computers is that those signature schemes like the world wide used RSA or like ElGamal or like ECDSA would not be secure any more. At the moment of writing this dissertation there not exist "big" quantum computers. Only some quantum systems of seven qu-bits has been developed for implementing a harmless version of the quantum factoring algorithm. Such an implementation is still far away for being a threat to cryptography. At this point two questions appear: (1) Cryptosystems based on FP or DLP can be still used during the develop of quantum

systems into "big" quantum computers? (2) Can signature schemes be provided which are efficient and secure though quantum computers?

Trying to answer the first question we could say that the bigger the keys of such schemes are the more secure they are. Unfortunately, an increase of the key length effects the efficiency of the scheme. Thus, an analysis of elements which permit the efficient use of those cryptosystems is required. Regarding to question two, there exist signature schemes based on hash functions and none quantum algorithm for finding pre-image or collision in quantum polynomial time is known, yet. Some disadvantages of those signature schemes are the limited number of possible signatures and the number and size of their elements which result in a less efficient scheme in compare with conventional schemes. Intuitively, a one-time signature scheme is one that guarantees that it is secure if it is not used more than once. A multi-time signature scheme is one that cannot be employed more than certain number of times.

In the late 70's Merkle proposed a signature scheme, although he did not give any formal proof of its security. Basically, his scheme helps to transform a one-time signature scheme into a multi-time one. The security of that scheme is based on that of their both underlying hash function and one-time signature scheme. Since there exist one-time signature schemes whose security is based on the security of hash functions, the security of the Merkle signature scheme can rely only on the security of the underlying hash functions.

Goldwasser, Micali and Rivest introduced concepts of types of attacks and forgeries on signature schemes. Basically, a signature scheme is secure if it is computationally difficult to produce a forgery. On the other hand, the security of hash functions rely on properties like one-way-ness and collision resistance. Intuitively, we can say that a function is one-way if the obtainment of a pre-image of a given value is not computationally feasible. In a similar way we say that a function is collision resistance if the obtainment of any couple of different values which result in the same image is not computationally feasible. Formal definitions of these concepts on families of functions are made in an asymptotical and in a concrete way. We work with concrete definitions, where the computational feasibility is measured as the advantage of any adversary modeled as a probabilistic

algorithm in obtaining the required pre-image or the required collision within a limited time of computation.

Outline

This work consists of two independent chapters.

In Chapter 1 we focus on signature schemes. The goal of that chapter is to provide provably secure and efficient signature schemes. The security of the analyzed and the proposed schemes rely on the cryptographic properties of the underlying hash functions and pseudorandom number generators. We give concrete security of the proposed schemes. First we describe the Lamport-Diffie one-time signature scheme and after that we provide an improvement to that scheme. We also give a formal proof of the security of our proposal and show some experimental results concerning its efficiency. Then, we describe the original multi-time signature scheme proposed by Merkle and provide a formal proof of its security. Finally we propose some variants of the original Merkle signature scheme and provide formal proofs of the security of each of them. We also give some estimates of their efficiency. We review definitions of the cryptographic security of hash function, pseudorandom generators and signature schemes which are concrete instead of asymptotical. We avoid the Random Oracle Model.

Since exponential modular multiplications are required in ElGamal and RSA alike cryptosystems and in those operations integer multiplications are involve, we analyze in Chapter 2 some multiplication algorithms. The goal in that chapter is to estimate which algorithm is faster for certain bit length. We take into account multiplications and additions of "digits" of the integers to be multiplied.

Chapter 1

Signature Schemes

In this Chapter we propose some improvements to the Lamport-Diffie one-time signature scheme (LDots) and to the Merkle multi-time signature scheme. We also provide a proof of their cryptographic security. Finally, we show some experimental results about their efficiency.

1.1 Introduction

Conventional signature scheme such RSA and ECC are widely used all around the world. The conventional signature schemes (css) do not have any restriction in the number of signatures that can be created during the lifetime of the private and public keys. These css are based on problems like the discrete logarithm problem (DLP) or the integer factorization problem (FP).

In the middle of the eighties the concept of quantum computer was introduced. Until the beginning of the nineties is when an advantage of quantum computers on classical ones was presented. In the middle of the nineties Shor presented a quantum algorithm for solving the FP and the DLP in polynomial time. A quantum system of seven qu-bits was created around 2001 for implementing Shor's algorithm. That quantum system is far away to be a threat to RSA or ECDSA alike cryptosystems. We do not know whether quantum computers which process hundreds of qu-bits are physically possible or not. In any case our goal is to provide efficient signature schemes whose security relies on their primitives like

cryptographically secure hash functions and pseudorandom number generators.

Merkle proposed in [Mer90] a multi-time signature scheme. Roughly speaking, his proposed scheme transforms any one-time signature scheme (ots) into a multi-time one (mts). In his original work he employed the Lamport-Diffie one-time signature scheme (LDots). He sketched the security of his proposal, but a formal proof was not given.

Goldwasser, Micali and Rivest provided in [GMR88] the notions of different classes of attacks and forgeries. They also proposed a signature scheme which is not (type of forgery) "existentially forgeable" under (type of attack) "adaptive chosen message attack" if certain problem is difficult to solve. Intuitively, if factoring integers is computationally difficult, then their proposed scheme is secure in the sense that any efficient algorithm which produces forged signatures yields to an efficient algorithm for factoring.

We consider the proposal of Merkle. We provide a formal proof of its security under the GMR model. We develop improved versions of the original proposal, provide proof of the security of each version and, finally, give some experimental results about the efficiency of each version.

A hash function is employed in each proposed scheme. Such a function is required to be collision resistant. Actually that assumption could be relaxed, since the collision needed to forge any proposed scheme is quite particular. For the sake of clarity, the underlying hash function will be required as collision resistant instead of resistant to a particular type of collisions.

In Section 1.2 we review some definitions and fix notation for this Chapter. In Section 1.3 we provide an improvement to the LDots, give a formal proof of its security and discuss its efficiency. In Section 1.4 we review the original Merkle signature scheme (oMss) and provide a formal proof of its security. In Section 1.5 we discuss the efficiency of the oMss. In Section 1.6 we present our improvements to the oMss and discuss their security and efficiency. In Section 1.7 we show some experimental results and compare them with analogous ones made with RSA. Finally, in Section 1.9 we present pseudo code for the proposed schemes.

1.2 Definitions and Notation

In this Section we review the definitions of hash functions, signature schemes, pseudorandom bit generators and cryptographic security.

Hash functions have an important role in cryptography. For instance, they are used in signature schemes, maybe either as a primitive of such scheme or as an auxiliary function for signing messages of any length. Loosely speaking, and ideal hash function $\mathcal{H}$ has as input a bit string of any length and as output a bit string of fixed length, i. e., $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^s$, for some $s \in \mathbb{N}$. The output value $\mathcal{H}(x)$ must be easy to compute for any x. $\mathcal{H}$ is one-way if a pre-image of $\mathcal{H}(x)$ is not feasible to obtain with the only knowledge of $\mathcal{H}(x)$, of course. $\mathcal{H}$ is collision resistant if a couple (x, y), where $x \neq y$, is difficult to find such that $\mathcal{H}(x) = \mathcal{H}(y)$. In practice we are restricted to inputs of finite bit length and have limited computational resources.

Informally, we say that a scheme or a function or a family of functions has (t, ϵ)-*property* $\mathcal{P}$ if no Adversary $\mathcal{A}$, modeled as a probabilistic algorithm which runs within time t, succeeds with probability larger than ϵ in breaking property $\mathcal{P}$. Here, we adopt the convention that the time-complexity is the total worst-case execution time of $\mathcal{A}$ plus the size of its code, all measured in some fixed model of computation.

We have adopted the definition of one-way and collision resistant functions from [RS04].

Definition 1.1 *Let $\mathcal{H} = \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^s$ be a family of functions. $\mathcal{H}$ is* (t, ϵ) one-way *if $\forall \mathcal{A}$ which run within time t, the advantage $Adv(\mathcal{A})$ of the adversary $\mathcal{A}$ is*

$$Pr[H_K(M') = H_K(M) | M' \leftarrow \mathcal{A}(K, H_K(M));\ M \in_R \{0,1\}^m;\ K \in_R \mathcal{K}] < \epsilon.$$

Here, $\mathcal{A}$ is an adversary modeled by a probabilistic algorithm.

Definition 1.2 *Let $\mathcal{H} = \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^s$ be a family of functions. $\mathcal{H}$ is* (t, ϵ) collision resistant *if $\forall \mathcal{A}$ which run within time t, the advantage $Adv(\mathcal{A})$ of the adversary $\mathcal{A}$ is*

$$Pr[(H_K(M') = H_K(M)) \wedge (M' \neq M);\ (M, M') \leftarrow \mathcal{A}(K);\ K \in_R \mathcal{K}] < \epsilon.$$

Here, $\mathcal{A}$ is an adversary modeled by a probabilistic algorithm.

Now we regard definitions concerning to signature schemes. Loosely speaking, a *signature scheme* is given as a triple $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ of probabilistic polynomial-time algorithms. Gen creates a signing key and a verifying key. Sig creates signatures of messages with the private key. Ver verifies with the public key whether a signature and a message are related through the corresponding private key. There are a variety of types of signature forgeries and attacks on signature schemes. We focus on the security of signature schemes against existential forgery under adaptive chosen message attacks. Roughly speaking, a signature scheme is secure in this sense if it is not computationally feasible to forge any signature of some message, given the public key and access to a sign oracle. A *one-time signature scheme* is a signature scheme which is guaranteed to be secure as long as the private key of an instance is not used more than once. A *pseudorandom bit generator* is a probabilistic polynomial-time algorithm whose output looks random.

Now we recall some definitions. Let $\mathcal{M}, \mathcal{K}_\mathcal{S}, \mathcal{K}_\mathcal{V}$ and $\mathcal{S}$ be the space of messages to be signed, the space of signing keys, the space of verification keys and the space of signed massages, respectively.

Definition 1.3 *A* signature scheme Sign *is a triple,* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$*, of probabilistic polynomial-time algorithms which satisfy the following conventions:*

Key generation algorithm Gen**.** *Given as input the security parameter* 1^s *and maybe some other information* I*,* Gen *outputs a pair* $(X, Y) \in \mathcal{K}_\mathcal{S} \times \mathcal{K}_\mathcal{V}$*.* X *is called the* signing *or* private *or* secret key *and* Y *is called the* verification *or* public key*.* s *is the* security parameter*.*

Signing algorithm Sig**.** *Having as input* $(M, X) \in \mathcal{M} \times \mathcal{K}_\mathcal{S}$*,* Sig *outputs an element* $\sigma \in \mathcal{S}$ *which is called a* signature *(of the message* M *with the signing key* X*).*

Verification algorithm Ver**.** *Given as input* $(M, \sigma, Y) \in \mathcal{M} \times \mathcal{S} \times \mathcal{K}_\mathcal{V}$*,* Ver *outputs an element in the set* $\{true, false\}$*.* $\mathsf{Ver}(M, \mathsf{Sig}(M, X), Y) = true\ \forall (X, Y)$ *obtained by* Gen *and all* $M \in \mathcal{M}$*.*

Goldwasser, Micali and Rivest formalized in [GMR88] the concept of security. They introduced notions like existential forgery and adaptive chosen message attack.

Roughly speaking, an existential forgery is when an adversary obtain a message M and a valid signature σ of M, but the adversary does not have any control of the obtained message. An adaptive chosen message attack is when the adversary chooses the message to be signed. His selection can be made after he has seen signed messages. In this sense we informally say that a signature scheme is secure if forgeries of signatures are extreme difficult to obtain. We have adopted the following definition.

Definition 1.4 ([BMS03]) *Let* $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be a signature scheme.* Sign *is* (t, ϵ, n) existentially unforgeable under adaptive chosen message attacks*, if for any algorithm* $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ *running in time at most* t *its advantage* $AdvSig_{\mathcal{F}}$ *in forging a signature is*

$$Pr[\mathsf{Ver}(\mathcal{F}_2(T), PK) = 1 \mid (SK, PK) \leftarrow \mathsf{Gen}(1^s);\ T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, SK)}(1^s, PK)] < \epsilon,$$

where $\mathcal{F}_1$ *requests no more than* n *signatures from* Sig *and the message output by* $\mathcal{F}_2$ *is different from the messages signed by* Sig*. The probability is taken over the coin tosses of* Gen*,* Sig*,* $\mathcal{F}_1$ *and* $\mathcal{F}_2$*. Here* $\mathcal{F}_2(T)$ *outputs a message/signature pair.*

We take the definition of one-time signature scheme from [EGM90] and reformulate it as follows.

Definition 1.5 *Let* $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be a signature scheme. We say that* Sign *is a* (t, ϵ) one-time signature scheme *or* ots scheme *for short, if* Sign *is* $(t, \epsilon, 1)$ *existentially unforgeable under adaptive chosen message attack.*

If we want to save space for the private key in some specific signature scheme, we need to re-compute the keys from a kind of seed. That is why a definition of a deterministic key generation algorithm is needed.

Definition 1.6 *Let* Gen *be a deterministic polynomial-time algorithm and let* Sig*,* Ver *be two probabilistic polynomial-time algorithms. If* Gen*,* Sig *and* Ver *satisfy the following conventions:*

1. ***Deterministic key generation algorithm*** Gen. *On input the security parameter* 1^s*, a seed* $\in \{0,1\}^s$ *and maybe some other information* I*,* Gen *outputs a pair* $(X, Y) \in \mathcal{K}_{\mathcal{S}} \times \mathcal{K}_{\mathcal{V}}$*.* X*,* Y *and* s *are called as in definition 1.3 on page 8 and seed is call the* seed for the generation algorithm.

2. ***Signature scheme.*** *Define the algorithm Genp as follow. Given as input* 1^s *and maybe some other information* I*, Genp chooses seed* $\in \{0,1\}^s$ *at random and calls* Gen *with input* 1^s*, seed and maybe* I*. Then* $(Genp, \mathsf{Sig}, \mathsf{Ver})$ *forms a signature scheme.*

we call $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *a* signature scheme with deterministic key generation.

With the previous notation, we call $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ a *one-time signature scheme with deterministic key generation* if $(Genp, \mathsf{Sig}, \mathsf{Ver})$ is a one-time signature scheme.

In the rump session of Eurocrypt 97, Anderson introduced the idea of forward security [And02]. The notion of a forward secure scheme is as follows. The use of a public key is divided into periods during its lifetime. The public key remains the same whereas the private key is changed when it passes from one period into the following one. If the private key is compromised in a certain period, the signatures which were made in previous periods remain secure, i. e. they cannot be forged.

Bellare and Miner [BM99] formalized these ideas and presented a forward secure scheme based on the hardness of factoring.

Krawczyk [Kra00] suggested the idea of using Merkle's certification tree where the leaves are the period certificate information which includes the period's public key and period's number but not a signature in order to make a conventional signature scheme a forward secure one. Maklin et al. [MMM02] construct a forward-secure signature scheme which is also based on a conventional signature scheme.

Bellare and Miner give in [BM99] the definition of a key-evolving signature scheme which consists of four process (Gen, Upd, Sig, Ver), where Gen is the key generation process, Upd is the update processes, Sig is the signature process and Ver is the verification process. The Gen receives as input the security parameter, the number of periods and maybe other information. The Upd process is called in order to update the private key.

Roughly speaking a key-evolving signature scheme is forward secure if any adversary who obtains the private key in a certain period cannot succeed in forging a signature of any of the previous periods. In this case, the adversary calls the *Upd* process any time he wants.

Definition 1.7 (key-evolving signature scheme [BM99]) *A* key-evolving signature scheme $\mathsf{KESign} = (\mathsf{KEGen}, \mathsf{KEUpd}, \mathsf{KESig}, \mathsf{KEVer})$ *consists of four algorithms.* $(\mathsf{KEGen}, \mathsf{KESig}, \mathsf{KEVer})$ *forms a signature scheme with the following modifications:*

Key generator algorithm KEGen *takes in addition as input the total number of periods T over which the scheme will operate. The returned keys PK and SK_0 are called* base public key *and* base secret key*, respectively.*

Signing algorithm KESig *takes in addition the current period α. The returned signature σ must contain the value α.*

Besides the

Secrete key update algorithm KEUpd *takes as input the secret signing key of the previous period $SK_{\alpha-1}$ to return the secret signing key of the current period SK_α.*

The secrete key update algorithm is usually deterministic.

In case that it is not clear what period the signature σ belongs to, we write $\langle \alpha, \sigma \rangle$ instead of σ as an explicit representation of the period α when the signature was made.

An advantage of a key evolving scheme is that the private key is changing during the lifetime of the instance. If in a certain period the private key is compromised, then from the current private key must not be computational feasible to reconstruct the (part of the) private key of previous periods which permits to sign messages of those periods. Roughly, this kind of security is called "forward security". We reformulate the definition given in [BM99].

Definition 1.8 *Let* $\mathsf{KESign} = (\mathsf{KEGen}, \mathsf{KEUpd}, \mathsf{KESig}, \mathsf{KEVer})$ *be a key-evolving signature scheme.* KESign *is a* (t, ϵ, n) forward-secure signature scheme*, if for any probabilistic algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ running in time at most t its advantage*

$$Adv^{fssig}_{\mathsf{KESign}}(\mathcal{A}) = Pr[ExpForge(\mathsf{KESign}, \mathcal{A}) = 1] < \epsilon,$$

where $\mathcal{A}_1$ makes a total of at most n queries to the signing oracle across all the stages and the probability is taken over the coin tosses of $Gen, Sig, \mathcal{A}_1$ and $\mathcal{A}_2$. The experiment ExpForge is described in Table 1.1.

Table 1.1: Experiment for a forward secure signature scheme

Experiment ExpForge($\mathsf{KESign}, \{\mathcal{A}_1, \mathcal{A}_2\}$)

$(SK_0, PK) \leftarrow Gen(1^s, T, I)$
$j \leftarrow 0$
$h \leftarrow \lambda$
Repeat
  $j \leftarrow j + 1$
  $SK_j \leftarrow Upd(SK_{j-1})$
  (d, h) $\leftarrow \mathcal{A}_1^{Sig(\cdot, SK_j)}(PK, h)$
Until (d = breakin) or (j=T)
If d $\neq$ breakin and $j = T$ then $j \leftarrow T + 1$
$(M, \langle b, \zeta \rangle) \leftarrow \mathcal{A}_2(SK_j, h)$
If $Ver(M, \langle b, \zeta \rangle, PK) = true$ and $1 \leq b < j$ and M was not queried of $Sig(\cdot, SK_b)$ in period b then return 1
else return 0

We adopt the convention that a secret key in the period $T+1$ is the null string, where T is the number of periods for the signature scheme.

Now, our intention is to provide forward-secure multi-time signature schemes. In contrast to a conventional signature scheme, a multi-time one has a limited number of possible signatures. An advantage of multi-time schemes is that they are much more efficient for signing or verifying than the conventional ones. In our proposals, the verification algorithm requires only the use of a hash function and, depending on the implementation of such hash function, a low cost of RAM. In our proposals we use a modified version of key-evolving signature scheme: Each period

consists of a certain number of signatures and the update algorithm is intrinsic to the signing algorithm. In this case, we can simplify the definition of the advantage of the adversary in definition 1.8.

Remark 1.1 With the notation of the definition 1.8 on page 11 the advantage $Adv^{fssig}_{\mathsf{KESign}}$ can be simplified in our case. Recall that each period consists of a certain number of signatures and the secret key update algorithm is intrinsic to the signing algorithm. Thus

$$Adv^{fssig}_{\mathsf{KESign}}(\mathcal{A}) = Pr[Ver(M, \langle b, \zeta \rangle) = true \mid \begin{array}{l} (SK_0, PK) \leftarrow Gen(1^s, T, I); \\ (info, k) \leftarrow \mathcal{A}_1^{Sig(\cdot, SK)}(PK); \\ (M, \langle b, \zeta \rangle) \leftarrow \mathcal{A}_2(SK_k, info) \\ \text{and } b < k] < \epsilon, \end{array}$$

where SK represents the evolving private key used to sign messages by the oracle and SK_k represents the private key at the intrinsic period k due to the number of created signatures.

We will make use of an auxiliary tool in order to reduce the size of the private key and preserve the security of our proposed schemes. First, we recall the definition of computational indistinguishability.

Definition 1.9 *Let $f : \mathcal{D}_f \to \mathcal{R}$ and $g : \mathcal{D}_g \to \mathcal{R}$ be two functions. We say that f and g are* (t, ϵ) computationally indistinguishable *or simple* (t, ϵ) indistinguishable *if for every D distinguishing algorithm that given $r \in \mathcal{R}$ return a bit, its advantage in distinguishing between $f(\mathcal{U}_{\mathcal{D}_f})$ and $g(\mathcal{U}_{\mathcal{D}_g})$ within time t is*

$$|Pr[D(f(\mathcal{U}_{\mathcal{D}_f})) = 1] - Pr[D(g(\mathcal{U}_{\mathcal{D}_g})) = 1]| < \epsilon.$$

Here $\mathcal{U}_{\mathcal{X}}$ is the uniform distribution on $\mathcal{X}$. The probability is taken on the coin tosses of D.

In case that a function $h : \mathcal{R} \to \mathcal{R}$ is the identity, we will simply write $\mathcal{U}_{\mathcal{R}}$ instead of $h(\mathcal{U}_{\mathcal{R}})$.

Next, we recall the definition of pseudorandom bit generator (prg) and the definition introduced by Bellare and Yee in [BY03] concerning a forward-secure

pseudorandom bit generator (fsprg). Roughly speaking, a pseudorandom bit generator $g : \{0,1\}^l \to \{0,1\}^{l+l'}$ is *cryptographically secure* if $g(\mathcal{U}_l)$ and $\mathcal{U}_{l+l'}$ are computationally indistinguishable, where $\mathcal{U}_\alpha$ denote the uniform distribution on $\{0,1\}^\alpha$.

We reformulate the definition given in [BY03].

Definition 1.10 *Let $g : \{0,1\}^l \to \{0,1\}^{l+l'}$ be a function. We say that g is a* (t,ϵ) pseudorandom bit generator *if g and $\mathcal{I}_{l+l'}$ are (t,ϵ) indistinguishable. Here $\mathcal{I}_{l+l'}$ is the identity function on $\{0,1\}^{l+l'}$.*

A stateful generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, b, n)$ consists of a pair of algorithms and a pair of positive integers. $\mathcal{G}.key$ is a probabilistic algorithm which takes no inputs and outputs a bit string called initial state. $\mathcal{G}.key$ is called the *key generation algorithm* and the initial state is also called *seed*. $\mathcal{G}.next$ is a deterministic algorithm which given the current state, returns a pair of bit strings consisting of an *output block* and the *next state*. $\mathcal{G}.next$ is called *next step algorithm* and the output block is a b-bit string. The generator $\mathcal{G}$ may use up to n output blocks to produce a sequence $Out_0, \ldots, Out_{n'}$ for $0 \leq n' < n$ as follows: first $St_{-1} \leftarrow \mathcal{G}.key$ and then $(Out_i, St_i) \leftarrow \mathcal{G}.next(St_{i-1})$ for $0 \leq i \leq n'$.

Definition 1.11 *A stateful generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, b, n)$ is a* (t,ϵ) forward-secure pseudorandom bit generator*, if for any probabilistic algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ running in time at most t*

$$Adv_{\mathcal{G}}^{fsprg}(t) = \left| Pr[Exp_{\mathcal{G}}^{fsprg\text{-}1}(\mathcal{A}) = 1] - Pr[Exp_{\mathcal{G}}^{fsprg\text{-}0}(\mathcal{A}) = 1] \right| < \epsilon,$$

where $Exp_{\mathcal{G}}^{fsprg\text{-}1}$ and $Exp_{\mathcal{G}}^{fsprg\text{-}0}$ are described in Table 1.2 on the next page. The probability is taken over the coins tosses of $\mathcal{G}.key, \mathcal{A}_1$ and $\mathcal{A}_2$.

In Proposition 1.1 a construction of a forward-secure pseudorandom bit generator from a cryptographically secure one is shown.

Proposition 1.1 (Proven in [BY03]) *Let $G : \{0,1\}^s \to \{0,1\}^{s+b}$ be a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator, let n be an integer such that $2n\epsilon_{prg} \leq 1$. Define $\mathcal{G} =$*

Table 1.2: Experiments for the forward secure pseudorandom bit generator

Experiment $\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A})$	Experiment $\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A})$
$St_{-1} \leftarrow \mathcal{G}.key$	$St_{-1} \leftarrow \mathcal{G}.key$
$i \leftarrow -1$; $h \leftarrow \lambda$	$i \leftarrow -1$; $h \leftarrow \lambda$
Repeat	Repeat
$i \leftarrow i+1$	$i \leftarrow i+1$
$(St_i, Out_i) \leftarrow \mathcal{G}.next(St_{i-1})$	$(St_i, Out_i) \leftarrow \mathcal{G}.next(St_{i-1})$
$(d, h) \leftarrow \mathcal{A}_1(Out_i, h)$	$Out_i \leftarrow \mathcal{U}_b$
Until (d = guess) or (i=n-1)	$(d, h) \leftarrow \mathcal{A}_1(Out_i, h)$
$g \leftarrow \mathcal{A}_2(St_i, h)$	Until (d = guess) or (i=n-1)
Return g	$g \leftarrow \mathcal{A}_2(St_i, h)$
	Return g

Here λ is the null string and $\mathcal{U}_b$ is the uniform distribution on the set of bit strings of length b.

$(\mathcal{G}.key, \mathcal{G}.next, b, n)$, where $\mathcal{G}.key$ outputs $St \in_R \{0,1\}^s$ and on input St, $\mathcal{G}.next$ obtains $X \leftarrow G(St)$ and outputs (St', Out), where $X = St' \| Out$, $St' \in \{0,1\}^s$ and $Out \in \{0,1\}^b$. Then $\mathcal{G}$ is a $(t_{fsprg}, \epsilon_{fsprg})$ forward secure pseudorandom bit generator, where $\epsilon_{fsprg} = 2n\epsilon_{prg}$ and $t_{prg} = t_{fsprg} + O(n \cdot (s+b))$.

1.3 Improvement to the LDots

In this Section we recall the Lamport-Diffie one-time signature scheme (LDots). We provide an improved version of LDots, prove its security and show some experimental results about its efficiency.

Let $H : \{0,1\}^* \to \{0,1\}^s$ be a hash function. Next we give the description of the LDots suggested by Merkle in [Mer90]. This scheme has as security parameter s, which is the length of the output of the underlying hash function. We can assume that the hash function restricted to bit strings of size s is a one-way function.

