Practical Forward Secure Signatures using Minimal Security Assumptions

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Dipl.-Inform. Andreas Hülsing

geboren in Karlsruhe.

Referenten:	Prof. Dr. Johannes Buchmann
	Prof. Dr. Tanja Lange
Tag der Einreichung:	07. August 2013
Tag der mündlichen Prüfung:	23. September 2013
Hochschulkennziffer:	D 17

Darmstadt 2013

List of Publications

[1] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In A. Nitaj and D. Pointcheval, editors, *Africacrypt 2011*, volume 6737 of *Lecture Notes in Computer Science*, pages 363–378. Springer Berlin / Heidelberg, 2011. Cited on page 17.

[2] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, volume 7071 of *Lecture Notes in Computer Science*, pages 117–129. Springer Berlin / Heidelberg, 2011. Cited on pages 41, 73, and 81.

[3] Andreas Hülsing, Albrecht Petzoldt, Michael Schneider, and Sidi Mohamed El Yousfi Alaoui. Postquantum Signaturverfahren Heute. In Ulrich Waldmann, editor, *22. SIT-Smartcard Workshop 2012*, IHK Darmstadt, Feb 2012. Fraunhofer Verlag Stuttgart.

[4] Andreas Hülsing, Christoph Busold, and Johannes Buchmann. Forward secure signatures on smart cards. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin Heidelberg, 2013. Cited on pages 63, 73, and 81.

[5] Johannes Braun, Andreas Hülsing, Alex Wiesmaier, Martin A.G. Vigil, and Johannes Buchmann. How to avoid the breakdown of public key infrastructures. In Sabrina Capitani di Vimercati and Chris Mitchell, editors, *Public Key Infrastructures, Services and Applications*, volume 7868 of *Lecture Notes in Computer Science*, pages 53–68. Springer Berlin Heidelberg, 2013.

[6] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. *Journal of Applied Cryptography*, 3(1):84–96, 2013. Cited on page 17.

[7] Andreas Hülsing. W-OTS+ — shorter signatures for hash-based signature schemes. In A.Youssef, A. Nitaj, and A.E. Hassanien, editors, *Africacrypt 2013*, volume 7918 of *Lecture Notes in Computer Science*, pages 173–188. Springer Berlin / Heidelberg, 2013. Cited on pages 17 and 81.

[8] Maina M. Olembo, Timo Kilian, Simon Stockhardt, Andreas Hülsing, and Melanie Volkamer. Developing and testing a visual hash scheme. In Nathan Clarke, Steven Furnell, and Vasilis Katos, editors, *Proceedings of the European Information Security Multi-Conference (EISMC 2013)*. Plymouth University, April 2013.

[9] Patrick Weiden, Andreas Hülsing, Daniel Cabarcas, and Johannes Buchmann. Instantiating treeless signature schemes. Cryptology ePrint Archive, Report 2013/065, 2013. `http://eprint.iacr.org/`.

[10] Andreas Hülsing, Johannes Braun. Langzeitsichere Signaturen durch den Einsatz hashbasierter Signaturverfahren. In *Tagungsband zum 13. Deutschen IT-Sicherheitskongress 2013*, Herausgeber: BSI, Secu-Media Verlag, Gau-Algesheim, 2013.

[11] Johannes Braun, Moritz Horsch, Andreas Hülsing. Effiziente Umsetzung des Kettenmodells unter Verwendung vorwärtssicherer Signaturverfahren. In *Tagungsband zum 13. Deutschen IT-Sicherheitskongress 2013*, Herausgeber: BSI, Secu-Media Verlag, Gau-Algesheim, 2013.

[12] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for XMSS^{MT}. In Alfredo Cuzzocrea, Christian Kittl, DimitrisE. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, volume 8128 of *Lecture Notes in Computer Science*, pages 194–208. Springer Berlin Heidelberg, 2013. Cited on pages 63 and 73.