Key generation algorithm Gen. Gen chooses $x_i \in_R \{0,1\}^s$ and computes $y_i =$

$H(x_i)$ for $0 \leq i < s'$, where $s' = s+1+\lfloor \log_2 s \rfloor$. Gen outputs the pair (X, Y), where $X = (x_0, \ldots, x_{s'-1})$ and $Y = (y_0, \ldots, y_{s'-1})$. X is the secret key and Y is the verification key.

Signature algorithm Sig. Given as input a message $M \in \{0,1\}^s$ and the secret key $X = (x_0, \ldots, x_{s'})$, Sig computes $M \parallel Z = m_0 \cdot \ldots \cdot m_{s'-1}$, where Z is the $1+\lfloor \log_2 s \rfloor$-bit representation of the number of zeros in the bit representation of M. Sig outputs $\tau = (x_{i_1}, \ldots, x_{i_k})$, where $0 \leq i_1 < \cdots < i_k < s'$ and $m_j = 1$ iff $j \in \{i_1, \ldots, i_k\}$. τ is the signature.

Verification algorithm Ver. Given as input the message M, a signature $\tau' = (z_1, \ldots z_l)$ and a verification key $Y = (y_0, \ldots, y_{s'-1})$, Ver computes $m_0 \cdot \ldots \cdot m_{s'-1}$ from M as in the signature process. Ver outputs *true* if $l = k$ and $y_{i_j} = H(z_j)$ for $j \in \{\alpha \mid m_{i_\alpha} = 1\}$ and k is obtained as in the signature process. Ver outputs *false* otherwise.

On the efficiency of LDots The sizes of the private and public key are equal to $ss' = s^2 + s(1 + \lfloor \log_2 s \rfloor)$ bits. The size of the signature is ls, where $1 \leq l \leq s$ which depends on the message to be signed. Let t_h and t_{rg} be the time needed in average to compute a hash value and an s-bit random string, respectively. The time needed to generate a key pair is $s'(t_h + t_{rg})$. The time needed to sign a message is lt_h, where $1 \leq l \leq s$.

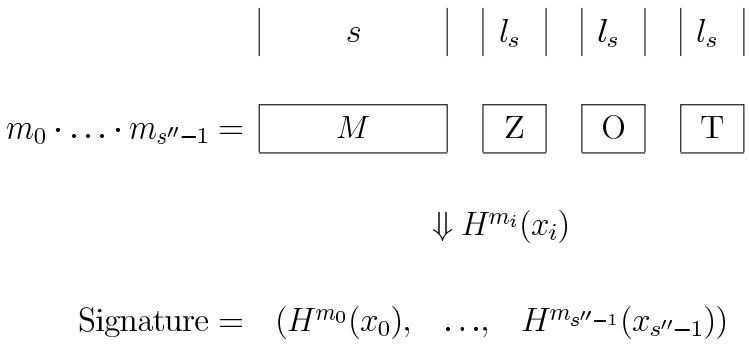
On the security of LDots Intuitively, if an adversary wants to forge a signature with the only knowledge of the public key and maybe a signed message, then the adversary must compute at least one pre-image of the function H for a randomly chosen input. Suppose that $M \parallel Z = m_0 \cdot \ldots \cdot m_{s'}$, then if $M = 0^s$, then the string Z, which is employed to count the number of zero bits in M, has at least a non zero bit. Now, if the adversary obtains a message M' with a forged signature $\tau' = (z_0, \ldots, z_k)$, we deduce the following: We set $m_0 \cdot \ldots \cdot m_{s'} \leftarrow M \parallel Z$ and $m'_0 \cdot \ldots \cdot m'_{s'} \leftarrow M' \parallel Z'$, where Z and Z' are the number of zero bits of M and M', respectively. We know that $M \neq M'$, therefore there exists $0 \leq j < s$ such that $m_j \neq m'_j$. If $m_j = 1$ and $m'_j = 0$, then the adversary should obtain x' such that $y_j = H(x')$ and give such x' as part of the signature. If for $0 \leq j < s$ any time

that $m_j \neq m'_j$ we have that $m_j = 0$ and $m'_j = 1$, then the number of zero bits in M' is lower than the one in M. Therefore $Z > Z'$ regarded as integers represented with $1 + \lfloor \log_2 s \rfloor$ bits. Thus there exists $s \leq j < s'$ such that $m_j = 1$ and $m'_j = 0$, which implies that the adversary should obtain x' such that $y_j = H(x')$ in order to give it as part of the signature.

Our improvement We reduce the size of the private and the public keys and fix the length of the signature.

Let H be as before and set $s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil$. If M is the message to be signed, we represent M as quaternary digits (quits). Let Z, O and T be the quit string representation of the number of zeros, ones and twos in the representation of M as quits, respectively. We use the concatenation $M\|Z\|O\|T$ to construct our signature in a similar way as it was previously presented. In Figure 1.1 this construction is depicted.

Figure 1.1: Representation of the construction of the auxiliary string $M\|Z\|O\|T$ for signing a message M. Here $l_s = \lfloor \log_2 s \rfloor$ and $0 \leq m_i < 4$ for all $0 \leq i < s''$.



Next we describe our first improvement to the Lamport-Diffie one-time signature scheme. In this version the key pair is randomly generated. As before, the security parameter s is the length of the output of the underlying hash function.

Scheme 1.12 Improved Lamport-Diffie one-time signature scheme with randomly generated key pair (*iLDots with random key*).

Key generation algorithm Gen. Gen chooses $x_i \in \{0,1\}^s$ at random for $0 \leq i < s''$, where $s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil$. Gen outputs $Y = H(H^3(x_0), \ldots, H^3(x_{s''-1}))$ as the verifying key and $X = (x_0, \ldots, x_{s''-1})$ as the private key.

Signature algorithm Sig. Given as input a message M and the private key $X = (x_0, \ldots, x_{s''-1})$, Sig computes $m_0 \cdot \ldots \cdot m_{s''-1} = M||Z||O||T$ as quits from M, where Z, O and T are the bit string representation of zeros, ones and twos, respectively, of the quits representation of M. Sig outputs the signature $\tau = (H^{m_0}(x_0), \ldots, H^{m_{s''-1}}(x_{s''-1}))$.

Verification algorithm Ver. Given as input a message M, a signature τ' and a public key Y, Ver computes $M||Z||O||T = m_0 \cdot \ldots \cdot m_{s''-1}$ as it is computed in the signature process. Suppose that $\tau' = (z_0, \ldots, z_{s''-1})$. Ver outputs *true* if $Y = H(H^{3-m_0}(z_0), \ldots, H^{3-m_{s''-1}}(z_{s''-1}))$ and $false$ in other case.

In this version the public and private keys are reduced to s bits and to ss'' bits, respectively. The signature is fixed to ss'' bits. Note that

$$s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil < \frac{s+3\lfloor \log_2 s \rfloor}{2} + 1 = \frac{s+1+\lfloor \log_2 s \rfloor}{2} + \frac{1}{2} + \lfloor \log_2 s \rfloor = \frac{s'+1}{2} + \lfloor \log_2 s \rfloor$$

Now we give a version of the LDots with a deterministic key generation algorithm. In this version the private key is reduced to s bits. As before, the security parameter s is the length of the output of the underlying hash function.

Scheme 1.13 Improved Lamport-Diffie one-time signature scheme with deterministic key generation algorithm (*iLDots with pseudorandom key*).

Deterministic key generation algorithm Gen. Given as input a *seed* $g_{-1} \in \{0,1\}^s$, Gen computes $(g_i, x_i) \leftarrow PRG(g_{i-1})$ for $0 \leq i < s''$. Gen outputs g_{-1} as the private key and $Y = H(H^3(x_0), \ldots, H^3(x_{s''-1}))$ as the verifying key.

Signature algorithm Sig. Sig remains the same as in Scheme 1.12 on the preceding page except that x_i is computed from g as in Gen for $0 \leq i < s''$.

Verification algorithm Ver. Ver remains the same as in Scheme 1.12 on the previous page.

1.3.1 Security

Next we give an intuitive idea of the security of the iDLots with random key. Then we give a formal proof of the security of the provided improvements of the LDots.

Roughly speaking, if an adversary gives a forged signature with only the knowledge of the public key and maybe a signed message, then the adversary should find a pre-image or a collision of the underlying hash function. Recall that the private key consists of random bit strings of size s. The message M to be signed has a size of s bits and three counters are taken into account for the signature. Such counters contain the number of zeros, ones and twos in the quit representation of M. The signature consists of s'' bit strings of size s. The adversary could try to use a signed message to create a forgery, since he is able to modify the message. If the quit m'_j of the new message M' is lower that the corresponding quit m_j of M for some j, i. e., $m'_j < m_j$, then the adversary needs to be able to obtain a pre-image of $H^{m_j}(x_j)$ or a collision of H in order to give a forged signature. Recall that the adversary knows $z_j = H^{m_j}(x_j)$ and he must give z'_j as part of the forged signature, such that $H^{3-m_j}(z_j) = H^{3-m'_j}(z'_j)$ or $H(\ldots, H^3(x_j), \ldots) = H(\ldots, H^{3-m'_j}(z'_j), \ldots)$. On the other hand, if $m'_j > m_j$ for all j such that $m_j \neq m'_j$ in the quit representation of M and M', then at least one of the three counters related with M' must be lower than the corresponding counter related with M, i.e. the number of zeros, ones or twos in M is lower than the corresponding one in M'. Therefore there must exist m_j and m'_j in the quit representation of such counter of M and M', respectively, such that $m'_j < m_j$. Thus, as before, the adversary should be able to obtain a pre-image of $H^{m_j}(x_j)$ or a collision of H in order to give a forged signature.

Our goal is to prove that the existence of a forger of the iDLots with random key implies the existence of an efficient algorithm for finding either a collision or a pre-image of a random input for the underlying hash function H. As you can see in the previous paragraph, the forger must obtain a pre-image of some hash value, not necessary from a random input, i. e. such a hash value could be one of $H^3(x)$ and $H^2(x)$ for some $x \in_R \{0,1\}^s$. In this case we require that for a random x, $H(x)$ and $H^2(x)$ be computationally indistinguishable and so are $H(x)$ and $H^3(x)$.

In Proposition 1.2 we show that the Scheme 1.12 on page 17 is a one-time

signature scheme. Its security is based on its underlying hash function H. In Proposition 1.3 on page 23 we show that the Scheme 1.13 on page 18 is a one-time signature scheme. This time, its security is based on its underlying hash function H and its cryptographically secure pseudorandom bit generator prg.

Proposition 1.2 *Let $H : \{0,1\}^* \to \{0,1\}^s$ be a (t_{ow}, ϵ_{ow}) one-way and a (t_{cr}, ϵ_{cr}) collision resistant hash function, such that (i) H_1 and H_2 are (t_{in}, ϵ_{in}) indistinguishable, and H_1 and H_3 are (t_{in}, ϵ_{in}) indistinguishable, where $\epsilon_{in} \leq \frac{4}{3}\min\{\epsilon_{cr}, \epsilon_{ow}\}$ (ii) $24s''\max\{\epsilon_{cr}, \epsilon_{ow}\} \leq 1$ and $t_{ow}, t_{cr}, t_{in} \geq t + t_S + t_G + 5t_h$ for some t, where t_S is the time needed to sign a message, t_h is the time needed to compute a hash value and t_G is the time needed to compute a pair of keys. Here $H_i : \{0,1\}^s \to \{0,1\}^s$ is a function defined as $H_i(x) = H^i(x)\ \forall x \in \{0,1\}^s$ and $1 \leq i \leq 3$. Let $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be the iLDots with random key as described in Scheme 1.12 on page 17, then Sign is a (t, ϵ) one-time signature, where $\epsilon = 24s''\max\{\epsilon_{ow}, \epsilon_{cr}\}$.*

Proof

The idea of the proof is as follows. If Sign is not secure, then there exists a forger $\mathcal{F}$ and with help of this forger we are able to construct an algorithm $\mathcal{A}$ for computing either a pre-image of a given y or a collision of H. Thus, given y as input, $\mathcal{A}$ computes a random instance of the iLDots including y in certain part of that process. $\mathcal{A}$ obtains a public key Y and a large part off the corresponding private key X. $\mathcal{A}$ calls the forger $\mathcal{F}$ with Y as input and uses X and y in case that $\mathcal{F}$ asks to sign the only one allowed query. With high probability, $\mathcal{F}$ returns a forgery, i. e. a message M' and a forged signature τ' such that $\mathsf{Ver}(M', \tau', Y) = true$. τ' is used by $\mathcal{A}$ in order to obtain with high probability either a pre-image of y or a collision of H.

Now, if Sign is not a one-time signature scheme, i. e. if Sign is existentially forgeable under a one adaptive chosen message attack, then we are able to provide an algorithm $\mathcal{A}$ which efficiently finds either a collision of H or a pre-image of $H(x)$ for a randomly chosen $x \in \{0,1\}^s$. Here H is the underlying hash function of Sign.

Suppose that there exist a forger algorithm $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ such that

$$Pr[\mathsf{Ver}(\mathcal{F}_2(T), Y) = true \,|\, (X,Y) \leftarrow \mathsf{Gen};\ T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot,X)}(Y)] \geq \epsilon$$

$\mathcal{A}$ works as follows. **Generating a random instance of the iLDots:** Given as input $y = H(x)$ for some $x \in_R \{0,1\}^s$, $\mathcal{A}$ chooses $0 \leq i_0 < s''$ and $1 \leq m \leq 3$ at random. Then $\mathcal{A}$ sets $y_{i_0} := H^{3-m}(y)$, chooses $x_i \in_R \{0,1\}^s$ and computes $y_i = H^3(x_i)$ for $i \neq i_0$ and $0 \leq i < s''$. The public key is $Y = H(y_0, \ldots, y_{s''-1})$.

Obtainment of a forgery through the employment of $\mathcal{F}$: $\mathcal{A}$ calls $\mathcal{F}$ in order to obtain a forgery with probability of at least ϵ. Let M' and τ' be the forgery. We write $\tau' = (z_0, \ldots, z_{s''-1})$ and $m'_0 \cdot \ldots \cdot m'_{s''-1} = M' \| Z' \| O' \| T'$, where Z', O' and T' are the counters of zeros, ones and twos regarded as quit strings. At this point we make a distinction. What happens if $\mathcal{F}$ does not make a query? In that case note that the cardinality of the set $\{j \mid m'_j = 0 \text{ for } 0 \leq j < s''\}$ is greater than or equal to 1, since $Z' \neq 0$ as the counter of zeros implies that a quit in M' must be zero. Thus the probability that $m'_{i_0} = 0$ is at least $\frac{1}{s''}$. **Finding a pre-image without a query from $\mathcal{F}$:** If $H^m(z_{i_0}) = y$, then $\mathcal{A}$ outputs $H^{m-1}(z_{i_0})$ as pre-image of y. **Finding a collision without a query from $\mathcal{F}$:** Otherwise, $\mathcal{A}$ computes $x'_j = H^{j-m}(y), z'_j = H^j(z_{i_0})$ for $m \leq j \leq 3$, $x'_4 = y_0 \| \cdots \| y_{s''-1}$ and $z'_4 = H^{3-m_0}(z_0) \| \cdots \| H^3(z_{i_0}) \| \cdots \| H^{3-m_{s''-1}}(z_{s''-1})$ and finally computes $x'_5 = H(x'_4)$ and $z'_5 = H(z'_4)$. Recall that $x'_5 = z'_5$ since τ' is a forgery and at this point we assume that $x'_m = y \neq H^m(z_{i_0}) = z'_m$. Therefore, there exists j_0 such that $x'_{j_0} \neq z'_{j_0}$ and $x'_j = z'_j \; \forall j > j_0$ and then $\mathcal{A}$ outputs (x'_{j_0}, z'_{j_0}) as a collision. Thus, in case that $\mathcal{F}$ does not require any signature we have the following. Set $p = Pr[y = H^m(z_{i_0}) \mid \tau' \leftarrow \mathcal{F}]$. If $p \geq \frac{1}{2}$, then $\mathcal{A}$ succeeds in finding a pre-image $H^{m-1}(z_{i_0})$ of $y = H(x)$ with probability of at least $\frac{\epsilon}{2s''} \geq \epsilon_{ow}$ within time t_{ow}. On the other hand, if $p < \frac{1}{2}$, then $\mathcal{A}$ succeeds in finding a collision of H with probability of at least $\frac{\epsilon}{2s''}$ within time t_{cr}.

Now, what happens if $\mathcal{F}$ does make a query? In that other case, we denote by M the message to be signed and by $m_0 \cdot \ldots \cdot m_{s''-1}$ the quits of the concatenation of $M \| Z \| O \| T$, where Z, O and T are the counters of zeros, ones and twos regarded as quit strings. The condition on M and M' implies that $M \neq M'$, and so there exists an ι such that $m_\iota \neq m'_\iota$.

Notice that there exists $0 \leq j_0 < s''$ such that $0 \leq m'_{j_0} < m_{j_0} \leq 3$. Indeed, if $m'_\iota < m_\iota$ we are done. Else, first let us define the following sets $B_k = \{j \mid m_j =$

k; m_j in the quit representation of $M\}$ and $B'_k = \{j \mid m'_j = k$; m'_j in the quit representation of $M'\}$ for $0 \leq k \leq 3$. Note that the cardinality of B_0, B_1 and B_2 are the numbers represented by Z, O and T, respectively. Analogously we have the relation between B'_0, B'_1 and B'_2 with Z', O' and T'. Thus, if $m_j < m'_j$ for all m_j and m'_j in the quit representation of M and M', respectively, we have that the cardinality of B_κ is greater than the cardinality of B'_κ, where $\kappa = \min\{m_j \mid m_j < m'_j;\ 0 \leq j < \frac{s}{2}\}$. In this case, the corresponding counter of B_κ and the one of B'_κ contain one each of them a quit m_{j_0} and m'_{j_0}, respectively, such that $m_{j_0} > m'_{j_0}$ as affirmed. In this case, the probability that $i_0 = j_0$ is at least $\frac{1}{s''}$ and the probability that $m = m_{j_0}$ is $\frac{1}{3}$.

If y were obtained from $H_m(\mathcal{U}_s)$, where $\mathcal{U}_s$ is the uniform distribution on $\{0,1\}^s$, then the adversary would have obtained a public key and a signature as described in Scheme 1.12. In that case, the probability that $i_0 = j_0$ and $m = m_{j_0}$ given a forgery by $\mathcal{F}$ is at least $\frac{\epsilon}{3s''}$, i. e. $Pr[i_0 = j_0$ and $m_{j_0} = m \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_m(\mathcal{U}_s)$; $m \leftarrow \{1,2,3\}] \geq \frac{\epsilon}{3s''}$. Recall that y is obtained from $H_1(\mathcal{U}_s)$ and not from $H_m(\mathcal{U}_s)$, if $m = 2, 3$. In this case we assure that the probability of $i_0 = j_0$ and $m = m_{j_0}$ given a forgery by $\mathcal{F}$ is at least $\frac{\epsilon}{3s''} - 3\epsilon_{in}$, i. e. $Pr[i_0 = j_0$ and $m_{j_0} = m \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_1(\mathcal{U}_s)$; $m \leftarrow \{1,2,3\}] \geq \frac{\epsilon}{3s''} - 3\epsilon_{in}$. To prove this assertion we will build a distinguisher $\mathcal{D}_{m'}$ of H_1 and $H_{m'}$ employing $\mathcal{F}$ under the assumption that $Pr[i_0 = j_0$ and $m_{j_0} = m \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_1(\mathcal{U}_s)$; $m \leftarrow \{1,2,3\}] < \frac{\epsilon}{3s''} - 3\epsilon_{in}$. Here $m' = 2, 3$. Basically $\mathcal{D}_{m'}$ constructs a key as $\mathcal{A}$ does and uses $\mathcal{F}$ in order to obtain a forgery. If $i_0 = j_0$ and $m_j = m'$, then $\mathcal{D}_{m'}$ returns 1. $\mathcal{D}_{m'}$ returns 0 otherwise. In this case we have that $Pr[\mathcal{D}_{m'}(H_1(\mathcal{U}_s)) = 1] = Pr[i_0 = j_0$ and $m_{j_0} = m' \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_1(\mathcal{U}_s)$; $m = m'$; $m \leftarrow \{1,2,3\}] = Pr[i_0 = j_0$ and $m_{j_0} = m \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_1(\mathcal{U}_s)$; $m \leftarrow \{1,2,3\}]Pr[m = m'] < \frac{1}{3}(\frac{\epsilon}{3s''} - \epsilon_{in})$, within time t_{in}. On the other hand $Pr[\mathcal{D}_{m'}(H_{m'}(\mathcal{U}_s)) = 1] = Pr[i_0 = j_0$ and $m_{j_0} = m' \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_{m'}(\mathcal{U}_s)$; $m = m'$; $m \leftarrow \{1,2,3\}] = Pr[i_0 = j_0$ and $m_{j_0} = m \mid$ forgery $\leftarrow \mathcal{F}$; $\mathcal{F}$ makes one query; $y \leftarrow H_m(\mathcal{U}_s)$; $m \leftarrow \{1,2,3\}]Pr[m = m'] \geq \frac{1}{3}\frac{\epsilon}{3s''}$, within time t_{in}. Therefore, we have that

$$Pr[\mathcal{D}_{m'}(H_1(\mathcal{U}_s)) = 1] - Pr[\mathcal{D}_{m'}(H_{m'}(\mathcal{U}_s)) = 1] > \epsilon_{in},$$

within time t_{in}. That contradicts that H_1 and $H_{m'}$ are (t_{in}, ϵ_{in}) computationally

indistinguishable.

Now as before we obtain from $\mathcal{F}$ a forgery M' and τ' and we write $\tau' = (z_0, \ldots, z_{s''-1})$ and $m'_0 \cdot \ldots \cdot m'_{s''-1} = M'\|Z'\|O'\|T'$, where Z', O' and T' are the counters of zeros, ones and twos regarded as quit strings. **Finding a pre-image:** If $H^{m-m'_{i_0}}(z_{i_0}) = y$, then $\mathcal{A}$ outputs $H^{m-m'_{i_0}-1}(z_{i_0})$ as pre-image of y. **Finding a collision:** Otherwise, $\mathcal{A}$ computes $x'_j = H^{j-m}(y), z'_j = H^{j-m'_{i_0}}(z_{i_0})$ for $m \leq j \leq 3$, $x'_4 = y_0\| \cdots \|y_{s''-1}$ and $z'_4 = H^{3-m_0}(z_0)\| \cdots \|H^{3-m'_{i_0}}(z_{i_0})\| \cdots \|H^{3-m_{s''-1}}(z_{s''-1})$ and finally computes $x'_5 = H(x'_4)$ and $z'_5 = H(z'_4)$. Recall that $x'_5 = z'_5$ since τ' is a forgery and at this point we assume that $x'_m = y \neq H^{m-m'_{i_0}}(z_{i_0}) = z'_m$. Therefore, there exists j_0 such that $x'_{j_0} \neq z'_{j_0}$ and $x'_j = z'_j \; \forall j > j_0$ and then $\mathcal{A}$ outputs (x'_{j_0}, z'_{j_0}) as a collision. Thus in case that $\mathcal{F}$ does require a signature and $m = 1$ we have the following. Set $p = Pr[y = H^{m-m'_{i_0}}(z_{i_0}) \mid \tau' \leftarrow \mathcal{F}]$, if $p \geq \frac{1}{2}$, then $\mathcal{A}$ succeeds in finding a pre-image $H^{m-m'_{i_0}-1}(z_{i_0})$ of $y = H(x)$ with probability of at least $\frac{\epsilon}{6s''} - \frac{3\epsilon_{in}}{2} \geq \epsilon_{ow}$ within time t_{ow}. On the other hand, if $p < \frac{1}{2}$, then $\mathcal{A}$ succeeds in finding a collision of H with probability of at least $\frac{\epsilon}{6s''} - \frac{3\epsilon_{in}}{2} \geq \epsilon_{cr}$ within time t_{cr}.

Finally, let $\rho = Pr[\mathcal{F}$ does not require a signature $]$. Recall that $\mathcal{F}$ outputs a forgery with probability at least ϵ. Therefore, if $\rho \geq \frac{1}{2}$, then $\mathcal{A}$ succeeds in finding a pre-image or a collision with probability of at least $\frac{\epsilon}{4s''} \geq \max\{\epsilon_{ow}, \epsilon_{cr}\}$ within time $t \leq \min\{t_{ow}, t_{cr}\}$. Now, in case that $\rho < \frac{1}{2}$ and $m = 1$, then $\mathcal{A}$ succeeds in finding a pre-image or a collision with probability of at least $\frac{\epsilon}{12s''} - \frac{3\epsilon_{in}}{4} \geq \max\{\epsilon_{ow}, \epsilon_{cr}\}$ within time $t \leq \min\{t_{ow}, t_{cr}\}$.

□

Proposition 1.3 *Let $H : \{0,1\}^* \to \{0,1\}^s$ be a (t_{ow}, ϵ_{ow}) one-way and a (t_{cr}, ϵ_{cr}) collision resistant hash function and let PRG be a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator, such that (i) $4s'' \max\{12\epsilon_{cr}, 12\epsilon_{ow}, \epsilon_{prg}\} \leq 1$, (ii) $t_{ow}, t_{cr} \geq t + t_S + t_G + 5t_h$, (iii) $t_{prg} = t + t_G + t_V + O(2ss'')$ for some t and (iv) H_1 and H_2 are (t_{in}, ϵ_{in}) indistinguishable, and H_1 and H_3 are (t_{in}, ϵ_{in}) indistinguishable, where $\epsilon_{in} \leq \frac{4}{3}\min\{\epsilon_{cr}, \epsilon_{ow}\}$. Here t_h is the time needed to compute an H value, t_G is the time needed to compute a pair of keys, t_S is the time needed to sign a message and t_V is the time needed to verify a signature. H_i is defined as in Proposition 1.2, where $i = 1, 2, 3$. If $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is as described in Scheme 1.13, then Sign is a (t, ϵ) one-time signature with deterministic generation key algorithm, where*

$\epsilon = 4s'' \max\{12\epsilon_{ow}, 12\epsilon_{cr}, \epsilon_{prg}\}$.

Proof

The idea of the proof is as follows. In Proposition 1.2 is shown that the iLDots with random key is secure if the underlying hash function is secure, as well. If the iLDots with pseudorandom key is existentially forgeable under a one adaptive chosen message attack by a forger $\mathcal{F}$, we are able to construct a distinguisher for the forward-secure pseudorandom bit generator obtained from PRG. The distinguisher $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ works as follows: $\mathcal{A}_1$ collects all the s'' values he is allowed to receive. $\mathcal{A}_1$ passes those values to $\mathcal{A}_2$ and $\mathcal{A}_2$ generates from those values a key pair (X, Y). $\mathcal{A}_2$ gives the public key Y to the forger $\mathcal{F}$ and maybe uses the private key X to sign the only one allowed query of $\mathcal{F}$. If $\mathcal{F}$ returns a forgery, $\mathcal{A}_2$ returns 1, which means that with "high" probability the s'' values were obtained by the forward-secure pseudorandom bit generator, since it is supposed that the iLDots with pseudorandom key is not secure. Otherwise, $\mathcal{A}_2$ returns 0, which means that with "high" probability the s'' values were obtained at random, since it is proven that the iLDots with random key is secure.

Now, suppose that the signature scheme is not (t, ϵ) one-time. Using the notation of definition 1.6 on page 9, there exists an algorithm $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ such that

$$Pr[\mathsf{Ver}(M, \sigma', Pu) = 1 \quad \Big| \quad (Pr, Pu) \leftarrow \mathsf{Gen}; \; T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, Pr)}(Pu); \; (M, \sigma') \leftarrow \mathcal{F}_2(T)] \geq \epsilon$$

within time t.

Under the assumption that H is a (t_{ow}, ϵ_{ow}) one-way and a (t_{cr}, ϵ_{cr}) collision resistant hash function, we construct a distinguisher $\mathcal{A}$ for the $(t_{fsprg}, \epsilon_{fsprg})$ forward-secure pseudorandom bit generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, s, s'')$ obtained from PRG, where $t_{fsprg} = t + t_g + t_v$, $\epsilon_{fsprg} = 2s''\epsilon_{prg}$, and $\mathcal{G}$ is as in Proposition 1.1 on page 14.

We construct the distinguisher $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ as follows: h is initialized as the null string λ. Given as input Out and h, $\mathcal{A}_1$ updates $h \leftarrow h||Out$ and outputs $(\texttt{not\_guess}, h)$. **Generating a random instance of iLDots either with random key or with pseudorandom key:** Given as input St and h, $\mathcal{A}_2$ obtains Out_i from h for $0 \leq i < s''$ and computes $Y = H(H^3(Out_0), \ldots, H^3(Out_{s''-1}))$

as the verifying key and retains $X = (Out_0, \ldots, Out_{s''-1})$ as the secret key. **Using $\mathcal{F}$ to try to obtain a forgery:** $\mathcal{A}_2$ calls $\mathcal{F}_1(Y)$ and uses X for signing the unique allowed query from $\mathcal{F}_1$. $\mathcal{A}_2$ uses the output T from $\mathcal{F}_1$ as input for $\mathcal{F}_2$ in order to try to obtain a forgery (M, τ). **Distinguishing the input between the pseudorandom bit generator and the uniform distribution:** If $\mathcal{A}_2$ obtains a forged signature, then outputs 1. In other case it outputs 0.

Note that in $\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the iLDots with random key and that in $\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the iLDots with pseudorandom key.

Now, because the assumption of the existence of $\mathcal{F}$, we have that

$$Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] \geq \epsilon,$$

within time t_{fsprg}.

Because of the assumption on H, we have that

$$Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] < \tfrac{\epsilon}{2},$$

within time t_{fsprg}, since Sign is $(t, \frac{\epsilon}{2}, 1)$ existentially unforgeable under adaptive chosen message attacks as proven in Proposition 1.2 on page 20.

Therefore,

$$Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] - Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] > \tfrac{\epsilon}{2} \geq 2s''\epsilon_{prg} = \epsilon_{fsprg}$$

within time t_{fsprg}, which contradicts Proposition 1.1 on page 14

□

1.3.2 Efficiency

Now we show the efficiency of these ots. Let $h : \{0,1\}^* \to \{0,1\}^s$ be a hash function. In Table 1.3 on the following page we estimate the size of the signatures and keys and the timing for the presented versions of the Lamport-Diffie ots. In that Table t_{rg}, t_{prg}, t_h and t_H are the times for computing a random number of s bits, a pseudorandom number of s bits, a hash value whose input is s bits and a hash value whose input is ss'' bits, respectively. We set $s' = s + 1 + \lfloor \log_2 s \rfloor$, $s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil$.

Table 1.3: Comparison of the versions of the Lamport-Diffie ots scheme.

	Merkle's version	iDLots with random key	iDLots with pseudorandom key
	Size		
Private Key	ss'	ss''	s
Public Key	ss'	s	s
Signature	js	ss''	ss''
	Timing		
Key Generation	$s'(t_{rg} + t_h)$	$s''(t_{rg} + 3t_h)$	$t_{rg} + s''(t_{prg} + 3t_h)$
Signing	$jt_h + t_H$	$lt_h + t_H$	$s''(t_{prg} + k) + t_H$
Verifying	jt_h	$ks'' + t_H$	$ks'' + t_H$

Here $1 \leq j \leq s$; $t_h \leq k \leq 3t_h$ and $s'' \leq l \leq 3s''$.

1.3.3 Experimental Results

Now we present a practical version of our proposal. We have implemented the iLDots with pseudorandom key as described in Scheme 1.13 on page 18. The bit length of the keys and of the signature are shown in Table 1.4. The time needed for the key generation, signature and verification algorithms are shown in Table 1.5 on the next page. These computations were made on a Pentium `III`, SuSE 9.3, at 1.1GHz. We have used the implementation of several hash functions from OpenSSL [Ope] version 0.9.8. and the implementation of the pseudorandom bit generator ISAAC from [ISA].

Table 1.4: Size of the private (Pr) and public (Pu) keys and of the signature of the iLDots with pseudorandom key.

size					
bit length s	160	224	256	384	512
Pu/Pr Key (in bytes)	20	28	32	48	64
Signature (in Kb)	1.78	3.36	4.38	9.56	16.9

Table 1.5: Time needed for elements of the iLDots with pseudorandom key.

	ripemd160	sha				
	160	160	224	256	384	512
	time in milliseconds					
one Hash computation	0.007	0.01	0.01	0.012	0.026	0.04
one PRG computation	0.028					
Key Generation	2.26	2.18	4.79	4.14	27.3	32.1
Signature	2.54	2.23	3.89	4.17	26.9	32.6
Verification	0.24	0.223	0.635	0.695	5.5	8.03

Let H be the underlying hash function and let s be the bit length of its output. Now, suppose that the birthday-paradox attack is the best for finding a collision or a pre-image of H. We know that the probability of success in that case is $< \frac{1}{2}$ through $2^{\frac{s}{2}}$ H computations on random values. In Table 1.6 we show the security of the iLDots with pseudorandom key, we have considered $t_{\text{hash}} = 2^{\frac{s}{4}}$.

Table 1.6: Security of iLDots with pseudorandom key.

	security				
s	160	224	256	384	512
$t_{\text{hash}}/\epsilon_{\text{hash}}$	2^{81}	2^{113}	2^{129}	2^{193}	2^{257}
$t_{\text{ots}}/\epsilon_{\text{ots}}$	$2^{68.9}$	$2^{100.4}$	$2^{116.2}$	$2^{179.7}$	$2^{243.3}$
$t_{\text{ots-prg}}/\epsilon_{\text{ots-prg}}$	$2^{67.9}$	$2^{99.4}$	$2^{115.2}$	$2^{178.7}$	$2^{242.3}$

1.4 The Merkle Signature Scheme

Merkle presents in [Mer90] a multi-time signature scheme. In that work, he improves the Lamport-Diffie one-time signature scheme [DH76], whose security hinges on that of the underlying hash function, and introduces his multi-time extension based on a binary tree, which we will call Merkle tree henceforth. No formal proof

of the security of the scheme has been given, although its properties are widely known. See e.g. Micali [Mic00] who mentions without proof, that the crucial property of Merkle trees is the difficulty of changing any node in the tree while keeping the root unaltered, provided that the used hash function is collision resistant.

Actually, the Merkle signature scheme transforms any one-time signature scheme into a multi-time one. The idea of the Merkle signature scheme is the following: Assume that we want to be able to make at most 2^N signatures. *Key generation* works as follows: Generate 2^N different key pairs of the one-time signature scheme. Then create a binary tree whose leaves are the hash values of the verification keys, and each parent is the hash value of the concatenation of its left and right children. The public value is the root of the tree. All the one-time key pairs taken together serve as the private key. The *signature* of k-th message to be signed is the one-time signature made with the k-th private one-time key, followed by the one-time verification key and N nodes from the Merkle tree which help to authenticate the verification key against the public value. The *verification* consists of two parts: First, authentication of the public one-time key with the auxiliary N nodes against the public value and second, authentication of the message with the public one-time key.

Even though a description of the Merkle signature scheme is given in [Mer90], in this Section we give a formal description of the scheme for the sake of completeness.

The following notion will be needed for the description of the Merkle signature scheme. Assume we are given a rooted full binary tree of depth N. Let us label the leaves of the tree from 0 to $2^N - 1$. If L is the leaf of the tree with label i, we call $(P_N, \ldots, P_1)$ the *i-labeled sibling path of L*. Here P_j is the sibling of the node of the path from the root to L at depth j, for $1 \leq j \leq N$. This is illustrated in Figure 1.2 on the next page.

Now, let $\mathsf{1Sign} = (\mathsf{1Gen}, \mathsf{1Sig}, \mathsf{1Ver})$ be a one-time signature scheme, and let $H : \{0,1\}^m \to \{0,1\}^s$ be a hash function. In what follows we use as security parameter of each scheme the size of the output of the hash function.

Scheme 1.14 Original Merkle signature scheme (*oMss*).

Key generation algorithm Gen**.** Given $N \in \mathbb{N}$ as input, Gen calls 2^N times $\mathsf{1Gen}$ in order to obtain the key pairs (X_i, Y_i) for $0 \leq i < 2^N$. Then it

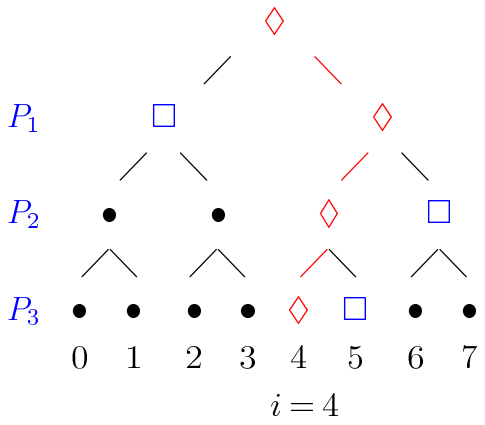


Figure 1.2: Example of an i-labeled sibling path for $N = 3$ which is depicted in □. The path of the leaf i is pictured in ◊.

computes recursively the nodes of the tree: $K_{N,i} = H(Y_i)$ and $K_{j,k_j} = H(K_{j+1,2k_j}, K_{j+1,2k_j+1})$ for $0 \leq i < 2^N$, $0 \leq j < N$ and $0 \leq k_j < 2^j$. Finally, it sets $R = K_{0,0}$.

The public key output by Gen is (N, R). The private key consists of all one-time key pairs (X_i, Y_i) (in case the one-time verifying keys Y_i can be computed from the X_i, it is sufficient to store only the one-time signing keys X_i instead). The user must keep a counter which contains the number of previously created signatures. In the beginning the counter is set to zero.

Signature algorithm Sig. Let i be the counter. Given as input a message M and the secret key, Sig calls 1Sig(M, X_i) in order to obtain the one-time signature τ and computes $(P_N, \ldots, P_1)$, the i-labeled sibling path for the leaf $H(Y_i)$.

Finally, Sig outputs the signature $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$, and then increments the counter i by one.

Verification algorithm Ver. Given as input a message M, a signature σ' and a public key (N, R), Ver accepts the signature σ' unless 1Ver$(M, \tau', Y') = false$ or $W_0 \neq R$, where $W_N = H(Y')$ and $W_{j-1} := H(l_j, r_j)$ for $0 < j \leq N$. Here $\sigma' = (i, \tau', Y', P'_N, \ldots, P'_1)$ and $l_j := W_j$, $r_j := P_j$ in case that $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even or $l_j := P_j$, $r_j := W_j$ otherwise.

1.5 Efficiency and security of the Merkle signature scheme

In this section we will show that assuming the existence of cryptographically secure hash functions and cryptographically secure one-time signature schemes, the Merkle signature scheme is not existentially forgeable under adaptive chosen message attacks. To do this, we prove first that any alteration of any number of nodes of a Merkle tree that keeps the root intact yields an efficient way to find collisions for the underlying hash function. Then we show that any existential forgery of signatures in the Merkle signature scheme leads to either an existential forgery of signatures for the underlying one-time signature scheme or a collision for the underlying hash function.

1.5.1 Security

Goldwasser et al. [GMR88] introduced the concepts of existentially forgeable and adaptive chosen message attack. We adopt concepts from [BMS03, EGM90] and [RS04] for existentially unforgeable under adaptive chosen message attack, one-time signature schemes and collision-resistant functions respectively. The main result of this section is Proposition 1.5 on the facing page. Lemma 1.4 proves an important property of the Merkle trees when an i-labeled sibling path becomes an i-labeled authentication path in a Merkle tree.

Here we give the notion of an authentication path. Let $H : \{0,1\}^m \to \{0,1\}^s$ be a function, where $m = *$ or $m \geq 2s$, let $N \in \mathbb{N}$ and let $L, P_N, \ldots, P_1, R \in \{0,1\}^s$. Set $0 \leq i < 2^N$. Let $W_N := L$ and $W_{j-1} := H(l_j, r_j)$ for $(0 < j \leq N)$ and $l_j := W_j$, $r_j := P_j$ in case that $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even or $l_j := P_j$, $r_j := W_j$ otherwise. If $R = W_0$, we will say that $(P_N, \ldots, P_1)$ is an *i-labeled authentication path of depth N for the leaf L with respect to the root R*. Here $H(x, y)$ denotes $H(x\|y)$ if $m = *$ and $H(x\|y\|0^{m-2s})$ otherwise, where $\|$ is the concatenation of strings.

Lemma 1.4 *Let H be a hash function, whose image is in $\{0,1\}^s$. Let $R \in \{0,1\}^s$ and let $N \in \mathbb{N}$. Set $0 \leq i < 2^N$. Suppose that $A = (P_N, \ldots, P_1)$ is an i-labeled authentication path of depth N for a leaf L and $B = (P'_N, \ldots, P'_1)$ is an i-labeled*

authentication path of depth N for a leaf L', both of them with respect to the root R. Then either $A \neq B$ or $L \neq L'$ implies an efficient computation of a collision of H.

Proof

We define $W_N := L$ and $W_{j-1} := H(l_j, r_j)$ for $0 < j \leq N$, where $l_j = W_j$ and $r_j = P_j$ in case $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even and $l_j = P_j$ and $r_j = W_j$ otherwise. In an analogous way we define W'_j for $0 \leq j \leq N$.

If $A \neq B$, set $k := \min_{1 \leq j \leq N}\{j \mid P_j \neq P'_j\}$. We know that $W_0 = W'_0 = R$, so if there exists $0 \leq j_0 < k$ such that $W_{j_0} \neq W'_{j_0}$ and $W_j = W'_j$ for $0 \leq j < j_0$ then $H(l, r) = W_{j_0-1} = W'_{j_0-1} = H(l', r')$, where $l = W_{j_0}, r = P_{j_0}, l' = W'_{j_0}, r' = P'_{j_0}$ in case $\lfloor \frac{i}{2^{N-j_0}} \rfloor$ is even and $l = P_{j_0}, r = W_{j_0}, l' = P'_{j_0}, r' = W'_{j_0}$ otherwise. This implies $(l, r) \neq (l', r')$ although $H(l, r) = H(l', r')$. This means we have found a collision of H. Thus we have that $W_{k-1} = W'_{k-1}$ and then $H(l, r) = W_{k-1} = W'_{k-1} = H(l', r')$, where $l = W_k, r = P_k, l' = W'_k, r' = P'_k$ in case $\lfloor \frac{i}{2^{N-k}} \rfloor$ is even and $l = P_k, r = W_k, l' = P'_k, r' = W'_k$ otherwise, which implies that $(l, r) \neq (l', r')$ although $H(l, r) = H(l', r')$. As before this means we have found a collision of H.

If $A = B$ and $L \neq L'$, then we can assume that $W_{N-1} = W_{N'-1}$. This can be seen as follows: notice that $W_0 = W'_0$, now if there is $1 \leq j_0 < N$ such that $W_j = W'_j$ for $0 \leq j < j_0$ and $W_{j_0} \neq W'_{j_0}$, then we proceed in a similar way as in the case $A \neq B$ in order to find a collision for H. So $H(l, r) = W_{N-1} = W_{N'-1} = H(l', r')$, where $l := L, r := P_N, l' := L', r' := P'_N$ in case i is even or $l := P_N, r := L, l' := P'_N, r' := L'$ otherwise, although $(l, r) \neq (l', r')$. □

Proposition 1.5 ***Security of the original Merkle signature scheme.*** *Let H be a (t_{cr}, ϵ_{cr}) collision resistant hash function and let $\mathsf{1Sign} = (\mathsf{1Gen}, \mathsf{1Sig}, \mathsf{1Ver})$ be a (t_{1s}, ϵ_{1s}) one-time signature scheme, such that $\epsilon_{cr} \leq \frac{1}{2}$ and $t_{1s}, t_{cr} \geq t + (N + 1)t_h + 2^N t_S + t_G + t_V$ for some t and some $N < \lfloor -\log_2 \epsilon_{1s} \rfloor$. Here t_h denotes the time needed to compute a hash value, t_S the time needed to compute a one-time signature, t_V the time needed to compute a one-time verification and t_G the time needed to generate a key pair for the Merkle signature scheme. Under this assumptions, for $\epsilon = 2\max\{\epsilon_{cr}, 2^N \epsilon_{1s}\}$ the oMss $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is $(t, \epsilon, 2^N)$ existentially unforgeable under adaptive chosen message attack.*

Proof

The idea of the proof is as follows. If the Merkle signature scheme is existentially forgeable under an adaptive chosen message attack, then there exists a forger $\mathcal{F}$. In this case we are able to construct an algorithm $\mathcal{A}$ which obtains either a collision of H or a forgery of the underlying ots. Given as input a one-time public key Pu and an oracle $\mathcal{O}_{1\mathsf{Sig}}$ to sign a message with the private key Pr corresponding to Pu, the algorithm $\mathcal{A}$ generates a random instance of the oMss which consists of a public key PK and almost a private key SK with the given oracle $\mathcal{O}_{1\mathsf{Sig}}$. $\mathcal{A}$ uses the forger $\mathcal{F}$ to obtain with "high" probability a message and its forged signature $(M', (i, \tau'_{ots}, Y', P'_N, \ldots, P'_1)) \leftarrow \mathcal{F}^{\mathsf{Sig}(\cdot,SK)}(PK)$. All the required messages M_j to be signed and their corresponding signatures σ_j are kept. Recall that the forger must give a forged signature of a message which was not required to be signed. $\mathcal{A}$ computes a genuine signature $\sigma_i = (i, \tau_{ots}, Y, P_N, \ldots, P_1))$ for the obtained message M_i. If the number of queries is greater than i, then such message was a query and its signature is already computed. Otherwise it must be kept signing the message M' until the signature number i is reached. $\mathcal{A}$ efficiently finds a collision if $(Y', P'_N, \ldots, P'_1) \neq (Y, P_N, \ldots, P_1)$. Otherwise, with high probability $\mathcal{A}$ outputs (M', τ'_{ots}) as a forgery of the underlying ots.

Now, suppose that the Merkle signature scheme can be existentially broken via a 2^N message attack within time t and probability ϵ. Then we prove that at least one of the following holds:

- A collision can be found for the underlying hash function with probability at least ϵ_{cr} and within time t_{cr}.
- The underlying one-time signature scheme can be existentially broken with probability at least ϵ_{1s} and within time t_{1s}.

The assumption of the insecurity of the oMss implies the existence of a forger $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ such that

$$Pr[\mathsf{Ver}(M, \sigma', PK) = 1 \,|\, (SK, PK) \leftarrow \mathsf{Gen}(N);\ T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot,SK)}(PK);\ (M, \sigma') \leftarrow \mathcal{F}_2(T)] \geq \epsilon$$

within time t.

This forger will be used by the algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ we construct below. $\mathcal{A}$ will receive as input a one-time public key Pu and an oracle $\mathcal{O}_{\mathsf{1Sig}}$ which signs a message with the corresponding private key Pr. $\mathcal{A}$ will output either a forgery for the underlying one-time signature or a collision of the underlying hash function.

Given as input a one-time public key Pu and its corresponding signing oracle $\mathcal{O}_{\mathsf{1Sig}}$, $\mathcal{A}$ works as follows. **Generating a random instance of the original Merkle signature scheme:** $\mathcal{A}_1$ chooses $0 \leq i_0 < 2^N$ at random and sets $Y_{i_0} \leftarrow Pu$. Now for $0 \leq i < 2^N$ with $i \neq i_0$, $\mathcal{A}_1$ obtains $(X_i, Y_i) \leftarrow \mathsf{1Gen}$. From all the $H(Y_i)$'s as leaves of the Merkle tree, $\mathcal{A}_1$ obtains the Merkle tree's root R. In this case, an instance of the Merkle signature scheme is computed. The public key is $PK = (N, R)$. As secret key SK, $\mathcal{A}$ uses X_i $\forall i \neq i_0$, Y_i $\forall i \neq i_0$ in case that Y_i cannot be obtained from X_i, Y_{i_0} and additionally the oracle $\mathcal{O}_{\mathsf{1Sig}}$ which signs a message with the one-time private key corresponding to Y_{i_0}. **Using $\mathcal{F}$ to obtain a forgery with high probability:** $\mathcal{A}_1$ calls $\mathcal{F}_1$ and obtains $T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, SK)}(PK)$. $\mathcal{A}_1$ uses its SK to sign each query of $\mathcal{F}_1$ and keeps the queries with their corresponding signatures. $\mathcal{A}_1$ outputs $(T, k, (M_0, \sigma_0), \ldots, (M_{k-1}, \sigma_{k-1}), SK)$, where k is the number of queries made by $\mathcal{F}_1$, M_j are the queries themselves and σ_j their corresponding signatures for $0 \leq j < k$. Given as input $(T, k, (M_0, \sigma_0), \ldots, (M_{k-1}, \sigma_{k-1}), SK)$, $\mathcal{A}_2$ calls $\mathcal{F}_2$ with T as input and obtains a pair (M, σ') with probability $\geq \epsilon$ that is a forged signature σ' of a message M. If $\sigma' = (i, \tau', Y', P'_N, \ldots, P'_1)$ is a forged signature of the message M, $\mathcal{A}_2$ obtains the genuine signature of the message M_i as follows: if $i < k$ the pair (M_i, σ_i) is already computed, otherwise $\mathcal{A}_2$ keeps calling $\mathsf{Sig}(M', SK)$ until the signature number i is reached. Here $\sigma_i = (i, \tau, Y_i, P_N, \ldots, P_1)$. **Finding efficiently a collision:** $\mathcal{A}_2$ sets $A = (P'_N, \ldots, P'_1)$ and $B = (P_N, \ldots, P_1)$. Under the assumption that σ' is a forged signature and σ_i is a genuine signature, we have that A is an i-labeled authentication path for the leaf $H(Y')$ and B is an i-labeled authentication path for the leaf $H(Y_i)$, both of them with respect to the root R. If $Y' \neq Y_i$ but $H(Y') = H(Y_i)$, then a collision is already found. If $(Y', A) \neq (Y_i, B)$ then $\mathcal{A}$ outputs the collision which is found by Lemma 1.4 on page 30. **Finding a forgery for the underlying one-time signature scheme:** Otherwise, $\mathcal{A}_2$ outputs (M, σ') as a forgery for the underlying ots.

Let ρ be the probability that (Y', A) obtained by $\mathcal{F}$ and (Y, B) obtained by the signature algorithm are different. We know that $\mathcal{F}$ produces a forgery with probability $\geq \epsilon$. Thus, if $\rho \geq \frac{1}{2}$ then $\mathcal{A}$ finds a collision of H within time t_{cr} with probability at least $\frac{\epsilon}{2} \geq \epsilon_{cr}$. Else with probability $\frac{1}{2^N}$, the forged signature output by $\mathcal{F}$ satisfies that $i = i_0$. Hence, the attack by $\mathcal{A}$ on the one-time signature scheme succeeds with probability at least $\frac{1}{2}\frac{\epsilon}{2^N} \geq \epsilon_{1s}$ within time t_{1s}. □

1.5.2 Efficiency

In Table 1.7 we show the size of the secret and public key as well as the size of the signature. The notation is as follows: l_{sec} is the maximal size of a one-time signing key, l_{ver} is the maximal size of the corresponding one-time verification key, l_{sig} is the maximal size of the corresponding one-time signature, l_{int} is the number of bits needed to store the counter, and s is the bit size of the output of the underlying hash function. In the third column we also show an example. We set $l_{int} = 32$ bits and consider the iLDots with pseudorandom key.

Table 1.7: Efficiency of the oMss (size)

	Size	
	in bits	$s = 160,\ N = 18$
Public key	$s + l_{int}$	24 bytes
Secret key	$2^N l_{sec}$	$\approx$ 56.8 Mb
Signature	$l_{int} + l_{sig} + l_{ver} + Ns$	$\approx$ 2.15 Kb

The Merkle tree is not suitable for being stored and for this reason an efficient way for computing authentication paths is required. Szydlo presented in [Szy03] an efficient algorithm for such task. Only up to $3N$ nodes must be stored and up to N computations must be made. We take as worst case the computation of leaves instead of only inner nodes in the Szydlo algorithm. Table 1.8 shows estimates of the time needed for an instance of the oMss, where $t_{1G}, t_{1S}, t_{1V}, t_h, t_L$ and t_{rg} are the times needed to generate a one-time key pair, to compute a one-time signature,

to verify a one-time signature, to compute a hash value, to compute a Merkle tree's leaf and to generate a random bit string of size s, respectively.

Table 1.8: Efficiency of the oMss (time)

	Time
Key generation	$2^N(t_{1G} + 2t_h + t_{rg}) - t_h$
Verification	$t_{1V} + Nt_h$
Signature	$t_{1S} + Nt_L$

1.6 Two improved versions of the Merkle signature scheme

We develop two version of the Merkle signature scheme. One of them is forward secure, while the other one has in addition a practically unlimited number of possible signatures.

A forward secure version of the Merkle Signature Scheme. We modify the original Merkle signature scheme in order to transform it into a forward secure one. Note that if all the one-time signing keys are stored and each one of them is deleted after its use, the resulting Merkle signature scheme is forward secure, where each period consists of exactly one signature. In this case, the private key of the Merkle signature scheme is as big as the stored one-time signing keys. In our version, the size of the private key is reduced by employing a pseudorandom bit generator and a one-time signature scheme with deterministic key-generation. Bellare and Yee have shown in [BY03] how to construct a forward-secure pseudorandom bit generator from a cryptographically secure pseudorandom bit generator. The use of that bit generator enables us to prove that our new version of the Merkle signature scheme becomes forward secure.