[13] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. To appear in proceedings of *Selected Areas in Cryptography 2013 (SAC'13).*

[14] Johannes Braun, Franziskus Kiefer, and Andreas Hülsing. Revocation & non-repudiation: When the first destroys the latter. To appear in proceedings of *EuroPKI 2013.*

Abstract

Digital signatures are one of the most important cryptographic primitives in practice. They are an enabling technology for eCommerce and eGovernment applications and they are used to distribute software updates over the Internet in a secure way. In this work we introduce two new digital signature schemes: XMSS and its extension XMSS^{MT}. We present security proofs for both schemes in the standard model, analyze their performance, and discuss parameter selection. Both our schemes have certain properties that make them favorable compared to today's signature schemes.

Our schemes are forward secure, meaning even in case of a key compromise, previously generated signatures can be trusted. This is an important property whenever a signature has to be verifiable in the mid- or long-term. Moreover, our signature schemes are generic constructions that can be instantiated using any hash function. Thereby, if a used hash function becomes insecure for some reason, we can simply replace it by a secure one to obtain a new secure instantiation. The properties we require the hash function to provide are minimal. This implies that as long as there exists any complexity-based cryptography, there exists a secure instantiation for our schemes. In addition, our schemes are secure against quantum computer aided attacks, as long as the used hash functions are.

We analyze the performance of our schemes from a theoretical and a practical point of view. On the one hand, we show that given an efficient hash function, we can obtain an efficient instantiation for our schemes. On the other hand, we provide experimental data that show that the performance of our schemes is comparable to that of today's signature schemes. Besides, we show how to select optimal parameters for a given use case that provably reach a given level of security.

On the way of constructing XMSS and XMSS^{MT}, we introduce two new one-time signature schemes (OTS): W-OTS^{+} and W-OTS$^{\$}$. One-time signature schemes are signature schemes where a key pair may only be used once. W-OTS^{+} is currently the most efficient hash-based OTS and W-OTS$^{\$}$ the most efficient hash-based OTS with minimal security assumptions. One-time signature schemes have many more applications besides constructing full fledged signature schemes, including authentication in sensor networks and the construction of chosen-ciphertext secure encryption schemes. Hence, W-OTS$^+$ and W-OTS$^\$$ are contributions on their own.

Altogether, this work shows the practicality and usability of forward secure signatures on the one hand and hash-based signatures on the other hand.

Zusammenfassung

Digitale Signaturen sind eines der meist genutzten kryptographischen Primitive in der Praxis. Sie stellen eine notwendige Technologie für eCommerce und eGovernment Anwendungen dar und werden benötigt, um Softwareupdates auf eine sichere Weise über das Internet zu verteilen. In dieser Arbeit werden zwei neue Verfahren zur digitalen Signatur vorgestellt: XMSS und dessen Erweiterung XMSS^{MT}. Für beide Verfahren werden Sicherheitsbeweise im Standardmodell gegeben, die Performanz analysiert und die Wahl sicherer Parameter diskutiert. Beide Verfahren haben bestimmte Eigenschaften auf Grund derer sie den heute verwendeten Verfahren zur Digitalen Signatur vorzuziehen sind.

Die vorgestellten Verfahren sind vorwärtssicher. Dies bedeutet, dass selbst im Falle einer Kompromittierung des geheimen Schlüssels, zuvor erzeugten Signaturen weiterhin vertraut werden kann. Dies ist eine wichtige Eigenschaft, die benötigt wird wenn eine Signatur über einen längeren Zeitraum verifizierbar sein muss. Darüber hinaus handelt es sich bei den vorgestellten Verfahren um generische Konstruktionen, die mit einer beliebigen kryptographischen Hashfunktion instanziiert werden können. Sollte die verwendete Hashfunktion aus irgendeinem Grund unsicher werden, reicht es aus die Hashfunktion durch eine neue, sichere Hashfunktion zu ersetzen, um eine sichere Instanziierung der Verfahren zu erhalten. Die Eigenschaften, welche wir von der verwendeten Hashfunktion fordern sind minimal. Daraus folgt, dass es eine sichere Instanziierung der vorgestellten Verfahren gibt, solange es überhaupt Komplexitäts-basierte Kryptographie gibt. Darüber hinaus bieten beide Verfahren Schutz vor Quantencomputer-basierten Angriffen, solange die verwendete Hashfunktion nicht anfällig für solche Angriffe ist.