Key generation process split through the signature process. In the original Merkle signature scheme, the number of possible signatures is 2^N, where N is the depth of the Merkle tree. The bigger the parameter N is, the slower the key generation process becomes: during key generation $2^{N+1} - 1$ hash values and 2^N one-time key pairs have to be computed.

We present a version where part of the key generation takes place during the use of the signature algorithm. This permits the use of sufficiently large parameters to allow for a practically unlimited number of signatures while keeping the cost of the initial key generation process low. The basic idea is to use one "main" Merkle tree to authenticate the roots of a series of other "secondary" trees. Only one of the secondary trees is kept at a time. During the use of one secondary tree, the next one is generated, namely two nodes at a time per signature.

1.6.1 First improved version of the Merkle Signature Scheme

We can save space for the private key in the Merkle signature scheme if we can compute a specific one-time-signature key pair any time we need it, that is why we need a deterministic key generation. This goal can be reached considering a cryptographically secure pseudorandom bit generator. Recall that a pseudorandom bit generator (prg) $g : \{0,1\}^l \to \{0,1\}^{l+l'}$ is *cryptographically secure* if $g(\mathcal{U}_l)$ and $\mathcal{U}_{l+l'}$ are computationally indistinguishable, where $\mathcal{U}_\alpha$ denotes the uniform distribution on $\{0,1\}^\alpha$. The basic idea of our first proposal is (i) to create "generators" for the needed seeds of the ots with deterministic key generation through the prg, (ii) to keep some of the "generators" as auxiliary data in order to compute from them the necessary seeds during the signing process and, finally, (iii) to erase each seed and its corresponding one-time-signature private key after it has been used to sign a message.

The security parameter s relies on the size of the output of the underlying hash function. We assume that $\mathsf{1Sign} = (\mathsf{1Gen}, \mathsf{1Sig}, \mathsf{1Ver})$ is a one-time signature scheme with deterministic key generation algorithm (e. g. the iLDots with pseudorandom key) and that there exists $PRG : \{0,1\}^s \to \{0,1\}^u$ which is a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator and $u \geq 2s$. In this version we take into account

the algorithm presented by Szydlo in [Szy03]; which sequentially, but efficiently, computes authentication paths in a Merkle tree. Because of Szydlo algorithm $3N$ nodes of the Merkle tree must be considered as auxiliary data for signing, where N is the depth of the Merkle tree.

Scheme 1.15 First improved version of the Merkle signature scheme *iMss*.

Key generation algorithm Gen. Given as input $N \in \mathbb{N}$, Gen chooses $g_{-1} \in_R \{0,1\}^s$ and obtains the key pair $(X_i, Y_i) \leftarrow \mathsf{1Gen}(seed_i)$ for each $0 \leq i < 2^N$, where $(g_i, seed_i) \leftarrow PRG(g_{i-1})$. As in Scheme 1.14 on page 28, Gen computes the corresponding Merkle tree, i.e. K_{j,i_j} for $0 \leq j \leq N$ and $0 \leq i_j < 2^j$. Gen outputs the tree's depth N and the tree's root R as the public key. As the private key would suffice g_{-1}. However, to reduce the computation while signing the g_j and up to $3N$ nodes needed by the Szydlo algorithm must be stored, where $j = q2^{\frac{N}{2}} - 1$ for $0 \leq q < 2^{\frac{N}{2}}$. The signer must keep a counter in order to sign sequentially. At this point such counter must be initialized to zero.

Signature algorithm Sig. Let i be the counter. Given as input the message M and the private key, which contains g_{i-1} and $g_{q2^{\frac{N}{2}}-1}$ for $\left\lfloor \frac{i}{2^{\frac{N}{2}}} \right\rfloor < q < 2^{\frac{N}{2}}$, Sig obtains $(g_i, seed_i) \leftarrow PRG(g_{i-1})$. Then, it calls $(X_i, Y_i) \leftarrow \mathsf{1Gen}(s, seed_i)$. It computes the one-time signature of the message $\tau \leftarrow \mathsf{1Sig}(M, X_i)$ and applies the Szydlo algorithm to compute $(P_N, \ldots, P_1)$, the i-labeled sibling path for the leaf $H(Y_i)$. Sig outputs the signature $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$ and finally, Sig stores g_i, deletes g_{i-1} and $seed_i$, and after that increments the counter i by one.

Verification algorithm Ver. This algorithm remains the same as in Scheme 1.14 on page 28.

Secret key update algorithm Upd. This algorithm is inherent to the signature one. The private key is changed after each signature. A period consists of only one signature instead of time.

1.6.2 Security of iMss

Bellare and Yee define in [BY03] the concept of forward-secure pseudorandom bit generator and give its construction from any pseudorandom bit generator. This tool allows us to prove easily that iMss remains not only secure, but also forward secure.

First we will prove in Proposition 1.6 that iMss is not existentially forgeable under adaptive chosen message attacks. After that, we will prove in Proposition 1.8 on page 41 that iMss is also forward secure.

Proposition 1.6 *Let H be a (t_{cr}, ϵ_{cr}) collision resistant hash function, let* 1Sign *be a (t_{1s}, ϵ_{1s}) one-time signature scheme with deterministic key generation algorithm and let PRG be a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator, such that $2^{N+2}\epsilon_{prg} \leq 1$, $\epsilon_{cr} \leq \frac{1}{4}$, $t_{cr} \geq t+(N+1)t_h+2^N t_s+t_g$, $t_{1s} \geq t+2^N t_s+t_g$ and $t_{prg} = t+t_g+2^N t_s+t_v+O(2^{N+1}s)$ for some t and some $N < \lfloor -\log_2 \frac{\epsilon_{1s}}{2} \rfloor$, where t_h is the time needed to compute a hash value, t_s is the time needed to compute a one-time signature, t_v is the time needed to verify a Merkle signature and t_g is the time needed to generate a key pair for the Merkle signature scheme. If $\epsilon = \max\{4\epsilon_{cr}, 2^{N+2}\epsilon_{1s}, 2^{N+2}\epsilon_{prg}\}$, then the iMss* Sign *is $(t, \epsilon, 2^N)$ existentially unforgeable under adaptive chosen message attacks.*

Proof

The idea of the proof is as follows. We know that the oMss is secure as proven in Proposition 1.5 and that prg is cryptographically secure by hypothesis. If the iMss is not secure, then there exists a forger $\mathcal{F}$ for iMss. Using $\mathcal{F}$, we can construct a distinguisher $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ for the prg, what would contradict the security of the prg. Basically, $\mathcal{A}$ collects all the 2^N permitted values. From such values, $\mathcal{A}$ computes a Merkle key pair, which is a random instance of either oMss or iMss, depending on how the values were generated. $\mathcal{A}$ calls $\mathcal{F}$ to try to obtain a forgery. $\mathcal{A}$ returns 1, if $\mathcal{F}$ returns a forged signature within the permitted time. That is, the success of $\mathcal{F}$ implies with "high" probability that the values come from the pseudorandom bit generator, since the iMss is not secure. Otherwise, $\mathcal{A}$ returns 0. That is, the failure of $\mathcal{F}$ implies with "high" probability that the values were randomly chosen, since the oMss is secure.

Now, under the assumption that H is a (t_{cr}, ϵ_{cr}) collision resistant hash function and that $\mathsf{1Sign}$ is a (t_{1s}, ϵ_{1s}) one-time signature scheme, if the iMss can be existentially broken via a 2^N message attack within time t with probability at least ϵ, we can construct a distinguisher $\mathcal{A}$ for the $(t_{fsprg}, \epsilon_{fsprg})$ forward-secure pseudorandom bit generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, s, 2^N)$ obtained from PRG. Here $t_{fsprg} = t + t_g + 2^N t_s + t_v$, $\epsilon_{fsprg} = 2^{N+1}\epsilon_{prg}$, and $\mathcal{G}$ is as in Proposition 1.1 on page 14.

Thus, suppose the existence of a forger $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ such that

$$Pr[\mathsf{Ver}(M, \sigma', PK) = 1 \mid (SK, PK) \leftarrow \mathsf{Gen}(N);\ T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, SK)}(PK);\ (M, \sigma') \leftarrow \mathcal{F}_2(T)] \geq \epsilon$$

within time t.

Using the notation of definition 1.11 on page 14 and table 1.2 on page 15, the distinguisher $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ is constructed as follows. **Collection of all permitted values:** The parameter h is initialized as the null string λ. Having as input Out_i and h, $\mathcal{A}_1$ updates $h \leftarrow h||Out_i$ and outputs $(\texttt{not\_guess}, h)$. **Generation of a random instance of either oMss or iMss:** Given as input St_{2^N-1} and h, $\mathcal{A}_2$ obtains Out_i from h for $0 \leq i < 2^N$ and computes $(X_i, Y_i) \leftarrow \mathsf{1Gen}(Out_i)\ \forall i$ and the Merkle tree's nodes K_{j,k_j} from Y_i $(0 \leq j \leq N,\ 0 \leq k_j < 2^j)$. As secret key SK, $\mathcal{A}_2$ retains all X_i, Y_i and the nodes K_{j,k_j} needed by the Szydlo algorithm for efficiently creating authentication paths. As public key PK, $\mathcal{A}_2$ outputs (N, R), where R is the Merkle tree's root. $\mathcal{A}_2$ deletes X_i after its employment in $\mathsf{1Sig}$. **Use of $\mathcal{F}$ to try to obtain a forgery:** $\mathcal{A}_2$ calls $\mathcal{F}_1(PK)$ and uses the secret key SK to sign queries of $\mathcal{F}_1$ as an oracle for $\mathcal{F}_1$. $\mathcal{A}_2$ uses the output T from $\mathcal{F}_1$ as input for $\mathcal{F}_2$ in order to obtain (M, σ'). **Distinction between random and pseudorandom values:** If (M, σ') is a forged signature, then $\mathcal{A}_2$ outputs 1. Otherwise $\mathcal{A}_2$ outputs 0.

Note that in $\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the iMss and in $\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the oMss.

Because of the assumption of the insecurity of the iMss, we have

$$Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] \geq \epsilon$$

within time t_{fsprg}. On the other hand

$$Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] < \tfrac{\epsilon}{2}$$

within time t_{fsprg} because of the assumption of H and 1Sign, which implies that the oMss is $(t, \frac{\epsilon}{2}, 2^N)$ existentially unforgeable under adaptive chosen message attacks as proven in Proposition 1.5 on page 31.

Therefore,

$$Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] - Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] > \tfrac{\epsilon}{2} \geq 2^{N+1}\epsilon_{prg} \geq \epsilon_{fsprg}$$

within time t_{fsprg}, which contradicts Proposition 1.1 on page 14.

□

1.6.3 Forward security of the iMss

As described in Scheme 1.15, the secret key update process is intrinsic to the signature one and cannot be called at any time. Each period consists of a certain number of signatures instead of time. We will prove that the iMss is also forward secure.

With the notation in Scheme 1.14 the oMss can be modified as follows.

Signature algorithm Sig. Having as input a message M, Sig uses (X_i, Y_i) in order to obtain $\tau \leftarrow \mathsf{1Sig}(M, X_i)$. It computes the i-labeled sibling path for the leaf $H(Y_i)$, outputs $(i, \tau, Y_i, P_N, \ldots, P_1)$, deletes X_i and then increments i by one.

The Key generation algorithm Gen **and the verification algorithm** Ver of the original Merkle signature scheme remain the same.

The private key at the period i is either

$$SK_i = \{X_j\}_{i \leq j < 2^N} \text{ or } SK_i = \{(X_j, Y_j)\}_{i \leq j < 2^N}.$$

Secret key update algorithm Upd is inherent to the signature algorithm.

This modified version is forward secure.

Lemma 1.7 *With the hypothesis of Proposition 1.5 on page 31, this modified oMss is $(t, \epsilon, 2^N)$ forward secure, where the number of periods is 2^N and each period consists of only one signature.*

The proof of Lemma 1.7 on the preceding page is similar to the one of Proposition 1.5 on page 31. Note that each instance of the underlying ots is chosen at random and is independent of each one of the others. Thus, if certain one-time private key is deleted after its use, then the knowledge of the other ones does not give any information at all of the deleted one. Therefore, when an adversary modeled by an algorithm $\mathcal{A}$ attempts to forge a signature, the knowledge of the Merkle private key after k signatures does not give to $\mathcal{A}$ any information at all of the one-time secret keys X_i $(0 \leq i < k)$. Recall that each period consists of only one signature instead of time. A forged signature of a previous period of k would result in an efficient computation of either a collision or a pre-image of the underlying hash function as in the proof of Proposition 1.5.

Proposition 1.8 *With the hypothesis of Proposition 1.6 on page 38. The iMss is* $(t, 3\epsilon, 2^N)$ *forward secure, with each period consisting of only one signature.*

With the hypothesis of this proposition and by Lemma 1.7 on the facing page, we have that the modified oMss is $(t, 2\epsilon, 2^N)$ forward secure. We know that the prg is cryptographically secure by hypothesis. If the iMss is not forward secure, we are be able to construct a distinguisher $\mathcal{A}$ for the forward-secure pseudorandom bit generator obtained from PRG, contradicting the assumption of the cryptographically security of the prg and Proposition 1.1 on page 14. The construction of $\mathcal{A}$ is analogous to the one made in the proof of Proposition 1.6.

As in the proof of Proposition 1.6, basically, $\mathcal{A}$ collects all the 2^N permitted values. From such values, $\mathcal{A}$ computes a Merkle key pair, which is a random instance of either the modified oMss or the iMss, depending on how the values were generated. $\mathcal{A}$ calls $\mathcal{F}$ to try to obtain a forgery. $\mathcal{A}$ returns 1, if $\mathcal{F}$ returns a forged signature within the permitted time. This fact means that

$$Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] \geq 3\epsilon$$

within time t_{fsprg}, since the instance generated by $\mathcal{A}$ is an iMss one, which is not $(t, 3\epsilon, 2^N)$ forward secure. Otherwise, $\mathcal{A}$ returns 0. This fact means that

$$Pr[\mathrm{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] < 2\epsilon$$

within time t_{fsprg}, since the instance generated by $\mathcal{A}$ is a modified-oMss one, which is $(t, 2\epsilon, 2^N)$ forward secure. Therefore we have that

$$Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] - Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] > \epsilon \geq 2^{N+2}\epsilon_{prg} \geq \epsilon_{fsprg}$$

within time t_{fsprg}. This last inequality contradicts Proposition 1.1.

1.6.4 Efficiency of iMss

Recall that by Szydlo algorithm [Szy03] authentication paths can be efficiently computed in time N and space less than $3N$, where the unit of space is the size of the nodes and the unit of computation is a hash function evaluation or leaf value generation. These two different types of computation are treated the same way by Szydlo. In our estimates we will consider as a unit of computation the leaf value generation, whose computation time is bigger than the one of a hash function evaluation.

In Table 1.9 we show the size of the secret and public key and the one of the signature, too. The notation is as follows: l_{sec} is the maximal size of a one-time signing key, l_{ver} is the maximal size of the corresponding one-time verification key, l_{sig} is the maximal size of the corresponding one-time signature, l_{int} is the number of bits needed to store the counter and s is the bit size of the output of the underlying hash function. In this version, the secret key has been reduced.

Table 1.9: Efficiency of the iMss (size)

	Size in bits
Public key	$s + l_{int}$
Secret key	$s\left(2^{\lceil \frac{N}{2} \rceil} + 3N\right) + l_{int}$
Signature	$l_{int} + l_{sig} + l_{ver} + Ns$

In Table 1.10 we show estimates of the time needed for an instance of the iMss, where $t_{1G}, t_{1S}, t_{1V}, t_h, t_L$ and t_{prg} are the times needed to generate a one-time key pair, to compute a one-time signature, to verify a one-time signature, to compute

a hash value, to compute a Merkle tree's leaf and to generate a pseudorandom bit string of size s, respectively.

Table 1.10: Efficiency of the iMss (time)

	Time
Key generation	$2^N(t_{1G} + 2t_h + t_{prg}) - t_h$
Verification	$t_{1V} + Nt_h$
Signature	$t_{1S} + N(t_L + 2^{\lceil \frac{N}{2} \rceil} t_{prg})$

In our estimate we consider the following worst case: N leaves must be computed by the Szydlo algorithm and the seed needed to compute each leaf must be generated through $2^{\lfloor \frac{N}{2} \rfloor}$ calls to the prg.

1.6.5 Second improved version of the Merkle signature scheme

We can reduce the time needed for the key generation and for the signature process. The idea is to split the generation of the private key during the lifetime of the instance. Recall that the Merkle signature scheme transforms an ots into a multi-time one. Then, we can use an instance of the iMss for "certifying" public keys of auxiliary instances. In this way, the size of the signature is increased, but the size of the private key and the time needed to sign are reduced.

Let $\mathsf{MSign} = (\mathsf{MGen}, \mathsf{MSig}, \mathsf{MVer})$ be the iMss as described in Scheme 1.15 on page 37. Next we describe our second version of the Merkle signature scheme. The security parameter relies on the bit size of the underlying hash function. This time, each period consists of up to 2^N signatures instead of time.

Scheme 1.16 Second improved version of the Merkle signature scheme (*iMss with split key*).

Key generation algorithm Gen**.** Having as input the parameter $2N$, for 2^{2N} possible signatures, Gen calls twice $\mathsf{MGen}(N)$ to compute two pairs of iMss

private and public keys. One pair of them is the main key (SK_M, PK_M) and the other one is a secondary key (SK_{S_0}, PK_{S_0}). MGen computes $\zeta_0 = \mathsf{MSig}(PK_{S_0}, SK_M)$ and outputs PK_M as the public key and (SK_M, SK_{S_0}) as the private key.

The signer must keep two counters. One of them counts the number of generated signatures modulo 2^N, which at this point must be initialized to zero, and the other counts the number of signatures created by the main private key minus one, which at this point must be initialized to zero. The signer must also keep (ζ_0, PK_{S_0}), which are the signature of the secondary public key and the secondary public key itself.

Recall that the iMss public key is of the form $PK = (N, R)$, where R is the Merkle tree's root. Thus, it suffices to sign R instead of PK.

Signature algorithm Sig**.** Let i and j be the counters described in the previous Key generation algorithm. Given as input a message M and the secret key (SK_M, SK_{S_j}), Sig computes $\tau_i \leftarrow \mathsf{MSig}(M, SK_{S_j})$ and sets $\sigma = (\tau_i, PK_{S_j}, \zeta_j)$. Sig increments i by one.

Recall that the iMss private key evolves after each signature.

If at this point $i \equiv 0 \mod 2^N$, Sig calls $\mathsf{MGen}(N)$ to obtain another secondary key $(SK_{S_{j+1}}, PK_{S_{j+1}})$ then computes $\zeta_{j+1} \leftarrow \mathsf{MSig}(PK_{S_{j+1}}, SK_M)$. Sig sets $i \leftarrow 0$ and increments j by one. Finally, Sig outputs the signature σ. The signer must know (ζ_j, PK_{S_j}) at any time during the $j+1$-th period.

Verification algorithm Ver**.** Having as input a message M and a signature $\sigma = (\tau, PK', \zeta)$, Ver accepts the signature unless $\mathsf{MVer}(M, \tau, PK') = false$ or $\mathsf{MVer}(PK', \zeta, PK_M) = false$.

Secret key update algorithm Upd**.** Given as input the secret key (SK_M, SK_{S_j}) at the $j+1$-th period, Upd calls $\mathsf{MGen}(N)$ to obtain another secondary key $(SK_{S_{j+1}}, PK_{S_{j+1}})$, then computes $\zeta_{j+1} \leftarrow \mathsf{MSig}(PK_{S_{j+1}}, SK_M)$. Upd sets $i \leftarrow 0$ and increments j by one. Recall that the iMss private key evolves after each signature. The signer must keep the new (ζ_j, PK_{S_j}) and deletes the old one.

1.6.6 Security of the iMss with split key

This iMss with split key uses random and independent instances of the iMss. There is a "main" instance and a "secondary" instance at a time. Intuitively, the iMss with split key is not existentially forgeable under adaptive chosen message attacks, since each instance of the underlying iMss it is also. In other words, a forgery of the iMss with split key implies a forgery of an instance of the iMss.

Proposition 1.9 *Let* MSign *be the iMss as described in Scheme 1.15 on page 37. Let* MSign *be* $(t_{ms}, \epsilon_{ms}, 2^N)$ *forward secure, such that* $t_{ms} \geq t + 2^N(2^N t_{sig} + t_{gen})$ *and* $2^{N+1}\epsilon_{ms} \leq 1$ *for some* t*, where* t_{sig} *is the time needed for* MSig *to sign a message and* t_{gen} *is the time needed for* MGen *to generate a key pair. Then the iMss with split key* Sign*, as described in Scheme 1.16 on page 43, is* $(t, \epsilon, 2^{2N})$ *forward secure, where* $\epsilon = 2^{N+1}\epsilon_{ms}$*.*

Proof

The idea of the proof is as follows. If the iMss with split key can be broken via a 2^{2N} message attack, then there exists a forger $\mathcal{F}$ which succeeds with "high" probability in forging signatures. In that case we are able to construct an algorithm $\mathcal{A}$ which breaks one of two given random instances of the iMss within time t_{ms} and probability at least ϵ_{ms}. Basically, having as input two public keys of random instances of the iMss and their oracles to sign messages and to obtain the corresponding private key, $\mathcal{A}$ constructs a random instance of the iMss with split key from that input. $\mathcal{A}$ uses $\mathcal{F}$ to obtain a forgery of the iMss with split key with "high" probability. Finally, $\mathcal{A}$ gives a forgery of one of the two given instances of the iMss with "high" probability. That fact would contradict Proposition 1.8 on page 41.

Now, if iMss with split key is not $(t, \epsilon, 2^{2N})$ forward secure, then there exists a forger $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ such that

$$Pr[\mathsf{Ver}(M, \langle l, \sigma' \rangle, PK) = 1 \mid (SK_0, PK) \leftarrow \mathsf{Gen}(N);\ T \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, SK_j), \mathsf{Upd}}(PK);\ (M, \langle l, \sigma' \rangle) \leftarrow \mathcal{F}_2(T, SK_k) \text{ and } l < k] \geq \epsilon$$

within time t.

Let (N, R_A) and (N, R_B) be the public keys of two random instances of the iMss. Let $\mathcal{O}_{\mathsf{X-Sig}}$ be the oracle for signing messages with the private key corresponding to the public key (N, R_X) and let $\mathcal{O}_{X-SK}$ be the oracle for obtaining the private key corresponding to the public key (N, R_X), where $X = A, B$.

We construct an algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ as follows.

Generation of a random instance of the iMss with split key: Having as input (N, R_A) and (N, R_B), $\mathcal{A}_1$ chooses $0 \leq j_0 < 2^N$ at random and sets $PK_M \leftarrow (N, R_A)$, the "main" public key, and $PK_{S_{j_0}} \leftarrow (N, R_B)$, the $(j_0 + 1)$-th "secondary" public key. $\mathcal{A}_1$ proceeds to generate an instance of the iMss with split key. Let j be the counter of the signatures created by the "main" secret key, i. e., by the oracle $\mathcal{O}_{\mathsf{A-Sig}}$, and let i be the counter of the signatures created by the secondary private key. If $j \neq j_0$, the signatures are made by the instance of the iMss created by $\mathcal{A}_1$. Otherwise, the signatures are created by $\mathcal{O}_{\mathsf{B-Sig}}$.

Use of $\mathcal{F}$ to try to obtain a forgery: $\mathcal{A}_1$ calls $\mathcal{F}_1(PK_M)$ and signs queries from $\mathcal{F}_1$ using $\mathcal{O}_{\mathsf{A-Sig}}$ and $\mathcal{O}_{\mathsf{B-Sig}}$, if necessary. If $\mathcal{F}_1$ calls the secret key update algorithm, $\mathcal{A}_1$ obtains the secondary key (SK_{j+1}, PK_{j+1}), computes $\zeta_{j+1} \leftarrow \mathcal{O}_{\mathsf{A-Sig}}(PK_{j+1})$ and then increases j by one. If T is the output of $\mathcal{F}_1$, $\mathcal{A}_1$ outputs T. Giving as input T, $\mathcal{A}_2$ obtains $SK_M \leftarrow \mathcal{O}_{A-SK}$. If $j = j_0$, $\mathcal{A}_2$ calls $SK_{S_j} \leftarrow \mathcal{O}_{B-SK}$. $\mathcal{A}_2$ calls $\mathcal{F}_2(T, \{SK_M, SK_{S_j}\})$. Since $\mathcal{F}$ is a forger of the iMss with split key, $\mathcal{F}_2$ outputs a message M and a forged signature $(\tau_{i'}, Pu', \zeta_{j'})$ with probability at least ϵ of a previous period $j' < j$. In that case, we have that $\mathsf{Ver}(M, (\tau_{i'}, Pu', \zeta_{j'}), PK_M) = true$ implies $\mathsf{MVer}(M, \tau_{i'}, Pu') = true \wedge \mathsf{MVer}(Pu', \zeta_{j'}, R) = true$.

Obtainment of a forgery of iMss: If $Pu' \neq PK_{S_{j'}}$, $\mathcal{A}_2$ outputs $(Pu', \zeta_{j'})$ as a forgery of iMss with respect to the public key (N, R_A). Otherwise, we have that $j' = j_0$ with probability at least $\frac{1}{2^N}$ and then $\mathcal{A}_2$ outputs $(M, \tau_{i'})$ as a forgery of iMss with respect to the public key (N, R_B).

Let $p = Pr[Pu' \neq PK_{S_{j'}} \mid (M, (\tau_{i'}, Pu', \zeta_{j'})) \leftarrow \mathcal{F}]$. We know that $\mathcal{F}$ succeeds in forging a signature with probability at least ϵ. Thus, if $p \geq \frac{1}{2}$, then $\mathcal{A}$ outputs a forgery of iMss with respect to the public key (N, R_A) with probability at least $\frac{\epsilon}{2} \geq \epsilon_{ms}$. Otherwise, $\mathcal{A}$ outputs a forgery of iMss with respect to the public key (N, R_B) with probability at least $\frac{\epsilon}{2 \cdot 2^N} \geq \epsilon_{ms}$. Being both cases within time t_{ms},

this contradicts the security of the iMss.

□

1.6.7 Efficiency of the iMss with split key

In Scheme 1.16 on page 43 the signature algorithm is not optimized yet. The computation of the Merkle tree of the third instance can be constructed during the creation of each signature by computing two nodes at a time. The computation of the needed leaves by the Szydlo algorithm for the main Merkle tree can be made through the signature process. In this way as a worst case we compute for the main tree only an extra prg value and a leaf instead of $N2^{\frac{N}{2}}$ prg values besides the corresponding leaves. After 2^N signatures $2^{N+1}-1$ nodes of the third tree would have been created, that is the whole tree. The needed leaves needed by the main tree would be also computed. In such a case the N nodes needed by the Szydlo algorithm and the seeds needed for the third instance must be stored, besides the nodes and seeds needed for the main and secondary instances and the (up to) N nodes needed during the Merkle tree computation of that third instance. We take the previous note in our estimates.

In case the secret key update algorithm is called, the construction of the the third Merkle tree must be finished.

In Table 1.11 we show the size of the secret and public key and the one of the signature, too. The notation is as follows: l_{sec} is the maximal size of a one-time signing key, l_{ver} is the maximal size of the corresponding one-time verification key, l_{sig} is the maximal size of the corresponding one-time signature, l_{int} is the number of bits needed to store the counter and s is the bit size of the output of the underlying hash function. Even though a signature and a public key of a one-time signature scheme is not a secret value, we include this two elements in the private key of iMss with split key as auxiliary values. That is because they are needed in the signature algorithm and it is a waste of time to recompute them each time a signature is created. The number of possible signatures is 2^N and each period consists of up to $2^{\frac{N}{2}}$ signatures instead of time. We consider even values of N.

Table 1.11: Efficiency of the iMss with split key (size)

	Size in bits
Public key	$s + l_{int}$
Secret key	$s\left(2^{\lceil \frac{N}{4} \rceil + 1} + 4N\right) + l_{sig} + l_{ver} + 2l_{int}$
Signature	$2(l_{int} + l_{sig}) + l_{ver} + Ns$

In Table 1.12 we show estimates of the time needed for an instance of the iMss with split key, where $t_{1G}, t_{1S}, t_{1V}, t_h, t_L$ and t_{prg} are the times needed to generate a one-time key pair, to compute a one-time signature, to verify a one-time signature, to compute a hash value, to compute a Merkle tree's leaf and to generate a pseudorandom bit string of size s, respectively.

Table 1.12: Efficiency of the iMss with split key (time)

	Time
Key generation	$2^{\frac{N}{2}+1}(t_{1G} + 2t_h + t_{prg}) - 2t_h + t_{1S}$
Verification	$2t_{1V} + Nt_h$
Signature	$2t_{1S} + \frac{N}{2}(t_L + 2^{\lceil \frac{N}{4} \rceil} t_{prg}) + 3t_L + 2t_{prg}$
Secret update	$t_{1S} + 2^{\frac{N}{2}}(t_{1G} + 2t_h + t_{prg}) - t_h + \frac{N}{2}(t_L + 2^{\lceil \frac{N}{4} \rceil} t_{prg})$

In our estimate we consider the following worst case: $\frac{N}{2}$ leaves must be computed for the secondary tree by the Szydlo algorithm and the seed needed to compute each leaf must be generated through $2^{\lceil \frac{N}{4} \rceil}$ calls to the prg.

1.7 Practical Results

Now a days there exist hash functions whose bit length of their output is 160, 224, 256, 384 and 512. We make some estimates of security of the iMss and iMss with split key, we take into account the previous bit lengths. We also make estimates of the size of the private key, public key and signature.

First we begin with estimates of the length of keys and signature. The underlying one-time signature scheme is the iLDots. In Table 1.13 we show the length of the private key, the public key and the signature. The number of possible signatures for iMss and iMss with split key is 2^N. The bit length of the output of the underlying hash function is s. Auxiliary values for signing are included in the estimates of the length of the private key.

Table 1.13: Length of keys and signatures.

	iLDots	iMss	iMss with split key
Public key	s		
Private key	s	$(2^{\frac{N}{2}} + 3N)s + \ell_{\text{int}}$	$(2^{\frac{N}{4}} + 4N + s'' + 1)s + 2\ell_{\text{int}}$
Signature	ss''	$\ell_{\text{int}} + s(s'' + 1 + N)$	$2\ell_{\text{int}} + s(2s'' + 2 + N)$

Here $s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil$

Now we make estimates of the time needed to generate keys, to sign a message and to verify a signature. The underlying one-time signature scheme is the iLDots. In Table 1.14 on the following page we show estimates for the needed time to generate keys, to sign messages and to verify signatures. The time needed for computing a hash value, a pseudorandom value, a random value, and a hash value whose input has $s''s$ bits are denoted by t_{hash}, t_{prg}, t_{rg}, and t_{Hash}, respectively.

In Table 1.15 on the next page we show some estimates related to the security of the proposed schemes. We suppose that the birthday attack is the best one in finding either a pre-image or a collision of a hash function. Thus, if the output of a hash values has bit length s, then an adversary succeeds with probability $< \frac{1}{2}$ through $2^{\frac{s}{2}}$ hash computations. That is why we consider $\frac{t_{\text{hash}}}{\epsilon_{\text{hash}}} = 2^{\frac{s}{2}+1}$. Now, we set $t_{\text{hash}} = 2^{\frac{s}{4}}$ in order to estimate the security of the proposed schemes. For the security and forward security of iMss we have used $N = 20$, which permits upto 2^{20} possible signatures. For the security of iMss with split key we have used $N = 20$, which permits up to 2^{40} possible signatures. We have taken into account the

Table 1.14: Time for generating, signing and verifying.

	iLDots
Generation key	$t_{\text{GK}} = t_{\text{rg}} + s''(t_{\text{prg}} + 3t_{\text{hash}}) + t_{\text{Hash}}$
Signature	$t_{\text{Sig}} = s''(t_{\text{prg}} + 3t_{\text{hash}})$
Verification	$t_{\text{Ver}} = 3s''t_{\text{hash}} + t_{\text{Hash}}$

	iMss	iMss with split key
Generation key	$2^N(t_{\text{prg}} + t_{\text{GK}} + 2^N t_{\text{hash}})$	$2^{\frac{N}{2}}(t_{\text{prg}} + t_{\text{GK}} + 2^{\frac{N}{2}} t_{\text{hash}})$
Signature	$N(2^{\frac{N}{2}} t_{\text{prg}} + t_{\text{GK}}) + t_{\text{Sig}}$	$\frac{N}{2}(2^{\frac{N}{4}} t_{\text{prg}} + t_{\text{GK}}) + t_{\text{Sig}} + 4t_{\text{GK}}$
Verification	$t_{\text{Ver}} + Nt_{\text{hash}}$	$2t_{\text{Ver}} + Nt_{\text{hash}}$

Here $s'' = \left\lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \right\rceil$

iLDots with pseudorandom key as the underlying one-time signature scheme. The notation is as follows: ots, iMss, iMss-fs and iMss-sk stand for one-time signature, improved Merkle signature scheme (without forward security), improved Merkle signature scheme which is forward secure, and improved Merkle signature scheme with split key, respectively.

Table 1.15: Security of the surveyed schemes. Several sizes of the output of the underlying hash function hash are considered.

security					
bit length s	160	224	256	384	512
$t_{\text{hash}}/\epsilon_{\text{hash}}$	2^{81}	2^{113}	2^{129}	2^{193}	2^{257}
$t_{\text{ots-prg}}/\epsilon_{\text{ots-prg}}$	$2^{67.9}$	$2^{99.4}$	$2^{115.2}$	$2^{178.7}$	$2^{242.3}$
$N \leq$	24	40	47	78	109
$t_{\text{iMss}}/\epsilon_{\text{iMss}}$	$2^{44.9}$	$2^{76.4}$	$2^{92.2}$	$2^{154.7}$	$2^{219.3}$
$t_{\text{iMss-fs}}/\epsilon_{\text{iMss-fs}}$	$2^{42.9}$	$2^{74.4}$	$2^{90.2}$	$2^{152.7}$	$2^{217.3}$
$t_{\text{iMss-sk}}/\epsilon_{\text{iMss-sk}}$	$2^{20.9}$	$2^{52.4}$	$2^{68.2}$	$2^{130.7}$	$2^{195.3}$

Brassard et al. present in [BHT88] a quantum algorithm for finding a collision in arbitrary r-to-one function $F : X \to Y$, such an algorithm returns a collision

after an expected number of $\Theta(\sqrt[3]{X})$ evaluations and uses space $\Theta(\sqrt[3]{X})$. Under the assumption that the considered hash functions are 2-to-1 and that the quantum computing does not reduce the time of hash value computations, we show the quantum security estimates in Table 1.16. In this case we consider $\frac{t_{\text{hash}}}{\epsilon_{\text{hash}}} = 2^{\frac{s}{3}+1}$. Now, we set $t_{\text{hash}} = 2^{\frac{s}{6}}$ in order to estimate the quantum security of the proposed schemes. For the security and forward security of iMss we have used $N = 10$, which permits upto 2^{10} possible signatures. For the quantum security of iMss with split key we have used $N = 10$, which permits up to 2^{20} possible signatures. We have taken into account the iLDots with pseudorandom key as the underlying one-time signature scheme.

Table 1.16: Quantum security of the surveyed schemes.

security					
bit length s	160	224	256	384	512
$t_{\text{hash}}/\epsilon_{\text{hash}}$	$2^{54.3}$	$2^{75.6}$	$2^{86.3}$	2^{129}	$2^{171.6}$
$t_{\text{ots}}/\epsilon_{\text{ots}}$	$2^{41.2}$	2^{62}	$2^{72.5}$	$2^{114.7}$	$2^{156.9}$
$N \leq$	11	21	26	46	67
$t_{\text{iMss}}/\epsilon_{\text{iMss}}$	$2^{28.2}$	2^{49}	$2^{59.5}$	$2^{101.7}$	$2^{143.9}$
$t_{\text{iMss-fs}}/\epsilon_{\text{iMss-fs}}$	$2^{26.2}$	2^{47}	$2^{57.5}$	$2^{99.7}$	$2^{141.9}$
$t_{\text{iMss-sk}}/\epsilon_{\text{iMss-sk}}$	$2^{14.2}$	2^{35}	$2^{45.5}$	$2^{87.7}$	$2^{129.9}$

1.8 An on-line/off-line signature scheme

Even et. al presented in [EGM90] an on-line/off-line signature scheme. Roughly speaking, an on-line/off-line signature scheme has (I) a key pair divided into: a home-key and a user-key, and (II) an "on-line" process for signing messages with the user-key and an "off-line" process for pre-computing, with the home-key, necessary values during the on-line process. Even et. al proposed in their work a generic scheme using a conventional signature and a one-time signature schemes. We replace these signature schemes by two (not necessary different) multi-time

signature schemes. The off-line phase of our proposal consists of: (I) generating an instance of the multi-time signature scheme as the user-key and (II) signing the public user-key with the private home-key. The on-line phase consists of signing with the private user-key. A signature of the proposed scheme consists of $(\zeta, P_{\mathcal{U}}, \tau)$, where ζ is a signature made by the private user-key, $P_{\mathcal{U}}$ is the public user-key and τ is the signature of $P_{\mathcal{U}}$ with the private home-key. As a practical on-line/off-line signature scheme we use private keys of instances of the iMss and of the iMss with split key for the home-keys and user-keys, respectively. We also give some estimates of the efficiency of the proposed practical version and compare them with RSA.

1.8.1 Our Proposal

Next we reformulate the definition of an on-line/off-line signature scheme given by Even et. al in [EGM90]. The goal of this kind of schemes is to take advantage of two signature schemes. The first scheme could be "slower" than the second one, whereas the second one could have a much more restricted number of possible signatures than the first one. The combination of this two schemes permits to sign "quickly" (on-line phase) with the second scheme, while the "slow" computation of a signature made by the first scheme can be done off-line. The first scheme is used to validate the public key needed to verify signatures of messages. Other advantage of this kind of scheme is that the private keys of the underlying schemes do not have to be stored in the same medium.

Definition 1.17 *Let consider two signature schemes* $\mathcal{H}\mathsf{Sign} = (\mathcal{H}\mathsf{Gen}, \mathcal{H}\mathsf{Sig}, \mathcal{H}\mathsf{Ver})$ *and* $\mathcal{U}\mathsf{Sign} = (\mathcal{U}\mathsf{Gen}, \mathcal{U}\mathsf{Sig}, \mathcal{U}\mathsf{Ver})$. *An* on-line/off-line signature scheme Sign *is a quadruple,* $(\mathsf{Gen}, \mathsf{Offline}, \mathsf{Sig}, \mathsf{Ver})$, *of probabilistic polynomial-time algorithms defined as follows:*

Key generation algorithm Gen. *Having as input the security parameter* 1^s *and maybe some other information* $(I_{\mathcal{H}}, I_{\mathcal{U}})$, Gen *computes two pairs* $(\mathcal{H}_S, \mathcal{H}_P) \leftarrow \mathcal{H}\mathsf{Gen}(1^s, I_{\mathcal{H}})$ *and* $(\mathcal{U}_S, \mathcal{U}_P) \leftarrow \mathcal{U}\mathsf{Gen}(1^s, I_{\mathcal{U}})$. *After that* Gen *computes* $\tau = \mathcal{H}\mathsf{Sig}(\mathcal{U}_P, \mathcal{H}_S)$. $\mathcal{H}_S$ *and* $\mathcal{H}_P$ *are called the* secret *and* public home-key, *respectively.* $\mathcal{U}_S$ *and* $\mathcal{U}_P$ *are called the* secret *and* public user-key, *respectively.*

Finally, Gen *outputs* $\mathcal{H}_P$ *as the public key of the scheme,* $(\mathcal{H}_S,\mathcal{U}_S)$ *as the secret key of the scheme and* $(\tau,\mathcal{U}_P,I_{\mathcal{U}})$ *as auxiliary data.*

Off-line algorithm Offline **(off-line phase).** *Given as input the private home-key* $\mathcal{H}_S$ *and the auxiliary data* $I_{\mathcal{U}}$, Offline *computes a new pair of private and public user-keys* $(\mathcal{U}_S,\mathcal{U}_P) \leftarrow \mathcal{U}\mathsf{Gen}(1^s, I_{\mathcal{U}})$ *and a new* $\tau = \mathcal{H}\mathsf{Sig}(\mathcal{U}_P, \mathcal{H}_S)$. *The old private user-key must be deleted.*

Signing algorithm Sig **(on-line phase).** *Having as input the private user-key* $\mathcal{U}_S$, *the auxiliary data* $(\tau,\mathcal{U}_P)$ *and a message* M, Sig *obtains* $\varsigma = \mathcal{U}\mathsf{Sig}(M,\mathcal{U}_S)$ *and outputs* $\sigma = (\varsigma,\mathcal{U}_P,\tau)$ *as the signature of the message* M.

Verification algorithm Ver. *Given as input a message* M, *a signature* σ *and the public key* $\mathcal{H}_P$, Ver *accepts the signature* σ *if* $\mathcal{U}\mathsf{Ver}(M,\varsigma,\mathcal{U}_P) = true$ *and* $\mathcal{H}\mathsf{Ver}(\mathcal{U}_P,\tau,\mathcal{H}_P) = true$. Ver *rejects it otherwise. Here* $\sigma = (\varsigma,\mathcal{U}_P,\tau)$.

Now, our goal is to provide from two multi-time signature schemes an on-line/off-line signature scheme which is also forward secure. In this case, we require one of them to be also forward secure. Let $\mathsf{mt_1Sign}$ and $\mathsf{mt_2Sign}$ be two multi-time signature schemes. Assume that one of these schemes is forward secure and each period of it consists of certain number of signatures instead of time, besides that the update of the private key can be done through the signature process, i. e. suppose that $\mathsf{mt_1Sign} = (\mathsf{mt_1Gen}, \mathsf{mt_1Sig}, \mathsf{mt_1Ver}, \mathsf{mt_1Upd})$ and $\mathsf{mt_2Sign} = (\mathsf{mt_2Gen}, \mathsf{mt_2Sig}, \mathsf{mt_2Ver})$, and that m_1 is the number of possible signatures in each period of $\mathsf{mt_1Sign}$, and if Upd is not called, then the private key of $\mathsf{mt_1Sign}$ is intrinsically updated in $\mathsf{mt_1Sig}$ after m_1 signatures. If the number of possible signatures of $\mathsf{mt_iSign}$ is M_i, where $i = 1, 2$, then we will construct an on-line/off-line signature scheme with the following properties.

- It is forward secure.
- Its maximal number of signatures is M_1M_2.
- Its period consists up to M_2m_1 signatures.
- The time needed for its signature algorithm is the one needed by $\mathsf{mt_2Sig}$.

- The time needed for its off-line phase is the one needed by $\mathsf{mt_2Gen}$ and $\mathsf{mt_1Sig}$.
- The time needed for its update process is the one needed by $\mathsf{mt_2Gen}$, $\mathsf{mt_1Sig}$ and $\mathsf{mt_1Upd}$.

Next we present our proposal of an on-line/off-line and forward-secure signature scheme.

Scheme 1.18 Let $\mathsf{mt_1Sign}$ and $\mathsf{mt_2Sign}$ be two multi-time signature schemes, where $\mathsf{mt_1Sign} = (\mathsf{mt_1Gen}, \mathsf{mt_1Sig}, \mathsf{mt_1Ver}, \mathsf{mt_1Upd})$ is additionally forward-secure and $\mathsf{mt_2Sign} = (\mathsf{mt_2Gen}, \mathsf{mt_2Sig}, \mathsf{mt_2Ver})$. An *on-line/off-line and forward-secure signature scheme* is a quintuple $\mathsf{Sign} = (\mathsf{Gen}, \mathsf{Offline}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Upd})$ defined as follows.

The algorithms $\mathsf{Offline}$, Sig and Ver remain the same as in Scheme 1.17 on page 52. The algorithm Gen additionally outputs a counter i as auxiliary data to indicate the number of previous periods. The number of periods relies intrinsically on the number of possible signatures of $\mathsf{mt_2Sign}$.

Secret key update algorithm Upd**.** Given as input the counter i and the private key with its auxiliary data, Upd updates the private home-key by a call to $\mathsf{mt_1Upd}$. At this point the counter i should be already incremented by one. Upd obtains a new pair of user-keys $(\mathcal{U}_S, \mathcal{U}_P) \leftarrow \mathsf{mt_2Gen}(1^s, \mathcal{I}_\mathcal{U})$. After that Upd computes $\tau \leftarrow \mathsf{mt_1Sig}(\mathcal{U}_P, \mathcal{H}_S)$. Finally, it outputs $\mathcal{H}_P$, $(\mathcal{H}_S, \mathcal{U}_S)$ and $(\tau, \mathcal{U}_P, \mathcal{I}_\mathcal{U})$.

1.8.2 Security

The security of the proposed scheme is based on that of the underlying multi-time signature schemes.

Theorem 1.10 *Let* $\mathsf{mt_1Sign}$ *be a multi-time signature scheme which is* (t_1, ϵ_1, M_1) *forward secure and whose period consists of* m_1 *signatures instead of time, where* $m_1 < M_1$*, i. e. there exists at least two periods. Let* $\mathsf{mt_2Sign}$ *be a multi-time*

signature scheme which is (t_2, ϵ_2, M_2) *existentially unforgeable under adaptive chosen message attacks. Let define* $\epsilon = 2\left(\frac{c}{c-1}\right)\max\{\epsilon_1, c\epsilon_2\}$, *where* $c = \left\lceil \frac{M_1}{m_1} \right\rceil$. *If* $\epsilon \leq 1$, *the Scheme 1.18 on the facing page* Sig *is* (t, ϵ, M) *forward secure, where* $t_1, t_2 \geq t + M_1(t_{\mathsf{mt_1Gen}} + t_{\mathsf{mt_1Sig}}) + Mt_{\mathsf{mt_2Sig}}$, $M = M_1M_2$ *and each period consists of* m_1M_2 *signatures instead of time. Here* $t_{\mathsf{mt_1Gen}}, t_{\mathsf{mt_1Sig}}$ *and* $t_{\mathsf{mt_2Sig}}$ *are the time needed for* $\mathsf{mt_1Gen}$, $\mathsf{mt_1Sig}$ *and* $\mathsf{mt_2Sig}$, *respectively.*

Proof

The idea of the proof is as follows. If Sign is not forward-secure, then we are able to construct an Algorithm $\mathcal{A}$ which breaks one of two instances of the underlying schemes. The insecurity of Sign implies the existence of a forger $\mathcal{F}$. Thus, given as input a public key $\mathcal{H}_P$ of scheme $\mathsf{mt_1Sign}$, a public key $\mathcal{U}_P$ of scheme $\mathsf{mt_2Sign}$ and the necessary oracles, $\mathcal{A}$ must compute an instance of Sign. Then, $\mathcal{A}$ calls $\mathcal{F}$ with the public key $\mathcal{H}_P$ and signs queries from $\mathcal{F}$ with its created instance and with help of the oracles. With "high" probability $\mathcal{F}$ outputs a forgery of Sign for a period i' previous to the period i when the private key was required. From that forgery $\mathcal{A}$ outputs with "high" probability either a forgery of $\mathsf{mt_1Sign}$ or one of $\mathsf{mt_1Sign}$.

Now, suppose that Sign is not (t, ϵ, M) forward secure. Then there exists a probabilistic algorithm $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ which run within time t and its advantage is

$$Pr[\mathsf{Ver}(M, \langle i', \sigma\rangle, PK) = true \mid \begin{array}{l}(SK_0, PK) \leftarrow \mathsf{Gen}(1^s, \{\mathcal{I}_{\mathcal{H}}, \mathcal{I}_{\mathcal{U}}\});\\ (info, i) \leftarrow \mathcal{F}_1^{\mathsf{Sig}(\cdot, SK), \mathsf{Offline}, \mathsf{Upd}}(PK);\\ (M, \langle i', \sigma\rangle) \leftarrow \mathcal{F}_2(SK_i, info)\\ \text{ and } i' < i] < \epsilon,\end{array}$$

the adversary obtains the secret key at the $(i+1)$-th period and the forgery corresponds to the $(i'+1)$-th period. $\mathcal{F}_1$ makes no more than M_2 queries to the signing oracle in each period and a total of at most M queries to the signing oracle across all the stages. The probability is taken over the coin tosses of $\mathsf{Gen}, \mathsf{Sig}, \mathsf{Offline}, \mathsf{Upd}, \mathcal{F}_1$ and $\mathcal{F}_2$.

We construct an algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ which breaks one of the two underlying schemes. Given as input PK_{mt_1} and PK_{mt_2} the public keys of $\mathsf{mt_1Sign}$ and $\mathsf{mt_2Sign}$, respectively, and the oracles $\mathcal{O}_{\mathsf{mt_1Sig}}$, $\mathcal{O}_{\mathsf{mt_1Sig}}$, $\mathcal{O}_{\mathsf{mt_1Upd}}$ and $\mathcal{O}_{SK}$ for

signing with the private key corresponding to PK_{mt_1}, signing with the private key corresponding to PK_{mt_2}, updating the private key corresponding to PK_{mt_1} and obtaining the private key corresponding to PK_{mt_1}, respectively, (**Generation of a random instance of** Sign) $\mathcal{A}_1$ sets $\mathcal{H}_P \leftarrow PK_{mt_1}$ and chooses $0 \leq i_0 < c$ at random, where $c = \left\lceil \frac{M_1}{m_1} \right\rceil$. If $i_0 = 0$, $\mathcal{A}_1$ sets $\mathcal{U}_{P_0} \leftarrow PK_{mt_2}$. Otherwise, $\mathcal{A}_1$ computes $(\mathcal{U}_{S_0},\mathcal{U}_{P_0}) \leftarrow \mathsf{mt_2Gen}(1^s,\mathcal{I}_{\mathcal{U}})$. $\mathcal{A}_1$ obtains $\tau \leftarrow \mathcal{O}_{\mathsf{mt_1Sig}}(\mathcal{U}_{P_0})$. **Signature oracle of** $\mathcal{A}_1$ **for** $\mathcal{F}_1$: Let i be the counter of previous periods ($0 \leq i < c$). Having as input a message $\mathcal{M}_{(i,j)}$, where $0 \leq j < M_2$, $\mathcal{A}_1$ computes $\varsigma \leftarrow \mathsf{mt_2Sig}(\mathcal{M}_{(i,j)},\mathcal{U}_{S_i})$ in case that $i \neq i_0$. Otherwise, $\mathcal{A}_1$ computes $\varsigma \leftarrow \mathcal{O}_{\mathsf{mt_1Sig}}(\mathcal{M}_{(i,j)})$. $\mathcal{A}_1$ keeps $\mathcal{M}_{(i,j)}$ and returns $(\varsigma,\mathcal{U}_{P_i},\tau)$ as the signature of the message. **Off-line oracle of** $\mathcal{A}_1$ **for** $\mathcal{F}_1$: i is increased by one. If $i = i_0$, $\mathcal{A}_1$ sets $\mathcal{U}_{P_i} \leftarrow PK_{mt_2}$. Otherwise, $\mathcal{A}_1$ computes $(\mathcal{U}_{S_i},\mathcal{U}_{P_i}) \leftarrow \mathsf{mt_2Gen}(1^s,\mathcal{I}_{\mathcal{U}})$. $\mathcal{A}_1$ obtains $\tau \leftarrow \mathcal{O}_{\mathsf{mt_1Sig}}(\mathcal{U}_{P_i})$. $\mathcal{A}_1$ deletes the old private user-key, if any exists, and keeps the public user-key. **Update oracle of** $\mathcal{A}_1$ **for** $\mathcal{F}_1$: $\mathcal{A}_1$ calls $\mathsf{mt_1Upd}$. At this point the counter i should be already incremented by one. As for the off-line oracle if $i = i_0$, $\mathcal{A}_1$ sets $\mathcal{U}_{P_i} \leftarrow PK_{mt_2}$. Otherwise, $\mathcal{A}_1$ computes $(\mathcal{U}_{S_i},\mathcal{U}_{P_i}) \leftarrow \mathsf{mt_2Gen}(1^s,\mathcal{I}_{\mathcal{U}})$. $\mathcal{A}_1$ obtains $\tau \leftarrow \mathcal{O}_{\mathsf{mt_1Sig}}(\mathcal{U}_{P_i})$. $\mathcal{A}_1$ deletes the old private user-key, if any exists. **Secret key obtainment oracle of** $\mathcal{A}$ **for** $\mathcal{F}$: if $i = i_0$, then $\mathcal{A}$ outputs "failure" and stops trying to find a forgery. Otherwise $\mathcal{A}_1$ has already $\mathcal{U}_{S_i}$, then obtains $\mathcal{H}_S \leftarrow \mathcal{O}_{SK}$ and finally, outputs $(\mathcal{H}_S,\mathcal{U}_{S_i})$ as the secret key.