In der vorliegenden Arbeit wird die Performanz der vorgestellten Verfahren sowohl theoretisch, als auch praktisch evaluiert. Einerseits wird gezeigt, dass eine beliebige effiziente kryptographische Hashfunktion eine effiziente Instanziierung der Verfahren ermöglicht. Andererseits werden experimentelle Ergebnisse geliefert. Diese belegen, dass die Performanz der vorgestellten Verfahren vergleichbar zu jener heute in der Praxis genutzter Verfahren ist. Weiterhin wird eine Methode vorgestellt, die es für beide Verfahren erlaubt, optimale Parameter für einen vorgegebenen Anwendungsfall zu wählen. Die erzeugten Parameter erreichen dabei beweisbar ein gegebenes Sicherheitsniveau.

Als Teil der Konstruktion von XMSS und XMSS^{MT}, werden zwei neue Einmalsignaturverfahren (One-Time Signature Schemes, OTS) vorgestellt: W-OTS^{+} und W-OTS$^{\$}$. Einmalsignaturverfahren sind Signaturverfahren, die es erlauben pro Schlüsselpaar genau eine Nachricht zu signieren. W-OTS^{+} ist aktuell das effizienteste Hash-basierte OTS. W-OTS$^{\$}$ ist das effizienteste OTS mit minimalen Sicherheitsannahmen. Neben der Konstruktion vollwertiger Signaturverfahren, finden OTS Verwendung in Anwendungsfällen wie der Authentifizierung in Sensornetzen oder der Konstruktion chosen-ciphertext sicherer Public-Key Verschlüsselungsverfahren. Daher stellt die Konstruktion von W-OTS^{+} und W-OTS$^{\$}$ einen selbständigen Beitrag dar.

Zusammenfassend belegt diese Arbeit die Praktikabilität und Nutzbarkeit vorwärtssicherer Signaturen einerseits und Hash-basierter Signaturen andererseits.

Contents

Abstract **v**

1 Introduction **1**

1.1 Contribution and Organization 3

2 Background **7**

2.1 Notation & Formal Definitions 7

2.1.1 (Hash) Function Families 7

2.1.2 Digital Signature Schemes 10

2.2 Hash-based Signature Schemes 11

2.2.1 The Lamport-Diffie One-Time Signature Scheme 12

2.2.2 The Merkle Signature Scheme 12

2.2.3 Tree Traversal Algorithms 14

3 New Variants of the Winternitz One Time Signature Scheme **17**

3.1 The Winternitz One-Time Signature Scheme 17

3.1.1 Function Chains 18

3.1.2 W-OTS 20

3.2 The Security of Generic W-OTS 22

3.3 A Function Chain using Second-Preimage Resistance 25

3.3.1 The $\mathcal{C}^+$ Function Chain 26

3.3.2 Preliminaries 26

3.3.3 Security Proof 27

3.4 A Function Chain using Pseudorandom Function Families 34

3.4.1 The $\mathcal{C}^\$$ Function Chain 35

3.4.2 Preliminaries 35

3.4.3 Security Proof 37

4 XMSS 41

4.1 The eXtended Merkle Signature Scheme XMSS 41

4.2 Standard Security 45

4.2.1 Preliminaries 45

4.2.2 XMSS is Existentially Unforgeable under Chosen Message Attacks 48

4.3 Forward Security 53

4.3.1 Preliminaries 53

4.3.2 XMSS is Forward Secure 56

4.4 Theoretical Performance 60

5 XMSS^{MT}– XMSS with Virtually Unlimited Signature Capacity 63

5.1 Multi Tree XMSS - XMSS^{MT} 63

5.2 Security 67

5.3 Analysis 68

5.3.1 Correctness 69

5.3.2 Theoretical Performance 70

6 Choosing Optimal Parameters for XMSS* 73

6.1 Security Level of XMSS* 73

6.1.1 Security Level of XMSS using W-OTS$^\$$ 74

6.1.2 Security Level of XMSS using W-OTS$^+$ 75

6.1.3 Security Level of XMSS^{MT} 75

6.2 Optimization 76

6.3 Results 78

7 XMSS* in Practice 81

7.1 Implementing the Function Families 81

7.2 C Implementation 83

7.3 Smart Card Implementation 86

8 Conclusion 91

1 Introduction

Digital signature schemes are among the most used cryptographic primitives in practice. They are means to guarantee for authenticity and non-repudiation of any kind of data like messages, documents, or software. Digital signature schemes are used to secure communication protocols, including SSL / TLS and SSH, to protect software updates, and in many countries they have become legally binding like a handwritten signature. Thereby, digital signatures enable applications like online banking, e-Commerce and e-Government as well as online distribution of software updates, secure communication, and counting. Today, there exist three signature schemes used in practice: RSA, DSA and EC-DSA. There are some issues these schemes have in common and we want to highlight them:

- When it comes to non-repudiation, additional measures from the field of public key infrastructures, like time stamping, have to be applied to guarantee that a validly generated signature remains valid for more than a short time period.

- The security of all three schemes is based on the hardness of certain problems from number theory. These problems become easy if large enough quantum computers can be built [Sho94].

- Only certain variants of these schemes can be proven secure and these proofs are only heuristic, i.e. the proofs are either in the random oracle or the generic group model. This means that there is no guarantee that it is really necessary to solve the underlying hard problem to break the signature scheme.

- The schemes either rely on a specific hardness assumption or they make use of a so called trapdoor one-way function. The latter is a notion that is hard to achieve and unnecessarily strong from a complexity theoretic point of view.

In this work we introduce XMSS and its extension XMSS^{MT}, two forward secure signature schemes based on minimal security assumptions. We show that these schemes solve all of the above issues without significant performance losses and thereby constitute an interesting alternative to today's signature schemes.

Our schemes fulfill the notion of forward secure signature schemes (FSS). The idea behind FSS is the following: Even in the case of a key compromise, all signatures issued before the compromise should remain valid. This is an important property for all use cases where signatures have to stay valid for more than a short time period, including use cases like document signing or certificates. If for example a contract is signed, it is important that the signature stays valid for at least as long as the contract has some relevance. The solutions used today require the use of time stamps [ETS10, ETS12]. This introduces the requirement for a trusted third party and the overhead of requesting a time stamp for each signature. FSS in turn already provide this property and thereby abandon the need for time stamps. Moreover, we show in [BHW$^+$13] that FSS can also be used to strengthen the security of actual public key infrastructures.

Our schemes are hash-based signature schemes. Thereby, they can be instantiated using any cryptographic hash function (as it turns out, we can also use most block ciphers to build the required function families). This means, we simply have to replace the used hash function and get a new signature scheme in case one instantiation based on a certain computational problem turns out to be insecure. This is in contrast to today's signature schemes that can not be repaired if the underlying computational problem becomes easy. Moreover, quantum computers do not threaten the security of hash functions in general [Gro96, AS04]. As there exist many hash functions relying on problems that are assumed to be quantum secure, XMSS and XMSS^{MT} will still have secure instantiations in the presence of quantum computers.

We give security proofs for XMSS and XMSS^{MT} in the standard model. Thus, in contrast to today's signature schemes, we show that attacking one of these schemes is as hard as solving the underlying problem. In our case this means we prove that breaking the forward or standard security of our schemes is almost as hard as breaking certain properties of the used hash function. In our proofs we make no use of any heuristic argument like the random oracle model. Furthermore, all our proofs are formulated as tight, exact reductions. This means, we exactly compute the relation between the hardness of breaking the security of our schemes and breaking the security properties of the used hash function and this relation is tight. This allows us to make strong statements about the security of parameters.