Use of $\mathcal{F}$ **to obtain a forgery:** Now, $\mathcal{A}_1$ calls $\mathcal{F}_1(\mathcal{H}_P)$ and signs queries of $\mathcal{F}_1$ as described above. The off-line oracle must be called at most each M_2 signatures, when the update oracle has not been called in the middle of those M_2 signatures. The update oracle can be called at any time, but not more than c. $\mathcal{A}_1$ obtains $(info, i)$, the output of $\mathcal{F}_1$. With probability at least of $1 - \frac{1}{c}$ we have that $i \neq i_0$, then with that probability $\mathcal{A}$ obtains the private key $(\mathcal{H}_S,\mathcal{U}_{S_i})$. Given as input $(info, \{\mathcal{M}_{(k,j)}\}_{0 \leq k < i}, i, \mathcal{H}_S, \mathcal{U}_{S_i})$, $\mathcal{A}_2$ calls $\mathcal{F}_2(info, (\mathcal{H}_S,\mathcal{U}_{S_i}))$ in order to obtain a forgery $(\mathcal{M}, \langle i', \sigma \rangle)$ of Sign with probability at least ϵ. If $\langle i', \sigma \rangle$ is a forgery, then $\langle i', \sigma \rangle$ has the form $(\varsigma,\mathcal{U}'_{P_{i'}},\tau)$. $\mathcal{A}_2$ outputs $(\mathcal{U}'_{P_{i'}},\tau)$ as forgery of $\mathsf{mt_1Sign}$ in case of $\mathcal{U}_{P_{i'}} \neq \mathcal{U}'_{P_{i'}}$ or outputs $(\mathcal{M},\varsigma)$ as forgery of $\mathsf{mt_2Sign}$ otherwise.

Success of $\mathcal{A}$ **in forging a signature:** Let $\mathcal{P}$ be the probability that $\mathcal{U}'_{P_{i'}} \neq \mathcal{U}_{P_{i'}}$. Recall that the probability that $\mathcal{F}$ obtains the private key of Sign when it

requires it is at least $1 - \frac{1}{c}$, and the probability that $\mathcal{F}$ forges a signature given the private key is at least ϵ. So, if $\mathcal{P} \geq \frac{1}{2}$, $\mathcal{A}$ outputs a forgery of $\mathsf{mt_1Sign}$ with probability $\geq \left(1 - \frac{1}{c}\right)\frac{\epsilon}{2} \geq \epsilon_1$ within time t_1. On the other hand, if $\mathcal{P} < \frac{1}{2}$, then the probability that $i' = i_0$ is $\frac{1}{c}$. Thus, $\mathcal{A}$ outputs a forgery of $\mathsf{mt_2Sign}$ with probability $> \left(1 - \frac{1}{c}\right)\frac{1}{c}\frac{\epsilon}{2} \geq \epsilon_2$ within time t_2. Any of the previous cases contradicts the assumption of either the forward security of $\mathsf{mt_1Sign}$ or the security of $\mathsf{mt_2Sign}$.

□

1.8.3 Efficiency

Next we show the size and the needed time of the elements of the proposed scheme in terms of the elements of the underlying schemes.

In Table 1.17 we show the size of the elements of the proposed scheme. The values $\ell(SK_i), \ell(PK_i)$ and $\ell(Sig_i)$ are the bit length of the private key, the public key and the signature, respectively, of the signature scheme $\mathsf{mt_iSign}$ for $i = 1, 2$.

Table 1.17: Efficiency of the proposed on-line/off-line scheme (size)

	Size in bits
Public key	$\ell(PK_1)$
Secret home-key	$\ell(SK_1)$
Secret user-key	$\ell(SK_2)$
Auxiliary values	$\ell(PK_2) + \ell(Sig_1)$
Signature	$\ell(Sig_2) + \ell(PK_2) + \ell(Sig_1)$

In Table 1.18 on the following page we show the time needed by the algorithms employed by the proposed scheme. The values $t_{\mathsf{mt_iGen}}, t_{\mathsf{mt_iSig}}$ and $t_{\mathsf{mt_iVer}}$ are the times needed by the key generation, signature and verification algorithms, respectively, of the scheme $\mathsf{mt_iSign}$, for $i = 1, 2$. The value $t_{\mathsf{mt_1Upd}}$ is the time needed by the secret key update algorithm of $\mathsf{mt_1Sign}$.

Table 1.18: Efficiency of the proposed on-line/off-line scheme (time)

	Time
Key generation	$t_{\mathsf{mt_1Gen}} + t_{\mathsf{mt_2Gen}}$
Signature	$t_{\mathsf{mt_2Sig}}$
Verification	$t_{\mathsf{mt_1Ver}} + t_{\mathsf{mt_2Ver}}$
Off-line phase	$t_{\mathsf{mt_2Gen}} + t_{\mathsf{mt_1Sig}}$
Update	$t_{\mathsf{mt_1Upd}} + t_{\mathsf{mt_2Gen}} + t_{\mathsf{mt_1Sig}}$

1.8.4 A Practical Case

In this Section we present a practical case of the proposed on-line/off-line signature scheme. We use the iMss and the iMss with split key as $\mathsf{mt_1Sign}$ and $\mathsf{mt_2Sign}$, respectively. Since multi-time signature schemes are more efficient than conventional ones for certain parameters and although the number of signatures are limited, our proposal is quite efficient and, in practical sense, unlimited in the number of possible signatures.

We will use the iMss as described in Scheme 1.15 on page 37, $\mathsf{M_1Sign} = (\mathsf{M_1Gen}, \mathsf{M_1Sig}, \mathsf{M_1Ver}, \mathsf{M_1Upd})$, to manage the home-keys and the iMss with split key as described in Scheme 1.16 on page 43, but without the secret key update algorithm, $\mathsf{M_2Sign} = (\mathsf{M_2Gen}, \mathsf{M_2Sig}, \mathsf{M_2Ver})$, to manage the user-keys. In this case we output as part of the public key the integer N_2 which denote the depth of the secondary tree. Thus the public key is (R, N_1, N_2), where R is the root of the main Merkle tree, N_1 and N_2 are the depth of the main and secondary Merkle tree, respectively. Besides, note that the update algorithm is not necessary for the application of the iMss with split key. After certain number of signatures, the secret key of an instance of the iMss with split key is updated through the signature algorithm.

In Table 1.19 on the next page we present the size of the private and public key and the size of a signature. In that table s is the length of the output of the underlying hash function, $s'' = \frac{s+3\lfloor \log_2 s \rfloor}{2}$, l_{int} is the bit length of the counter, and N_i indicates that by $\mathsf{M_iSign}$ can be created up to 2^{N_i} signatures, for $i = 1, 2$. With the proposed scheme can be created up to 2^N signatures, where $N = N_1 + N_2$.

Table 1.19: Size of the keys and signature

	Size in bits
Public key	$s + 2l_{\text{int}}$
Secret home-key	$s\left(2^{\lceil \frac{N_1}{2} \rceil} + 3N_1\right) + l_{\text{int}}$
Secret user-key	$s\left(2^{\lceil \frac{N_2}{4} \rceil + 1} + 4N_2 + s'' + 1\right) + 3l_{\text{int}}$
Auxiliary values	$s(s'' + N_1 + 1) + l_{\text{int}}$
Signature	$s(3s'' + 3 + N_2 + N_1) + 3l_{\text{int}}$

An advantage of this scheme is that the user-key and the home-key do not have to be stored in the same place. Another advantage is also that the off-line phase could be done either at the beginning or at the end of the day when the private key is used. In this last case, if we use $N_1 = 14$ and $N_2 = 16$, we are able to sign up to 2^{16} times per day during 2^{14} days. In Table 1.20 we give two examples of the proposed scheme with hash functions whose size of the output is 256 and 384. The auxiliary values has been included into the private user-key, since those values are part of the signature which is made by such private key.

Table 1.20: Size of the keys and signature

			home-key in Kb		user-key in Kb		signature in Kb		security $\frac{t}{\epsilon}$	
N_1	N_2	N	256	384	256	384	256	384	256	384
14	16	30	4.66	6.98	12.27	33.96	14.17	30.25	2^{94}	2^{158}
20	20	40	33.87	50.81	13.96	36.48	14.46	30.7	2^{88}	2^{152}

1.9 Pseudocode for the proposed algorithms

From page 61 to page 69 we give a pseudo-code for the algorithms presented in this chapter.

The parameter N allows to create up to 2^N signatures. The iMss with split key is much more efficient than the oMss and iMss. During the key generation algorithm 2^N one-time key pairs and pseudorandom numbers must be computed by iMss, while only $2^{\frac{N}{2}}$ one-time key pairs and pseudorandom numbers plus a one-time signature must be computed in iMss with split key.

During the signature algorithm a new key pair of the Merkle signature scheme must be computed after 2^N signed messages. In this case, a signature requires the same time as a generation key every 2^N signed messages. Such time can be spread along the signature algorithm itself. Two nodes of the Merkle tree can be computed right after each created signature as part of the signature process. We describe it in Algorithm 1.5 on page 64, which is employed in Algorithm 1.2.

By Szydlo algorithm up to N Merkle tree's leaves are computed and are spread. Therefore, we must have an efficient way to compute them. If $0 \leq p \leq N$ and $q = N - p$, then we can store 2^q PRG's outputs in order to compute each Merkle tree's leaf with up to 2^p calls to the PRG.

In Algorithm 1.8 on page 67 we set $p = \lfloor \frac{N}{2} \rfloor$. The auxiliary values $Auth_k$ are needed in the Szydlo algorithm, that is why they are computed, too.

The Szydlo algorithm `Szydlo_method` is used as a black box in Algorithm 1.9 on page 68 in order to compute the i-th sibling path of a Merkle tree's leaf needed to compute a Merkle signature in an efficient way. This method uses as input the values $Auth_k$, which are computed in Algorithm 1.8 on page 67. The auxiliary $Keep_k$ and $Need_k$ are updated during each use of `Szydlo_method`, for $0 \leq k < N$. The Szydlo algorithm outputs the sibling paths sequentially. The values $Keep_k$ and $Need_k$ can be initialized to the null string for $i = 0$.

In the Szydlo algorithm a leaf must be computed calling `Leaf-Calc`. In Algorithm 1.6 on page 66 the value of leaf $leaf$ is obtained from $Pr = \{\chi_{i-1}\} \cup \{\chi_{l2^p-1}\}_{\lfloor \frac{i}{2^N} \rfloor < l < 2^q}$.

Therefore, the signature algorithm of the iMss with split key is equivalent to two calls to the signature algorithm of the iMss with parameter $\frac{N}{2}$ plus the computation of two nodes of a Merkle tree. The verification algorithm of the iMss with split key is equivalent to two calls to the verification algorithm of the iMss, but with parameter $\frac{N}{2}$.

The Merkle verification algorithm is presented in Algorithm 1.10 on page 69. The improved versions for the key generation, the signature and the verification algorithms are presented in Algorithms 1.1, 1.2 on the following page and 1.3 on page 63, respectively.

In all algorithms we assume that the underlying hash function H and the one-time signature scheme $\mathsf{1Sign} = (\mathsf{1Gen}, \mathsf{1Sig}, \mathsf{1Ver})$ are known by all parties. Recall that the security parameter s is inherent to the length of the output of H.

The improved Merkle signature scheme with split key generation algorithm is described by the algorithms 1.1, 1.2, 1.3 and 1.4.

Algorithm 1.1 iMskGen***Key Generation Algorithm of the iMss with split key:*** obtainment of a key pair and auxiliary values for signing efficiently

`Input:` $N = 2N'$

`Output:` Private and public keys (Pr, Pu) and auxiliary values

`Procedure:`

$(Pr_A, Pu_A, \{Auth_{A,k}\}_{0 \le k < N'}) \leftarrow \mathsf{iMGen}(N')$;
$(Pr_B, Pu_B, \{Auth_{B,k}\}_{0 \le k < N'}) \leftarrow \mathsf{iMGen}(N')$;
$\chi \leftarrow \chi_{A,0}$ / * $\chi_{A,0}$ extracted from Pr_A * /
$\chi_C \leftarrow \{0,1\}^s$ / * chosen at random $(\chi_{C,-1})$ * /
i_mod_$2^{N'} \leftarrow 0$;
i_div_$2^{N'}$__plus_1 $\leftarrow 0$;
$j \leftarrow N'$;
$FL \leftarrow true$;
k_at_level $\leftarrow \vec{0}$;
$\zeta \leftarrow \mathsf{iMSig}(Pu_B, Pr_A, \text{i\_div\_}2^{N'}\text{\_\_plus\_1}, \{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N'-1})$;
$Pu = (N', Pu_A)$ / * the public key * /
$Pr = (Pr_A, Pr_B)$ / * the private key * /
/*
Keep the auxiliary values
$\{Auth_{X,k}, Keep_{X,k}, Need_{X,k}\}_{0 \le k < N', X=A,B}, (\zeta, Pu_B), N, \text{i\_mod\_}2^{N'} = 0,$
i_div_$2^{N'}$__plus_1 $= 1, j, FL$ and k_at_level
*/
return $(Pr, Pu,$ and the auxiliary values);

Note: The functions iMGen and iMSig are described in Algorithms 1.8 on page 67 and 1.9 on page 68, respectively.

End of Algorithm 1.1

Algorithm 1.2 iMskSig ***Signature Algorithm of the iMss with split key:*** Computation of a signature. Obtainment of a sibling path by Szydlo method.
`Input:` Message M, the private key $Pr = (Pr_A, Pr_B,$ and auxiliary values)
`Output:` Signature $\sigma = (\tau, Pu_B, \zeta)$ and actualization of the auxiliary values
`Procedure:`

$/* N = 2N' */$
$\tau \leftarrow \mathsf{MSig}(M, Pr_B, \text{i\_mod\_}2^{N'}, \{Auth_{B,k}, Keep_{B,k}, Need_{B,k}\}_{k=0}^{N'-1})$;
$\sigma \leftarrow (\tau, Pu_B, \zeta)$;
$\mathsf{miMGen}(N', Pr_C, Pu_C, \chi_C, \{Auth_{C,k}\}_{k=0}^{N'-1}, j, \text{k\_at\_level}, stack, FL)$;
if $(\text{i\_div\_}2^{N'}\text{\_plus\_}1 <= \text{i\_mod\_}2^{N'})$
  $NLCom(\chi, \text{i\_mod\_}2^{N'}, index, needed\_index, needed\_leaf\_values)$;
fi
$\text{i\_mod\_}2^{N'} \leftarrow \text{i\_mod\_}2^{N'} + 1$;
if $(0 == \text{i\_mod\_}2^{N'} \mod 2^{N'})$
  $(Pr_B, Pu_B) \leftarrow (Pr_C, Pu_C)$;
  $(Pr_C, Pu_C) \leftarrow empty\_key\_pair$;
  $\chi_C \leftarrow \{0,1\}^s$ $/*$ chosen at random $*/$
  $j \leftarrow N'$;
  $\text{k\_at\_level} \leftarrow \vec{0}$;
  $stack \leftarrow empty\_stack$;
  $FL \leftarrow true$;
  $\chi \leftarrow \chi_{A,\text{i\_div\_}2^{N'}\text{\_plus\_}1}$;
  $index \leftarrow 0$;
  Obtain needed_index from Szydlo_method with $\{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N'-1}$
  $needed\_leaf\_values \leftarrow vector\_of\_null\_bit\_strings$;
  $\zeta \leftarrow \mathsf{MSig}(Pu_B, Pr_A, \text{i\_div\_}2^{N'}\text{\_plus\_}1, \{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N'-1})$;
  $\text{i\_mod\_}2^{N} \leftarrow 0$;
fi
return σ;

Note: The functions miMGen, iMSig and $NLCom$ are described in Algorithms 1.5 on the following page, 1.9 on page 68 and 1.6 on page 66, respectively.

End of Algorithm 1.2

Algorithm 1.3 iMskVer ***Verification Algorithm of the iMss with split key:*** Verification of the validity of a signature with respect to a message
`Input:` Message M, signature $\sigma = (\tau, Y, \zeta)$ and public key Pu
`Output:` $true$ for a valid one. Otherwise $false$.
`Procedure:`

$/* N = 2N' */$
if (iMVer$(M, \tau, (N', Y)) == false$)
  return $false$;
fi
if (iMVer$(Y, \zeta, Pu) == false$)
  return $false$;
fi
return $true$

Note: The function iMVer is described in Algorithm 1.10 on page 69.

End of Algorithm 1.3

Algorithm 1.4 iMskUpd ***Update Algorithm of the iMss with split key:*** Actualization of the private key and auxiliary data needed for the next period.
`Input:` Private key (Pr_A, Pr_B) and auxiliary data.
`Output:` Private key for the following period and auxiliary data.
`Procedure:`

while $(j \neq 0)$ do
    miMGen$(N', Pr_C, Pu_C, \chi_C, \{Auth_{C,k}\}_{k=0}^{N'-1}, j,$ k_at_level$, stack, FL)$;
od
$(Pr_B, Pu_B) \leftarrow (Pr_C, Pu_C)$;
$(Pr_C, Pu_C) \leftarrow empty\_key\_pair$;
$\chi_C \leftarrow \{0,1\}^s$ $/*$ chosen at random $*/$

$j \leftarrow N'$;
k_at_level $\leftarrow \vec{0}$;
stack $\leftarrow$ *empty_stack*;
$FL \leftarrow true$;
$\chi \leftarrow \chi_{A,\text{i\_div\_}2^{N'}\text{\_\_plus\_}1}$;
index $\leftarrow 0$;
Obtain needed_index from Szydlo_method with $\{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N'-1}$
needed_leaf_values $\leftarrow$ *vector_of_null_bit_strings*;
$\zeta \leftarrow \mathsf{MSig}(Pu_B, Pr_A, \text{i\_div\_}2^{N'}\text{\_\_plus\_}1, \{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N'-1})$;
i_mod_$2^N \leftarrow 0$;
return (Pr_A, Pr_B) and the auxiliary data;

End of Algorithm 1.4

In Algorithm 1.5 is computed two nodes of a third Merkle tree during each call. This algorithm is an auxiliary one employed in Algorithm 1.2.

Algorithm 1.5 miMGen ***modified Key Generation Algorithm of the iMss:*** Obtainment of a key pair and auxiliary values through computation of two nodes of the Merkle tree by each call
`Input:` $N = 2N'$, (Pr_C, Pu_C), χ, $\{Auth_{C,k}\}_{0 \le k < N'}$, j, k_at_level, stack and FL
`Output:` updated (Pr_C, Pu_C), χ, $\{Auth_{C,k}\}_{0 \le k < N'}$, j, k_at_level, stack and FL
`Procedure:`

int *computed*;
hash_value $K, left, right, root$;
computed $\leftarrow 0$;
while $(0 < j)$ do
  if $(FL == true)$
    $j \leftarrow N'$;
    if $(0 ==$ k_at_level$[N'] \mod 2^p)$
      update_Pr(χ, Pr_C);
      $/* \{\chi_{l2^p-1}\}_{0 \le l < 2^q}$ is being stored $*/$
    fi
    $(\chi, seed) \leftarrow PRG(\chi)$;

```
    (Pr_1s, Pu_1s) ← 1Gen(seed);
    K ← H(Pu_1s);
    computed ← computed + 1;
    push(stack, K, j, k_at_level[j]);
    k_at_level[j] ← k_at_level[j] + 1;
  fi
  if (computed == 2)
    FL = not(two_at_same_level(stack));
    return;
  fi
  while (two_at_same_level(stack)) do
      pop(stack, right, j, right_at_level_j);
      pop(stack, left, j, left_at_level_j);
      if (right_at_level[j] == 1)
        Auth_{C,N'-j} ← right;
      fi
      j ← j − 1;
      K ← H(concat(left, right));
      computed ← computed + 1;
      push(stack, K, j, k_at_level[j]);
      k_at_level[j] ← k_at_level[j] + 1;
      if (computed == 2)
        FL ← not(two_at_same_level(stack));
        return;
      fi
  od
  FL ← true;
od
pop(stack, root, j, k_at_level[0]);
Pu_C ← (N', root);
Pr_C ← ({χ_{l2^p−1}}_{0≤l<2^q}, {Auth_{C,k}}_{k=0}^{N'−1});
return;
```

End of Algorithm 1.5

The auxiliary algorithms 1.6 and 1.7 compute a leaf of the Merkle tree in two

different fashion.

In Algorithm 1.6 one seed at a time is computed. A certain leaf needed by Szydlo method is computed after that the corresponding seed is reached. In this way, the computation of the needed leaves for a signature of a main Merkle tree is the iMss with split key is spread during the signature process.

In Algorithm 1.7 a leaf is computed at once. This process is needed by the signature of the iMss.

Algorithm 1.6 *NLCom* ***computation of a needed leaf:*** Computation of the needed leaf values in the Szydlo method, which will be used in the improved Merkle Signature Algorithm

`Input:` $\chi, leaf\_index, index, needed\_index, needed\_leaf\_values$

`Output:` updated values for:$\chi, index, needed\_index, needed\_leaf\_values$

`Procedure:`

$(\chi, seed) \leftarrow PRG(\chi)$;
if $(leaf\_index == needed\_index[index])$
  $(X, Y) \leftarrow$ 1Gen$(seed)$;
  $needed\_leaf\_values[index] \leftarrow H(Y)$;
  $index \leftarrow index + 1$;
fi

End of Algorithm 1.6

Algorithm 1.7 ***Computing a Merkle tree's leaf:*** `Leaf-Calc` Algorithm

`Input:` N, private key $Pr = \{\chi_{i-1}\} \cup \{\chi_{l2^p-1}\}_{\lfloor \frac{i}{2^N} \rfloor < l < 2^q}$, index $leaf$ and counter i

`Output:` the value of the leaf whose index is $leaf$

`Procedure:`

$k \leftarrow \left\lfloor \frac{leaf}{2^p} \right\rfloor$;
if $i < k2^p$
  then
    $(G, seed) \leftarrow PRG(\chi_{k2^p-1})$;
    $k \leftarrow k2^p$;

```
    else
        (G, seed) ← PRG(χ_{i−1});
        k ← i;
fi
for (count = k; count < leaf; count++) do
    (G, seed) ← PRG(G);
od
(Pr_leaf, Pu_leaf) ← 1Gen(seed);
L ← H(Pu_leaf);
return L;
```

End of Algorithm 1.7

The improved Merkle signature scheme (iMss) is described by the algorithms 1.8 to 1.10. Recall that the secret key update algorithm iMUpd is inherent to the signature algorithm iMSig.

Algorithm 1.8 iMGen ***Key Generation Algorithm of the iMss:*** Obtainment of the key pair and auxiliary values for signing efficiently

`Input:` N

`Output:` (Pr, Pu) and $\{Auth_k\}_{0 \le k < N}$

`Procedure:`

```
int j, k_at_level[N_max];
hash_value K;
χ ← {0,1}^s /* χ_{−1} at random */
k_at_level ← 0⃗;
j ← N;
while (0 < j) do
    j ← N;
    if (0 == k_at_level[N] mod 2^p)
      store(χ);
      /* {χ_{l2^p−1}}_{0≤l<2^q} is being stored */
    fi
    (χ, seed) ← PRG(χ);
```

$(Pr_{1s}, Pu_{1s}) \leftarrow \mathsf{1Gen}(seed)$;
$K \leftarrow H(Pu_{1s})$;
$push(stack, K, j, \text{k\_at\_level}[j])$;
$\text{k\_at\_level}[j] \leftarrow \text{k\_at\_level}[j] + 1$;
while $(two\_at\_same\_level(stack))$ **do**
$right \leftarrow pop(stack)$;
$left \leftarrow pop(stack)$;
if $(k\_at\_level[j] == 1)$
$Auth_{N-j} \leftarrow right.value$;
fi
$j \leftarrow left.j - 1$;
$K \leftarrow H(concat(left.value, right.value))$;
$push(stack, K, j, \text{k\_at\_level}[j])$;
$\text{k\_at\_level}[j] \leftarrow \text{k\_at\_level}[j] + 1$;
od
od
$root \leftarrow pop(stack)$;
$Pu \leftarrow (N, root.value)$;
$Pr \leftarrow (\{\chi_{l2^p-1}\}_{0 \le l < 2^q})$

End of Algorithm 1.8

Algorithm 1.9 iMSig ***Signature Algorithm of the iMss:*** Signature computation. Efficiently computation of a sibling path by Szydlo method
`Input:` Message M, private key Pr, counter i, and $\{Aut_k, Keep_k, Need_k\}_{0 \le k < N}$
`Output:` Signature $\tau = (i, \tau_{1s}, Y_{1s}, P_N, \ldots, P_1)$
`Procedure:`

$\chi_{i-1} \leftarrow extract\_from\_private\_key(Pr, i)$
$(\chi_i, seed_i) \leftarrow PRG(\chi_{i-1})$
$(Pr_{1s}, Pu_{1s}) \leftarrow \mathsf{1Gen}(s, seed_i)$
$\tau_{1s} \leftarrow \mathsf{1Sig}(M, Pr_{1s})$
$(P_N, \ldots, P_1) \leftarrow \text{Szydlo\_method}(\{Auth_k, Keep_k, Need_k\}_{0 \le k < N}, Pr)$
$\tau \leftarrow (i, \tau_{1s}, Pu_{1s}, P_N, \ldots, P_1)$
$(\chi_{i+1}, seed_{i+1}) \leftarrow PRG(\chi_i)$

Delete χ_{i-1} from Pr and include χ_i in Pr
Increment i by one

End of Algorithm 1.9

Algorithm 1.10 iMVer ***Verification Algorithm of the iMss:*** Verification of the validity of a signature with respect to a message
`Input:` Message M, signature $\tau = (i, \tau_{1s}, Y, P_N, \ldots, P_1)$ and public key (N, R)
`Output:` *true* for a valid signature. Otherwise, *false*
`Procedure:`

hash_value $W, left, right$;
int j;
if (1Ver$(\tau_{1s}, Y) == false$)
  return $false$;
fi
$W \leftarrow H(Y)$;
for ($j = N$; $0 < j$; $j--$) do
  if ($\lfloor \frac{i}{2^{N-j}} \rfloor \mod 2 == 0$)
    then
      $left \leftarrow W$;
      $right \leftarrow P_j$;
    else
      $left \leftarrow P_j$;
      $right \leftarrow W$;
  fi
  $W \leftarrow H(concat(left, right))$;
od
if ($W \neq R$)
  return $false$;
fi
return $true$;

End of Algorithm 1.10

Conclusion

We proved that the original Merkle signature scheme is existentially unforgeable under adaptive chosen message attack. We also presented a forward-secure signature scheme from any cryptographically secure pseudorandom bit generator, one-time signature scheme and hash function. As practical case we have used for our estimates and measurements hash functions like RIPEMD160, SHA-224, SHA-256, SHA-384 and SHA-512 [RFCa, FIPa, RFCb, FIPb, DBP96], and ISAAC as pseudorandom bit generator [ISA]. We have employed our improved version of the Lamport-Diffie one-time signature scheme as the underlying one-time signature scheme in our improvements to the Merkle multi-time signature scheme. We have implemented our improved versions of the Lamport-Diffie one-time signature scheme (with random key and with pseudorandom key) and have shown some experimental results of their efficiency and security. We have presented some schemes based on the work of Merkle. We have shown some estimates of the length of the keys and signature and we have proved their security. These schemes are also secure in the quantum computing epoch provided that the best quantum collision finder is the one proposed in [BHT88].