Our proofs show that XMSS and XMSS^{MT} are forward secure – which implies standard security – if the schemes are instantiated using a second preimage resistant hash function and a pseudorandom function family. Rompel [Rom90] - building upon results by Naor and Yung [NY89] - showed that the necessary and sufficient condition for the existence of a standard secure digital signature scheme is the existence of a one-way function. This in turn is the minimal condition for the existence of complexity-based cryptography. As pseudorandom function families and second-preimage resistant hash functions can be constructed from any one-way function [Rom90, HILL99, GGM86] our assumptions are minimal. This means, there exists a secure instantiation of our schemes, as long as there exists any secure digital signature scheme. RSA requires the existence of trapdoor one-way function families, which is a strictly stronger assumption than the existence of a one-way function. Moreover, all of today's schemes use specific properties of the underlying hardness assumption and can therefore not be constructed from any one-way function.

Rompel's result combined with that of Naor and Yung already gives a construction of a digital signature scheme from any one-way function. However, their construction is only of theoretical interest as the performance of this scheme is completely impractical. In contrast, we show that XMSS and XMSS^{MT} are indeed practical. On the one hand, we present a theoretical analysis that shows that both schemes are almost as efficient as the used hash function. Hence, given a practical hash function both schemes are practical. On the other hand, we provide experimental results. We present two implementations, one for regular CPUs and one for smart cards. Experiments show that using today's hash functions, both schemes can beat some of today's schemes in terms of runtimes. Key and signature sizes, while slightly larger, are also practical — even on resource constrained devices like smart cards.

1.1 Contribution and Organization

The main contribution of this work is the development and comprehensive analysis of XMSS and its extension XMSS^{MT}. This contribution can be separated into several technical contributions we detail now.

Background (Chapter 2) We begin with some background in Chapter 2. There we present definitions of the basic security notions for hash functions and digital signature schemes that we use in more than one chapter. Notions that are only used in a single chapter are introduced in the respective chapter. We also introduce the basic concepts of hash-based signature schemes that we use later.

New One-Time Signatures (Chapter 3) The basic building block of a hash-based signature scheme is a one-time signature scheme (OTS). This is a signature scheme where a key pair can be used to sign only one arbitrary message. In Chapter 3 we introduce two new one-time signature schemes called W-OTS$^+$ and W-OTS$^\$$.

Both schemes are based on the general idea of the Winternitz OTS (W-OTS) first mentioned in [Mer90a]. W-OTS turned out to be the best choice of an OTS for hash-based signature schemes for several reasons (see Chapter 3). Our constructions outperform all previous variants of W-OTS in the sense that we require weaker security assumptions to proof the schemes secure. This allows to reduce the signature size while obtaining the same level of security as previous constructions.

More detailed, the only provably secure W-OTS construction known before required a collision resistant hash function family [HM02, DSS05]. We require either a second-preimage resistant, undetectable one-way function family or a pseudorandom function family for W-OTS$^+$ and W-OTS$^\$$, respectively. These security assumptions are strictly weaker than collision resistance. Moreover, the assumption that a pseudorandom function family exists is minimal. Thereby W-OTS$^\$$ has minimal security assumptions. We also assume that this is the case for W-OTS$^+$, but we can not prove it (see the discussion in Section 3.3). However, for W-OTS$^+$ we achieve stronger security and a tighter security proof. The latter allows for more efficient parameter choices.

XMSS (Chapter 4) Given our OTS constructions, we develop our first forward secure signature scheme XMSS in Chapter 4. It is the first efficient forward secure signature scheme (FSS) with minimal security assumptions. Previous efficient FSS are based on specific number theoretic assumptions, i.e. [AMN01, AR00, BM99, CK06, IR01, KR03, Son01]. In addition, there are two generic constructions by Krawczyk [Kra00] and Malkin et al. [MMM02]. They allow building forward secure signature schemes from any secure digital signature scheme. However, their security assumptions are not minimal and the schemes do not appear to be practical because of their key size. We note that some of the techniques used for XMSS can be applied to the construction from [MMM02] to solve these issues.

On the other hand, there are three other signature schemes with minimal security assumptions in the above sense: [Gol09, Rom90, DOTV08]. They are not forward secure. Moreover, the schemes in [Gol09] and [Rom90] are not practical and the security proof of MSS-SPR [DOTV08] does not cover the pseudorandom key generation which is necessary for the scheme being efficient. In addition, compared to MSS-SPR, XMSS reduces the signature size by more than 25 % at the same level of security. This is an important improvement as the signature size is considered the main drawback of hash-based signatures. However, XMSS uses the new tree construction introduced in [DOTV08].

XMSS^{MT} (Chapter 5) Using XMSS the number of signatures that can be generated using one key pair is limited. The reason is that the runtime of the key generation algorithm is linear in the number of signatures p. Hence, the key generation time constitutes a bound on p in practice. Multi-Tree XMSS (XMSS^{MT}) is an extension of XMSS, presented in Chapter 5. In contrast to XMSS it allows for a virtually unlimited number of signatures. Therefor, XMSS^{MT} allows a trade-off between key generation time and signature size. The key generation time is reduced from $\mathcal{O}(p)$ to $\mathcal{O}(p/d)$ for any integer parameter $0 < d < p$ while the signature size grows linearly in d. Towards this end, we apply the tree chaining concept introduced in [BGD$^+$06] and improve the idea of distributed signature generation [BDK$^+$07]. Besides virtually unlimited signatures, XMSS^{MT} also enables implementations on resource constrained devices like smart cards with practical key generation times. Previous smart card implementations of hash-based signature schemes did not achieve on-card key generation at all [RED$^+$08].

Optimal Parameters (Chapter 6) The performance of XMSS and XMSS^{MT} are controlled by many parameters, defining various trade-offs. We show how linear optimization can be used to select optimal parameters for both schemes in Chapter 6. This was done before in [BDK$^+$07]. Unfortunately, the results given there do not carry over to XMSS and XMSS^{MT} as fewer parameters were taken into account and the authors of [BDK$^+$07] do not provide details about how to model the problem. In this context we also show how to use the exact security reductions to compute the security level of parameter sets in the sense of [Len04]. Our optimization chooses optimal parameters for a given level of security.

Experimental Evaluation (Chapter 7) We evaluate the practical performance of our schemes in Chapter 7. We present two implementations. A XMSS implementation for traditional CPUs and a XMSS^{MT} implementation for smart cards. In this context, we show how to implement the used function families for XMSS and XMSS^{MT} in practice using a hash-function or a block cipher. Our experimental results show that the performance of both schemes is comparable to that of RSA, DSA and ECDSA. So far, there exists no other smart card implementation of a forward secure signature scheme. We are only aware of a study on the performance of FSS on traditional CPUs [CJMM03]. For hash-based signature schemes, there already exist a lot of implementations. However, the only known smart card implementation did not achieve on-card key generation [RED$^+$08].

Conclusion (Chapter 8) Finally, we draw a conclusion in Chapter 8 and discuss possible future work.

2 Background

In this chapter we present some background for the work at hand. First, we fix notation and give some formal definitions used through out the whole paper. We only present the most important formal definitions for hash functions and signature schemes. Definitions that are only used in one chapter are given there. Afterwards we present some background on hash based signature schemes.

2.1 Notation & Formal Definitions

In this section we first fix some notation. Afterwards we present the formal definitions of the most important hash function properties and digital signature schemes. Through out this thesis, we write $x \xleftarrow{\$} \mathcal{X}$ if x is randomly chosen from the set $\mathcal{X}$ using the uniform distribution. We further write log for $\log_2$. We denote the uniform distribution over bit strings of length n by $\mathcal{U}_n$. We write $m = \mathrm{poly}(n)$ to denote that m is a function, polynomial in n. We call a function $\epsilon(n) : \mathbb{N} \rightarrow [0,1]$ negligible and write $\epsilon(n) = \mathrm{negl}(n)$ if for any $c \in \mathbb{N}, c > 0$ there exists a $n_c \in \mathbb{N}$ s.th. $\epsilon(n) < n^{-c}$ for all $n > n_c$. In all our proofs, we count runtimes in terms of evaluations of a function family.