Chapter 2

Integer Multiplication

In this chapter we give an overview on some multiplication algorithms and estimate depending on the bit length of the integers to be multiplied, which algorithm is faster. We compare the number of multiplications of base words ($\mathcal{MOB}$) to be made by each multiplication algorithm taking also into account the number of additions needed. We view a multiplication of two base words as a unit of computation and an addition of two base words as a fraction of a unit of computation.

2.1 Introduction

The RSA and ElGamal alike cryptosystems are widely employed for signing digital documents. The security of those type of schemes relies on the difficulty in factoring big integers or in obtaining the discrete logarithmus. Shor presented in [Sho94] a quantum algorithm for solving the integer factorization problem and the discrete logarithm problem in polynomial time. In [VSS+01, VSS+00] Breyta et al. reported their successful implementations of the Shor's algorithm [Sho94] in a quantum system of seven qu-bits and of the Grover's algorithm [Gro96] in a quantum system of three qu-bits. They were able to create some quantum environments for factoring the integer 15 and for searching in a list of three elements. Although these quantum environments are still too small to be a threat on RSA and ElGamal alike cryptosystems, they could be a first step for the development of bigger quantum environments.

A question is raised: Could the RSA and ElGamal alike cryptosystems be still used during the development of quantum computers? In other words; how big should the bit length of the corresponding modulus be in order that the cryptosystem remains secure, under the assumption of the existence of limited quantum computers? Would the cryptosystem be still efficient with such a bit length of the modulus?

Modular exponentiations are employed in the RSA and ElGamal alike cryptosystem. Modular exponentiation involves modular multiplication and this last operation can be done with integer multiplications avoiding division by the modulus as presented by Montgomery in [Mon85]. Thus, if we want to compute modular exponentiations fast, we can take advantage of fast integer multiplication algorithms and apply an adequate one in order to speed up the RSA or ElGamal alike cryptosystems. Karatsuba [AK63, Knu97] and Schönhage [SS71], [SWV] provided multiplication methods faster than the naïve one, if the size of the two integers to multiply is big enough.

2.2 RSA and limited Quantum Computers

The product of two prime numbers is used as part of the public key in the RSA cryptosystem. Such a product is the modulus for the modular exponentiations needed in the algorithms of the cryptosystem. If the factorization of the modulus is known, then the private key can be efficiently computed. The security of the RSA cryptosystem is based on the assumption that the computation of certain types of roots is as difficult as computing the factorization of the modulus. The fastest factoring "classical" algorithm is the general number field sieve with a sub-exponential run-time $O(e^{1.9229(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}})$. Brent made in [Bre99] estimates of when a number of certain bit length could be factored. If B is the bit size of the RSA modulus and Y is the year when such number could be factored, then $Y = 13.24(B\log 2)^{\frac{1}{3}} + 1928.6$. According to this formula, we construct Table 2.1 on the facing page.

From the table we could infer the security in long term for each modulus size against classical computers.

Table 2.1: Factoring RSA modulus with classical computers

Bit size	Year
768	2009.86
1024	2018.04
2048	2041.29
4096	2070.57

Lenstra and Verheul made in [LV01] some key size recommendations for several cryptosystems, in particular for RSA, in different years up to 2050. They suggested not to use a 1028, 2054 and 4047 bit RSA key beyond 2002, 2023 and 2050, respectively. If we compare those suggestions with Table 2.1, we can see that messages ciphered with those kind of keys are secure for at least 15 years after their last suggested year of use.

Shor presented a couple of quantum algorithms in [Sho94] for solving the *integer factoring* and *discrete logarithm* problems in polynomial time. A consequence of his result is that the RSA cryptosystem will not be secure in the time of quantum computers any more.

We do not know if quantum computers can be built in such way that they can manage a big enough number of qu-bits and quantum gates to factor a integer of 2^{12} bits, for instance. On the other hand, a 7-qu bit quantum environment has been developed in [VSS+01] to implement the Shor's factoring algorithm to factorize $N = 15$ and also a 3-qu bit quantum system in [VSS+00] to implement the Grover's search algorithm [Gro96]. These sort of quantum environment are still harmless to be a threat to the RSA scheme.

In [Bea03] Beaurgard introduced a circuit for Shor's algorithm which uses $2n+3$ qu-bits and $O(n^3 \log_2 n)$ elementary quantum gates to factor an n-bit integer.

Therefore, even though the quantum computer could be gradually improved, the widely employed RSA cryptosystem will remain secure if an adequate size of the modulus is used. A disadvantage of increasing the bit length of the modulus is an increase in the consumption of time for encrypting and decrypting.

In Table 2.2 we show the time needed to sign and verify a message. We have used the cryptographic library OpenSSL [Ope] version 0.9.8. The experiments were made on a Pentium `III`, SuSE 9.3 at 1.1 GHz. As we can see the application of 2^n bit-length RSA keys is impractical with the current technology for $n \geq 13$.

Table 2.2: Timing for signing and verifying with RSA on a Pentium `III`, SuSE 9.3, at 1.1 GHz.

key bit length	Signature	Verification
512	3.0 ms	0.26 ms
1024	9.15 ms	0.45 ms
2048	58.41 ms	1.3 ms
4096	285.83 ms	4.42 ms
8192	1.9 sec	27.14 ms
16384	13.34 sec	68.72 ms
32768	1.81 min	0.238 sec
65536	14.20 min	0.887 sec
131072	2.07 hrs	3.703 sec

2.3 Multiplication Algorithms

The RSA cryptosystem makes use of modular exponentiation for encrypting and decrypting and intrinsically, multiplication of integers whose bit length is the one of the RSA-modulus. In this section we describe briefly the multiplication algorithms Karatsuba, Toom Cook and Schönhage in order to calculate the number of multiplications of base words ($\mathcal{MOB}$); in our estimates are included the additions of base words, too. We consider each addition between two base words as q multiplications between them, where $0 < q \leq 1$ should depend on the implementation. We use Zimmermann's version of the Schönhage algorithm [Zim92]. In that version a parameter κ must be previously fixed and it has been experimentally chosen for some sizes of the integers to be multiplied. Our goal is to find an adequate value

of κ in order to make least number of operations. As a result of this, we have that starting from 2^{17} bits Schönhage's algorithm is the fastest.

2.3.1 Notation

Let $\nu_0 \in \mathbb{Z}^+$ and $\mathcal{B} = 2^{2^{\nu_0}}$. A *base word* is an integer x such that $0 \leq x < \mathcal{B}$. For $z \in \mathbb{Z}^+$, the *base-word length* (or simply the length) of z is the number $\ell_{\mathcal{B}}(z)$ of base words needed to represent such z. In a similar way we define the bit length $\ell_2(z)$ of z. We define $\ell_{\mathcal{B}}(0) = \ell_2(0) = 1$ and for $z \in \mathbb{Z}^-$ we define $\ell_{\mathcal{B}}(z) := \ell_{\mathcal{B}}(-z)$ and $\ell_2(z) := \ell_2(-z)$.

Let $s_a = a_{m-1} \cdots a_1 a_0$ be a bit string, which is not the null string. We think of it as the integer $a = \sum_{j=0}^{m-1} a_j 2^j$ and we say that its bit length is $\ell_2(s_a) = m$ and its base-word length is $\ell_{\mathcal{B}}(s_a) = \lceil \frac{m}{2^{\nu_0}} \rceil 2^{\nu_0}$. So, if we want to add two integers $\alpha = \sum_{j=0}^{n_1-1} \alpha_j 2^j$ and $\beta = \sum_{j=0}^{n_2-1} \beta_j 2^j$, we can consider the bit strings $s_\alpha = \alpha_{m-1} \cdots \alpha_1 \alpha_0$ and $s_\beta = \beta_{m-1} \cdots \beta_1 \beta_0$, where $m = \max\{n_1, n_2\}$ and $\alpha_i = \beta_j = 0$ for $i \geq n_1$ and $j \geq n_2$. In this way we can define the number of additions of base words for a couple of integers of bit length m as $\mathrm{add}_{\mathcal{B}}(m) = \lceil \frac{m}{2^{\nu_0}} \rceil$.

We take a multiplication of two base words as a unit of computation and denote it by mult. We assume that an addition of two base words is equivalent to q unit of computation, where $0 < q \leq 1$ depends on the implementation. Therefore the addition of two m-bit integers is equivalent to $q\mathrm{add}_{\mathcal{B}}(m)$ multiplications of base words.

If $x, n, B \in \mathbb{Z}^+$, such that $B > 1$ and $x = \sum_{j=0}^{m} x_j B^j$, then the value $xB^n = \sum_{j=0}^{m} x_j B^{j+n}$ will be denoted by $x <<_B n$, and will be called an n *B-block right shift*. The symbol | represents the bitwise or.

2.3.2 Multiplication algorithms and $\mathcal{MOB}$

First, we describe some multiplication algorithms. Then we calculate the number of multiplications of base words needed for each one of them to multiply integers of the same length. Finally, we compare those quantities with each other. In all next subsections we assume $\mathcal{B} = 2^{2^{\nu_0}}$ for a previously set $\nu_0 \in \mathbb{Z}^+$. The bit length of a base words is 2^{ν_0}. We suppose that the two integers to be multiplied in each

algorithm have the same length N.

2.3.3 Naïve algorithm

The complexity of this algorithm is $O(N^2)$, where N is the length of the integers to be multiplied. We give a description of it in Algorithm 2.1.

Algorithm 2.1 ***Naïve algorithm:*** $\texttt{Na}(\alpha, \beta)$
`Input:` $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
`Output:` $\gamma = \alpha\beta$.
`Procedure:`

$\ell \leftarrow \ell_2(\alpha)$
if $(\ell \leq 2^{\nu_0})$
  return $\alpha * \beta$
fi
$\ell' \leftarrow \lceil \frac{\ell}{2^{\nu_0}} \rceil 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{2}$
represent $\alpha = \alpha_1 B + \alpha_0$ and $\beta = \beta_1 B + \beta_0$
$c_3 \leftarrow \texttt{Na}(\alpha_1, \beta_1)$
$c_2 \leftarrow \texttt{Na}(\alpha_0, \beta_1)$
$c_1 \leftarrow \texttt{Na}(\alpha_1, \beta_0)$
$c_0 \leftarrow \texttt{Na}(\alpha_0, \beta_0)$
return $\gamma = ((c_3 <<_B 2)|c0) + ((c_2 + c_1) <<_B 1)$

End of Algorithm 2.1

Four multiplications must be made, one addition of integers of size ℓ' and one addition of size $2\ell'$ as well. We could write this as 4 mult and 1 $\text{add}_{\mathcal{B}}(\ell')$ and 1 $\text{add}_{\mathcal{B}}(2\ell')$. Notice that one addition of integers of size ℓ' and one addition of size $2\ell'$ is equivalent to

$$\left\lceil \frac{\ell'}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 3 \left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil$$

additions of base words. Then we have that $\mathcal{MOB}(\mathtt{Na}, \ell)$ is

$$4^r \mathcal{MOB}(\mathtt{Na}, \frac{\ell}{2^r} + 2^{\nu_0} \sum_{i=0}^{r-1} \frac{c_{r-i}}{2^i}) + 3q \left(\frac{\ell}{2^{\nu_0}}(2^r - 1) + \sum_{i=0}^{r-1} 4^i \sum_{j=0}^{i} \frac{c_{i-j+1}}{2^j} \right),$$

where $0 \leq c_i < 1 \ \forall i$.

Result 2.1 *If $\ell = 2^{\nu_0+\nu}$, where $\nu \geq 0$ is an integer, then $c_i = 0 \ \forall i$ and*

$$\mathcal{MOB}(\mathtt{Na}, \ell) = 4^\nu + 3q(4^\nu - 2^\nu).$$

2.3.4 Karatsuba algorithm

The complexity of this algorithm is $O(N^{\log_2 3})$, where N is the length of the integers to be multiplied. We give a description of it in Algorithm 2.2.

Algorithm 2.2 ***Karatsuba:*** $\mathtt{Ka}(\alpha, \beta)$
`Input:` $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
`Output:` $\gamma = \alpha\beta$.
`Procedure:`

$\ell \leftarrow \ell_2(\alpha)$
if $(\ell \leq 2^{\nu_0})$
  return $\alpha * \beta$
fi
$\ell' \leftarrow \lceil \frac{\ell}{2^{\nu_0}} \rceil 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{2}$
represent $\alpha = \alpha_1 B + \alpha_0$ and $\beta = \beta_1 B + \beta_0$
$x \leftarrow \mathtt{Ka}(\alpha_1, \beta_1)$
$y \leftarrow \mathtt{Ka}(\alpha_0 - \alpha_1, \beta_1 - \beta_0)$
$z \leftarrow \mathtt{Ka}(\alpha_0, \beta_0)$
return $\gamma = ((x <<_B 2)|z) + ((x + y + z) <<_B 1)$

End of Algorithm 2.2

Three multiplications must be made, two additions of integers of size $\frac{\ell'}{2}$, two additions of integers of size ℓ' as well as one addition of size $2\ell'$. Note that two additions of integers of size $\frac{\ell'}{2}$, two additions of integers of size ℓ' and one addition of size $2\ell'$ are equivalent to

$$2\left\lceil \frac{\frac{\ell'}{2}}{2^{\nu_0}} \right\rceil + 2\left\lceil \frac{\ell'}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 5\left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil + \left(\left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil \mod 2 \right)$$

additions of base words. Therefore, we have that $\mathcal{MOB}(\mathtt{Ka}, \ell)$ is

$$3^r \mathcal{MOB}(\mathtt{Ka}, \frac{\ell}{2^r} + 2^{\nu_0} \sum_{i=0}^{r-1} \frac{c_{r-i}}{2^i}) + 10q \left(\frac{\ell}{2^{\nu_0}} \cdot \frac{3^r - 2^r}{2^r} + \sum_{i=0}^{r-1} 3^i \sum_{j=0}^{i} \frac{c_{i-j+1}}{2^j} \right) + C_r(\ell)q,$$

where $0 \leq c_i < 1 \ \forall i$ and $0 \leq C_r(\ell) \leq \frac{3^r-1}{2}$.

Result 2.2 *If $\ell = 2^{\nu_0+\nu}$, where $\nu \geq 0$ is an integer, then $c_i = 0 \ \forall i$, $C_r(\ell) = 0$ and*

$$\mathcal{MOB}(\mathtt{Ka}, \ell) = 3^\nu + 10q(3^\nu - 2^\nu).$$

2.3.5 Toom-Cook algorithm

The complexity of this algorithm is $O(N^{\log_3 5})$, where N is the length of the integers to be multiplied. We give its description in Algorithm 2.3.

Algorithm 2.3 ***Toom Cook:*** $\mathtt{T3}(\alpha, \beta)$
`Input:` $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
`Output:` $\gamma = \alpha\beta$.
`Procedure:`

$\ell \leftarrow \ell_2(\alpha)$
if $(\ell \leq 2^{\nu_0})$
  return $\alpha * \beta$
fi
$\ell' \leftarrow \lceil \frac{\ell}{3 \cdot 2^{\nu_0}} \rceil \, 3 \cdot 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{3}$

represent $\alpha = \alpha_2 B^2 + \alpha_1 B + \alpha_0$ and $\beta = \beta_2 B^2 + \beta_1 B + \beta_0$
$\gamma_4 \leftarrow \texttt{T3}(\alpha_2, \beta_2)$
$t_{\frac{1}{2}} \leftarrow \texttt{T3}(\alpha_2 + 2\alpha_1 + 4\alpha_0, \beta_2 + 2\beta_1 + 4\beta_0)$
$t_1 \leftarrow \texttt{T3}(\alpha_2 + \alpha_1 + \alpha_0, \beta_2 + \beta_1 + \beta_0)$
$t_2 \leftarrow \texttt{T3}(4\alpha_2 + 2\alpha_1 + \alpha_0, 4\beta_2 + 2\beta_1 + \beta_0)$
$\gamma_0 \leftarrow \texttt{T3}(\alpha_0, \beta_0)$
Solve a linear system.
$2\gamma_3 + 4\gamma_2 + 8\gamma_1 = t_{\frac{1}{2}} - 16\gamma_0 - \gamma_4$
$\gamma_3 + \gamma_2 + \gamma_1 = t_1 - \gamma_0 - \gamma_4$
$8\gamma_3 + 4\gamma_2 + 2\gamma_1 = t_2 - \gamma_0 - 16\gamma_4$
return $\gamma = ((((\gamma_4 <<_B 2)|\gamma_2) <<_B 2)|\gamma_0) + (((\gamma_3 <<_B 2)|\gamma_1) <<_B 1)$

End of Algorithm 2.3

In the Toom-Cook algorithm represented in Algorithm 2.3, five multiplications must be made, 12 additions of integers of size $\frac{\ell'}{3}$, 6 additions of integers of size $\frac{2\ell'}{3}$ and one addition of integers of size $2\ell'$ as well. If we define $c_1 = t_{\frac{1}{2}} - \gamma_4 - 16\gamma_0$, $c_2 = t_1 - \gamma_4 - \gamma_0$ and $c_3 = t_2 - 16\gamma_4 - \gamma_0$, then c_1 and c_3 must be even since all elements in the equation that must be solved are integers. Therefore, we can set $c'_1 = \frac{c_1}{2}$ and $c'_3 = \frac{c_3}{2}$ and then express the solution for the linear system as

$$\begin{aligned} 3\gamma_1 &= c_1 - 6c_2 + c'_3 \\ \gamma_2 &= -c'_1 + 5c_2 - c'_3 \\ 3\gamma_3 &= c'_1 - 6c_2 + c_3 \end{aligned}$$

Note that $5x = (x <<_2 2) + x$ and $6x = ((x <<_2 1) + x) <<_2 1$ and then we have 9 additions of numbers of size $\frac{2m}{3}$.

Therefore, the solution of the linear system requires 9 more additions of integers of size $\frac{\ell'}{3}$.

Note that 12 additions of integers of size $\frac{\ell'}{3}$, 15 additions of integers of size $\frac{2\ell'}{3}$ and one addition of integers of size $2\ell'$ is equivalent to

$$12\left\lceil \frac{\frac{\ell'}{3}}{2^{\nu_0}} \right\rceil + 15\left\lceil \frac{\frac{2\ell'}{3}}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 48\left\lceil \frac{\ell}{3 \cdot 2^{\nu_0}} \right\rceil$$

additions of base words. Thus we have that $\mathcal{MOB}(\mathtt{T3}, \ell)$ is

$$5^r \mathcal{MOB}(\mathtt{T3}, \frac{\ell}{3^r} + 2^{\nu_0} \sum_{i=0}^{r-1} \frac{c_{r-i}}{3^i}) + 48q \left(\frac{\ell}{2^{\nu_0+1}} \cdot \frac{5^r - 3^r}{3^r} + \sum_{i=0}^{r-1} 5^i \sum_{j=0}^{i} \frac{c_{i-j+1}}{3^j} \right)$$

where $0 \leq c_i < 1 \ \forall i$.

Result 2.3 *If $\ell = 2^{\nu_0+\nu}$, where $\nu \geq 0$ is an integer, then $\ell = 3^{\nu \log_3 2} 2^{\nu_0}$ and*

$$\mathcal{MOB}(\mathtt{T3}, \ell) = 5^{\nu \log_3 2} + 24q(5^{\nu \log_3 2} - 2^{\nu}) + C(\ell),$$

where $C(\ell) < 18q5^{\nu \log_3 2}$.

2.3.6 Schönhage algorithm

The complexity of this algorithm is $O(N \log N \log\log N)$. This algorithm takes advantage of the Fast Fourier Transform (FFT) whose complexity is $O(M \log M)$, where M is the number of elements to be transformed. The FFT is computed in the ring $R = \mathbb{Z}/(2^m + 1)\mathbb{Z}$, where m is a power of two and $\zeta_p = 2^q$ is a p-th primitive root of unity in R if $pq = 2m$.

The multiplication algorithm represented in Algorithm 2.4 on the next page is a different version of the one described by Zimmermann in [Zim92]. We have added to Zimmermann version the lines denoted by `+++` and modified the lines denoted by `***`. Because of the proposition 2.1, we use $\nu \geq 5$, which is not necessary an integer, and $\kappa = \lfloor \frac{\nu+1}{2} \rfloor$, while in Zimmermann's version there must be a certain previously fixed parameter κ.

Proposition 2.1 shows for which value of κ, the length of the integers to be multiplied in a step inside the algorithm is minimal.

Proposition 2.1 *Consider the function $n(\kappa) = \lceil 2^{\nu+1-2\kappa} + \frac{\kappa+3}{2^\kappa} \rceil 2^{\kappa+\nu_0}$ for real κ and fixed real numbers $\nu \geq 6$ and ν_0. If $4 \leq \kappa \leq \nu$ then $n(\kappa)$ has a minimum in $\kappa = \frac{\nu}{2} + 1$*

Proof

If $\frac{\nu}{2} + 1 \leq \kappa \leq \nu$, then $2^{\nu+1-2\kappa} \leq \frac{1}{2}$ and $\frac{\kappa+3}{2^\kappa} \leq \frac{1}{2}$. Thus $n(\kappa) = 2^{\kappa+\nu_0}$, which is an increasing function. Thus, $n(\kappa) \geq n(\frac{\nu}{2} + 1)$ for $\frac{\nu}{2} + 1 \leq \kappa \leq \nu$.

If $4 \leq \kappa \leq \frac{\nu}{2}+1$, then $2^{\nu-7} \geq 2^{\nu+1-2\kappa} \geq \frac{1}{2}$ and thus $n(\kappa) \geq 2^{\kappa+\nu_0} = n(\frac{\nu}{2}+1)$ for $4 \leq \kappa \leq \frac{\nu}{2}+1$.

□

Algorithm 2.4 ***Schönhage:*** $\mathtt{Sch}(\alpha, \beta)$

`Input:` $\alpha, \beta \in \mathbb{Z}/(2^m+1)\mathbb{Z}$, where $m = 2^{\nu_0+\nu}$.

`Output:` $\gamma = \alpha\beta \in \mathbb{Z}/(2^m+1)\mathbb{Z}$

`Procedure:`

Choose

+++ **if** $(\nu < 6)$

use an adequate multiplication algorithm as Karatsuba or Toom-Cook.

fi

Set

+++ $\kappa \leftarrow \lfloor \frac{\nu}{2} \rfloor + 1$

$M \leftarrow 2^\kappa$

*** $L \leftarrow \frac{m}{2^{\nu_0} M}$

*** $n \leftarrow \lceil \frac{\kappa+2L+3}{M} \rceil 2^{\nu_0} M$

$d \leftarrow \frac{n}{M}$

Represent

*** $\alpha = \sum_{j=0}^{M-1} \alpha_j \left(2^{2^{\nu_0} L}\right)^j$ and $\beta = \sum_{j=0}^{M-1} \beta_j \left(2^{2^{\nu_0} L}\right)^j$

Define

$f_\alpha(j) \leftarrow \alpha_j \zeta_{2M}^j \mod (2^n+1), f_\beta(j) \leftarrow \beta_j \zeta_{2M}^j \mod (2^n+1)$,

where $\zeta_{2M} = 2^d$ and $0 \leq j < M$.

Compute direct FFT

$FFT_M(f_\alpha, k) \mod (2^n+1)$ and $FFT_M(f_\beta, k) \mod (2^n+1) for 0 \leq k < M$,

where 2^{2d} is a M-th primitive root of unit.

Multiply

$H(k) \leftarrow \mathtt{Sch}(FFT_M(f_\alpha, k), FFT_M(f_\beta, k)) \mod (2^n+1)$ for $0 \leq k < M$.

Compute inverse FFT

$h(j) \leftarrow IFFT_M(H, j) \mod (2^n+1) for 0 \leq j < M$,

where 2^{2d} is a primitive root of unit.

Compute

$\gamma_j \leftarrow h(j)(2^d)^{-j} \mod (2^n+1)$.
*** if $(\gamma_j > (j+1)2^{2^{\nu_0+1}L})$
subtract 2^n+1 to it (considered as integers)
fi
Result
*** $\gamma = \sum_{j=0}^{M-1} \gamma_j \left(2^{2^{\nu_0}L}\right)^j \mod (2^m+1)$

End of Algorithm 2.4

If $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta) = 2^\nu$ and we want the product $\alpha\beta$ as integer (instead of mod $2^{\nu_0+\nu}$), the value $m = 2^{\nu_0+\nu+1}$ must be used. If $\nu \geq 6$, then in order to compute $\mathcal{MOB}(\mathtt{Sch}, 2^{\nu_0+\nu})$ we require 2^κ multiplications of integers of bit length $n(\kappa)$, two FFT of 2^κ elements of bit length $2^{\nu+\nu_0-\kappa}$ as well as one FFT of 2^κ elements of bit length $2^{\nu+1+\nu_0-\kappa}$, where $\kappa = \lfloor \frac{\nu}{2} \rfloor + 1$. That is to say,

Result 2.4 *If $\ell = 2^{\nu_0+\nu}$, $\nu \geq 6$ is an integer and $\kappa = \lfloor \frac{\nu}{2} \rfloor + 1$, then*

$$\mathcal{MOB}(\mathtt{Sch}, 2^{\nu_0+\nu}) = 2^\kappa \mathcal{MOB}(\mathtt{Sch}, n(\kappa)) + q\kappa 2^{\nu+2}.$$

2.3.7 Comparison amongst the multiplication algorithms

From the algorithms showed in Subsection 2.3.2 on page 75 we have the table 2.3.

Table 2.3: Comparison amongst the surveyed multiplication algorithms

Algorithm $\mathcal{A}$	$\mathcal{MOB}(\mathcal{A}, 2^{\nu+\nu_0})$
naïve (`Na`)	$2^{2\nu} + 3q(2^{2\nu} - 2^\nu)$.
Karatsuba (`Ka`)	$2^{\nu \log_2 3} + 10q\left(2^{\nu \log_2 3} - 2^\nu\right)$.
Toom Cook (`T3`)	$2^{\nu \log_3 5} + 24q\left(2^{\nu \log_3 5} - 2^\nu\right) + C$; $C < 18q2^{\nu \log_3 5}$.
Schönhage (`Sch`)	$\mathcal{MOB}(\mathtt{MA}, 2^{\nu+\nu_0})$ if $\nu < 6$, where `MA` is an adequate multiplication algorithm. Otherwise $2^\kappa \mathcal{MOB}(\mathtt{Sch}, 2^{\kappa+\nu_0}(2^{\nu+1-2\kappa}+1)) + q\kappa 2^{\nu+2}$.

As an example let us set $\nu_0 = 5$ and $q = 0.95$, we have the following comparison $\mathcal{MOB}(\texttt{Ka}, m) \leq \mathcal{MOB}(\texttt{Na}, m)$ for $2^{10} \leq m$ and $\mathcal{MOB}(\texttt{T3}, m) \leq \mathcal{MOB}(\texttt{Ka}, m)$ for $2^{15} \leq m$. If the Karatsuba multiplication algorithm is used in the Schönhage algorithm, then $\mathcal{MOB}(\texttt{Sch}, m) \leq \mathcal{MOB}(\texttt{T3}, m)$ for $m \geq 2^{17}$.

In figures 2.1 on the following page, 2.2 on page 85, 2.3 on page 86 and 2.4 on page 87 we show theoretical and practical behaviours of the surveyed algorithms for multiplying integers of length $2^{\nu_0+\nu}$. We have used the gmp library.

The gmp library makes uses of precomputed values of κ. In table 2.4 we compare the values chosen by gmp and our suggestions.

Table 2.4: Comparison of κ with gmp implementations

on a Solaris at 500MHz ($\nu_0 = 6$)						
gmp base word threshold	432	864	1856	3328	9216	20480
gmp κ	4	5	6	7	8	9
our suggested κ	5	5	6	6	7	8

on a Pentium 4 at 2.40 GHz ($\nu_0 = 5$)						
gmp base word threshold	624	1568	2688	7680	18432	40960
gmp κ	4	5	6	7	8	9
our suggested κ	5	6	6	7	8	8

on a Pentium III at 1.1 GHz ($\nu_0 = 5$)						
gmp base word threshold	592	1440	2688	5632	14336	40960
gmp κ	4	5	6	7	8	9
our suggested κ	5	6	6	7	7	8

2.4 Conclusion

We have shown that `Sch` needs no more multiplications of base words than `Ka`, or `T3` for integers whose bit length is greater than or equal to 2^{17} and we have shown

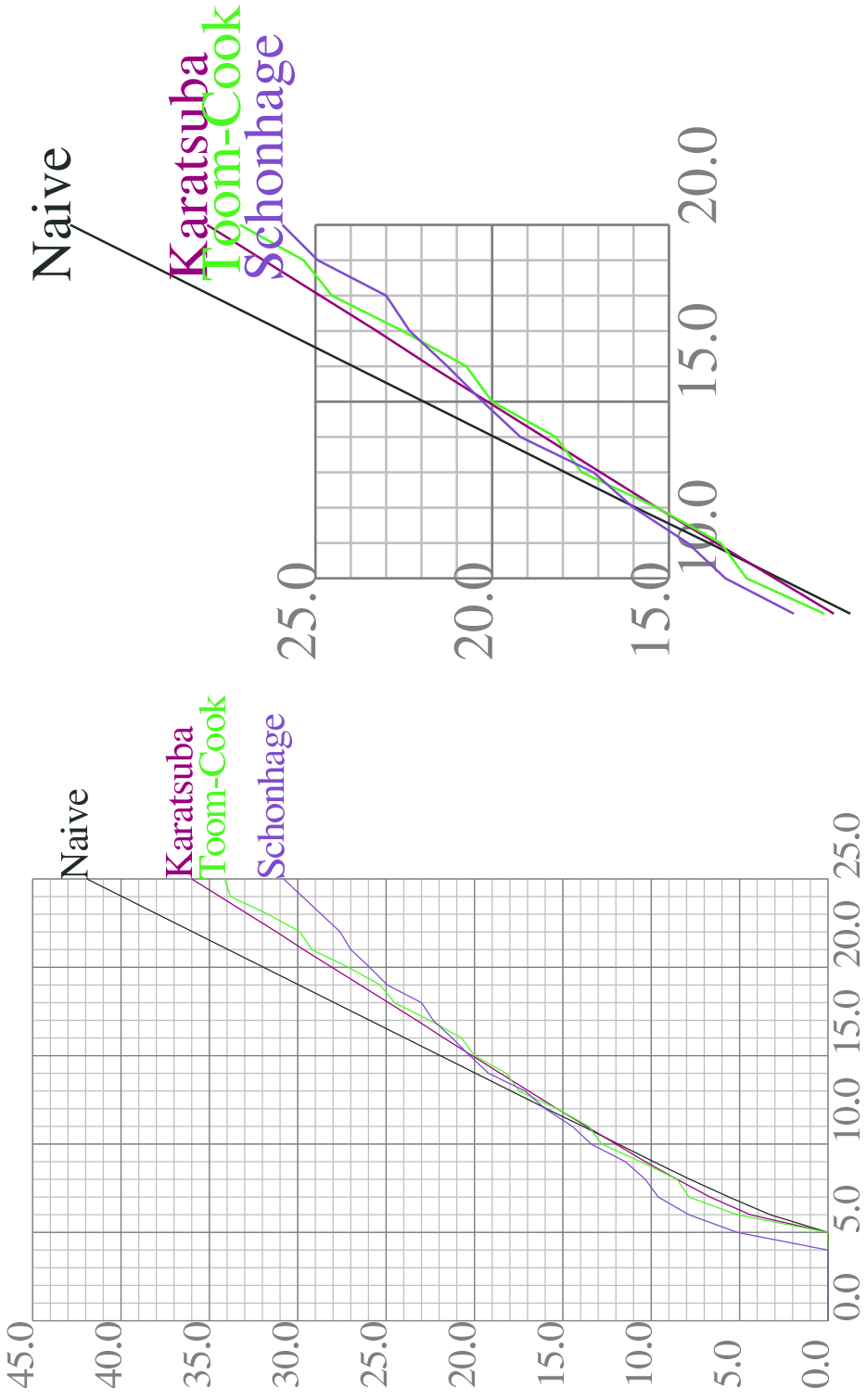


Figure 2.1: $\log_2(\mathcal{MOB}(alg, 2^{size}))$ with $q = 0.95$. The algorithms are: Naïve, Karatsuba, Toom Cook 3 and Schönhage

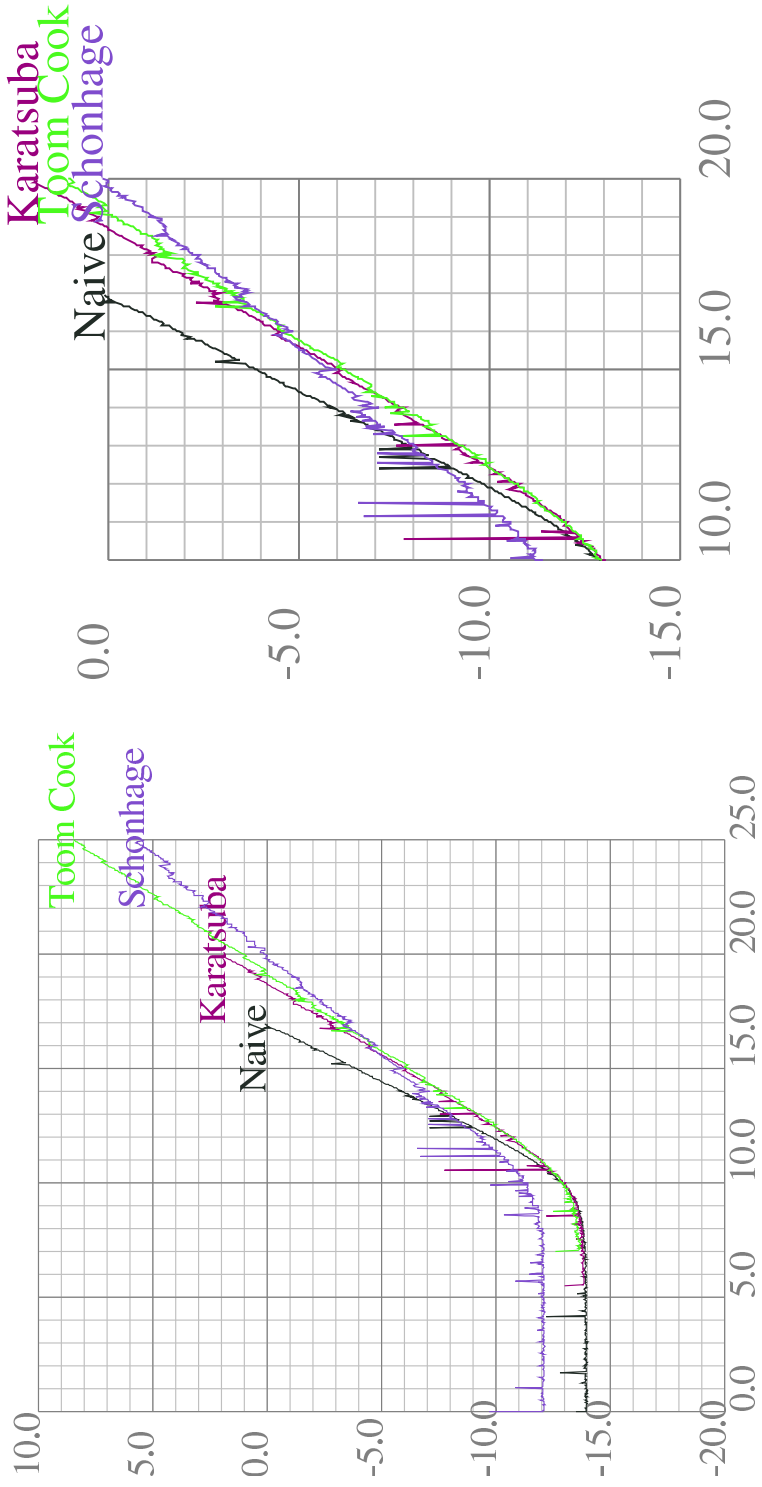


Figure 2.2: $\log_2(\text{timing}(alg, 2^{size}))$ Algorithm's implementation running on Solaris at 500MHz. The algorithms are: Naïve, Karatsuba, Toom Cook 3 and Schönhage

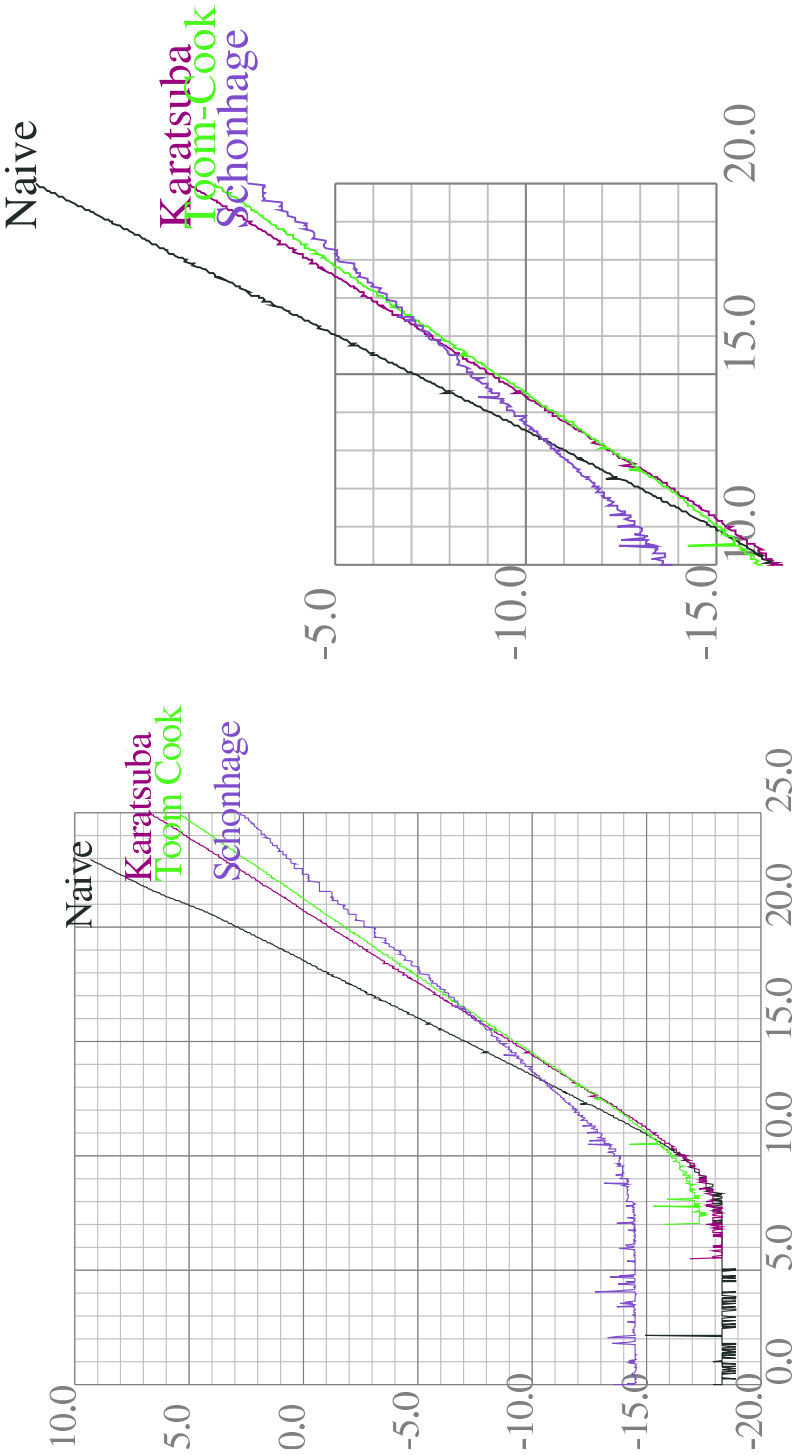


Figure 2.3: $\log_2(\text{timing}(alg, 2^{size}))$ Algorithm's implementation running on linux on Intel(R) Pentium(R) 4 at 2.40GHz. The algorithms are: Naïve, Karatsuba, Toom Cook 3 and Schönhage

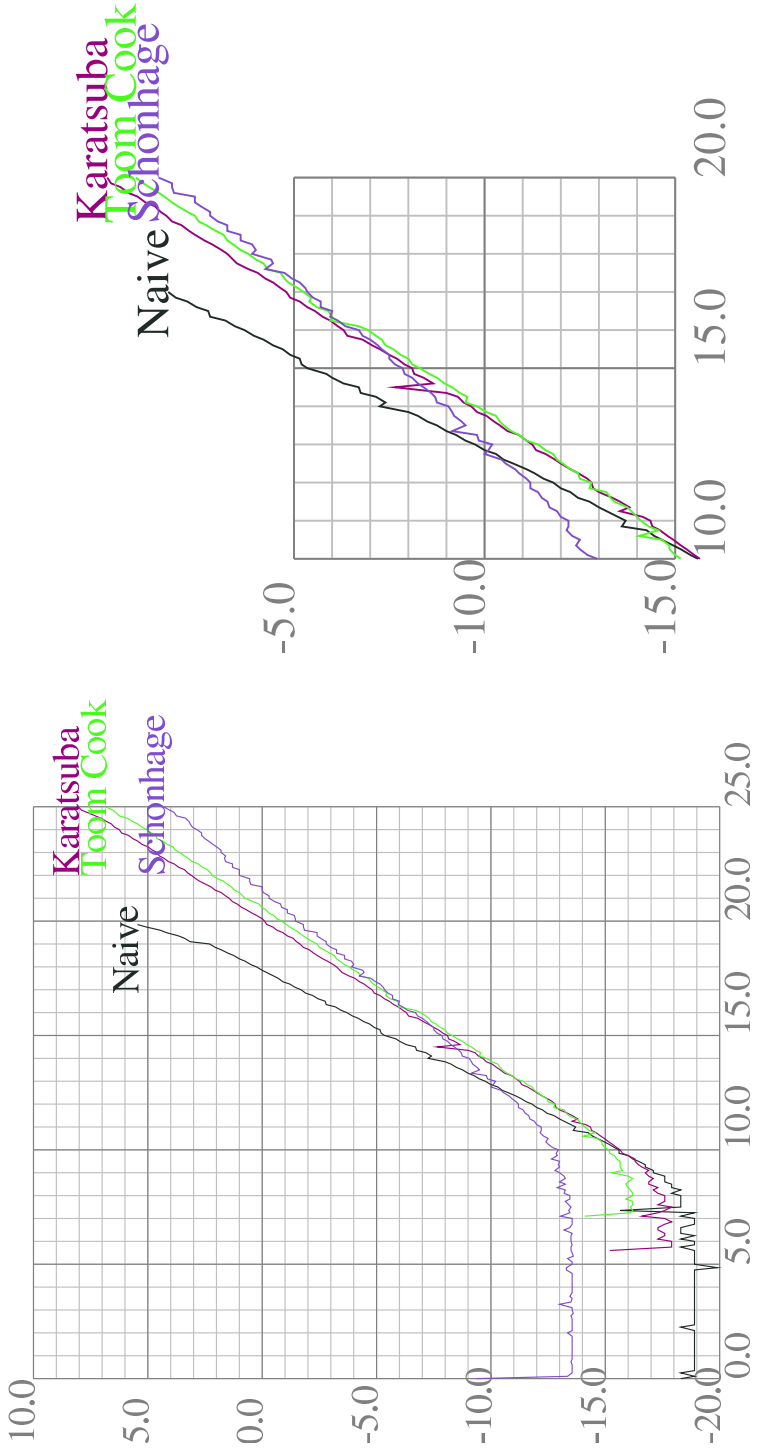


Figure 2.4: $\log_2(\text{timing}(alg, 2^{size}))$ Algorithm's implementation running on linux on Intel(R) Pentium(R) III at 1.10GHz. The algorithms are: Naïve, Karatsuba, Toom Cook 3 and Schönhage

a recurrent formula for computing the number of multiplications of base words for `Sch`. In figures 2.2, 2.3, and 2.4 is shown the graph of $y = \log_2(\mathcal{MOB}(alg, 2^{log\_size}))$, where alg is one of Naïve, Karatsuba, Toom Cook or Schönhage algorithms and $0 \leq log\_size \leq 25$.

With Schönhage algorithm less multiplication of base words are made than with Karatsuba or Toom Cook for integers whose bit length is greater than or equal 2^{17} and therefore for a modulus of such size. The modular multiplication [Mon85] and the modular exponentiation [Gor88] would be faster if Schönhage is used as multiplication algorithm, and so the RSA encryption or decryption as well. Actually, the employment of integer multiplication in modular exponentiation makes this result useful also in ElGamal encryption or signature scheme, since modular exponentiation in that scheme is required.

In Figures 2.1 on page 84, 2.2 on page 85, 2.3 on page 86, 2.4 on the preceding page the graph of $y = \log_2(\mathcal{MOB}(alg, 2^{log\_size}))$ is shown, where alg is one of the naïve, Karatsuba, Toom-Cook or Schönhage algorithms and $0 \leq log\_size \leq 25$.

Bibliography

[AK63] Yu. Ofman A. Karatsuba. Multiplication of multidigit numbers on automata. *Soviet Physics - Doklady*, 7(7):595–596, 1963.

[And02] Ross Anderson. Two remarks on public key cryptology. Technical Report 549, University of Cambridge, December 2002.

[Bea03] Stéphane Beauregard. Circuit for shor's algorithm using $2n+3$ qubits. *arXiv:quant-ph/0205095*, February 2003.

[BHT88] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In A. V. Moura C. L. Lucchesi, editor, *LATIN'98, LNCS*, volume 1380, pages 163–169. Springer-Verlag Berlin Heidelberg, 1988.

[BK04] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In Jan Camenisch Christian Cachin, editor, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 401–418. Springer-Verlag Heidelberg, 2004.

[BKN04] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for merkle tree traversal. In *Electronic Colloquium on Computational Complexity*, 2004.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - Crypto'99*, volume 1666, August 1999.

[BMS03] Dan Boneh, Ilya Mironov, and Victor Shoup. A secure signature scheme from bilinear maps. In *Topics in Cryptology – CT-RSA*, 2003.

[Bon99] Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *In Notices of the American Mathematical Society*, 1(2):203–213, 1999.

[BP99] Colin Boyd and Christopher J. Pavlovski. Efficient batch signature generation using tree structures. In *International Workshop on Cryptographic Techniques and E-Commerce*, pages 70–77. City University of Hong Kong Press, 1999.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: Aparadigm for designing efficient protocols. In *First ACM Conferences on computer and communications security*, 1993.

[Bre99] Richard P. Brent. Some parallel algorithms for integer factorisation. In *European Conference on Parallel Processing*, pages 1–22, 1999.

[Bre02] Richard P. Brent. Recent progress and prospects for integer factorisation algorithms. In D.-Z. Du et al., editor, *Computing and Combinatorics: 6th Annual International Conference, COCOON 2000*, volume 1858 of *LNCS*. Springer-Verlag Heidelberg, 2002.

[BTT03] Kemal Bicakci, Gene Tsudik, and Brian Tung. How to construct optimal one-time signatures. In *Computer Networks*, volume 43, pages 339–349, 2003.

[BY03] Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In M. Joye, editor, *Topics in Cryptology - CT-RSA'03*, Lectures Notes in Computer Science. Springer-Verlag, 2003.

[CDL+00] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter M. Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gerard Guillerm, Paul C. Leyland, Joel Marchand, Francois Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit RSA modulus.

In *Theory and Application of Cryptographic Techniques*, pages 1–18, 2000.

[Coh96] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer-Verlag, 1996.

[Cor02] Jean-Sébastien Coron. Security proof for partial-domain hash signature schemes. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 613–626. Springer-Verlag, 2002.

[CvDD+02] Dwaine Clarke, Marten van Dijk, Srinivas Devadas, Blaise Gassend, and G. Edward Suh. Caches and merkle trees for efficient memory authentication. Technical report, MIT, 2002.

[CWZ04] YuanDa Cao, Dong Wang, and LieHuang Zhu. Digital signature multicast stream secure against adaptive chosen message attack. In *Computers and Security*, volume 23, pages 229–240, 2004.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In *Fast Software Encryption*, volume 1039 of *LNCS*, pages 71–82. Springer-Verlag, 1996.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[EFMS04] Oscar Esparza, Jordi Forne, Jose L. Muñoz, and Miguel Soriano. Certificate revocation system implementation based on the merkle hash tree. *International Journal of Information Security*, 2(2):110–124, January 2004.

[EGM90] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 263–275. Springer-Verlag, 1990.

[FIPa] *SECURE HASH STANDARD*. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[FIPb] *SECURE HASH STANDARD (256, 384, 512).* http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman and company, 1979.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. In *Siam Journal on Computing*, pages 281–308, 1988.

[Gol87] Oded Goldreich. Two remarks concerning the goldwasser-micali-rivest signature scheme. In *Advances in Cryptology Lecture Notes in Computer Science*, volume 263, pages 104–110, 1987.

[Gor88] Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1988. Article No. AL970913.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 212–219, 1996.

[HILL] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computting.*

[Hug97] Richard J. Hughes. Cryptography, quantum computation and trapped ions. *http://arxiv.org/abs/quant-ph/9712054*, November 1997.

[ISA] ISAAC. Indirection, shift, accumulate, add, and count. http://www.burtleburtle.net/bob/rand/isaacafa.html.

[JLMS03] Markus Jakobsson, Tom Leighton, Silvio Micali, and Michael Szydlo. Fractal merkle tree representation and traversal. In *RSA Security Conference*, 2003. RSA Cryptographers Track.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1997.

[Kra00] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In *7th ACM Conference on Computer and Communications Security*, November 2000.

[LV01] A.K. Lenstra and E.R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[LW99] Simon S. Lam and Chung Kei Wong. Digital signature for flows and multicasts. In *ACM transactions on networking*, August 1999.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology - CRYPTO'87 LNCS*, volume 293, pages 369–378. Springer-Verlag Berlin Heidelberg 1988, 1988.

[Mer90] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology - CRYPTO'89 LNCS*, volume 435. Springer-Verlag Berlin Heidelberg 1990, 1990.

[Mic00] Silvio Micali. Cs proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of times periods. In *Eurocrypt 2002*, volume 2332. Springer-Verlag Berlin Heidelberg, 2002.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathemetics of Computation*, 44(170):519–521, April 1985.

[MvOV96] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Ope] OpenSSL. Openssl project. http://www.openssl.org/.

[Pol71] J. M. Pollard. The fast fourier transform in a finite field. *Mathematics on computation*, 25(114):365–374, 1971.

[RFCa] *The MD5 Message-Digest Algorithm.* http://www.faqs.org/rfcs/rfc1321.html.

[RFCb] *A 224-bit One-way Hash Function: SHA-224.* http://www.faqs.org/rfcs/rfc3874.html.

[RS04] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Lecture Notes in Computer Science*, volume 3017, pages 371 – 388. Springer-Verlag Heidelberg, 2004.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings, 35th Annual Symposium n Fundamentals of Comp. Science (FOCS)*, pages 124–134, 1994.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[SWV] A. Schönhage, A. F. W.Grotefeld, and E. Vetter. *Fast Algorithms.* Wissenschaftsverlag.

[Szy03] Michael Szydlo. Merkle tree traversal in log space and time. In *Eurocrypt 2004, LNCS*, volume 3027, pages 541–554, 2003.

[VSS+00] M. K. Vandersypen, M. Steffen, M. H. Sherwood, C. S. Yannoni, G. Breyta, and I. L. Chuang. Implementation of a three-quantum-bit search algorithm. *Applied Physics Letters*, 76(5), 2000.

[VSS+01] M. K. Vandersypen, M. Steffen, M. H. Sherwood, C. S. Yannoni, G. Breyta, and I. L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414:883–887, 2001.

[WY05] Xiaoyun Wang and Hongbo Yua. How to break md5 and other hash functions. In R. Cramer, editor, *EUROCRYPT 2005,*, volume 3494 of *LNCS*, pages 19–35. Springer-Verlag, 2005.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Collision search attacks on sha1. http://theory.csail.mit.edu/ yiqun/shanote.pdf, 2005.

[Zim92] P. Zimmermann. An implementation of Schönhage's multiplication algorithm. Technical report, Loria, 1992.

Curriculum Vitae

Education

1985 – 1989	High School "Centro de Estudios Científicos y tecnológicos 'Juan de Dios Bátiz' " (Scientific and technological study center 'Juan de Dios Bátiz')
1989 – 1994	"Escuela Supeior de Física y Matemáticas" (Institut for Physics and Mathematics) Degree of Bachelor of Science in Physics and Mathematics.
1994 – 1996	"Centro de investigación y de estudios avanzados" (Research and advanced studies center) Degree of Master of Science in Mathematicsc.
2002 – 2005	Technische Universität Darmstadt (Darmstadt University of Technology) PhD in Computer Science

Work expirience

1994 – 2001	Teacher of Mathematics at National Polytechnic Institut.
1997 – 2001	Employee by Banco de México.

Index