2.1.1 (Hash) Function Families

We now give formal definitions for the four most important properties of a (hash) function family, namely one-wayness, second-preimage resistance, collision resistance, and pseudorandomness. For the definitions we follow Rogaway and Shrimpton [RS04]. We will also shortly discuss some relations between these notions. We restrict ourselves to function families that operate on bit strings and have a fixed input size, as this is the case in our constructions. However, the definitions are the same for the more general case. In the following let $n \in \mathbb{N}$ be the security parameter, $m, k = \mathrm{poly}(n)$, $\mathcal{H}_n = \{\mathrm{H}_K : \{0,1\}^m \rightarrow \{0,1\}^n \,|\, K \in \{0,1\}^k\}$ a family of functions. We say a function family $\mathcal{H}_n$ is efficient if there exists a probabilistic polynomial time (PPT) algorithm that evaluates $\mathrm{H}_K(M)$ for any $M \in \{0,1\}^m$ and $K \in \{0,1\}^k$. We require all used function families to be efficient, unless we state otherwise. We call $\mathcal{H}_n$ a hash function family if $\mathcal{H}_n$ is efficient and $m > n$, i.e. $\mathcal{H}_n$ is compressing.

We start defining the success probability of an adversary $\mathcal{A}$ against the one-wayness (OW) of a function family $\mathcal{H}_n$. Informally, $\mathcal{A}$ receives a random output of a function from the family and has to find a valid preimage under this function, i.e. invert the function. Formally this reads as

$$\begin{aligned}\mathrm{Succ}^{\mathrm{OW}}_{\mathcal{H}_n}(\mathcal{A}) = \mathsf{Pr}\,[\,& K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m, Y \longleftarrow \mathrm{H}_K(M),\\ & M' \xleftarrow{\$} \mathcal{A}(K,Y) : Y = \mathrm{H}_K(M')]\ .\end{aligned} \tag{2.1}$$

We next define the success probability of an adversary $\mathcal{A}$ against second-preimage resistance (SPR). Informally, the adversary receives a random element of the function family and a random element of its domain. The goal of $\mathcal{A}$ is to come up with a collision for the given domain value and function. More formally:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{SPR}}_{\mathcal{H}_n}(\mathcal{A}) =& \mathsf{Pr}\,[\,K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m, M' \longleftarrow \mathcal{A}(K,M) :\\ & (M \neq M') \wedge (\mathrm{H}_K(M) = \mathrm{H}_K(M'))]\ .\end{aligned} \tag{2.2}$$

The last classical property of a hash function is collision resistance (COLL). Here the adversary has to come up with two elements of the domain that collide under a given element of the function family:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{COLL}}_{\mathcal{H}_n}(\mathcal{A}) =& \mathsf{Pr}\,[\,K \xleftarrow{\$} \{0,1\}^k; (M, M') \longleftarrow \mathcal{A}(K) :\\ & (M \neq M') \wedge (\mathrm{H}_K(M) = \mathrm{H}_K(M'))]\ .\end{aligned} \tag{2.3}$$

We say that a function family has one of these properties if it is efficient and there exists no PPT adversary that has a non-negligible success probability. As an example we give the full definition for a one-way function family. The definitions for second-preimage resistance and collision resistance are obtained accordingly and hence we omit them.

Definition 2.1 (OW). *Let $\mathcal{H}_n$ be defined as above. We call $\mathcal{H}_n$ a one-way function family if $\mathcal{H}_n$ is efficient and if for any $t = \mathrm{poly}(n)$ the maximum success probability $\mathrm{InSec}^{\mathrm{OW}}(\mathcal{H}_n; t)$ of all possibly probabilistic adversaries $\mathcal{A}$ running in time $\leq t$ is negligible in n:*

$$\mathrm{InSec}^{\mathrm{OW}}(\mathcal{H}_n; t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{OW}}_{\mathcal{H}_n}(\mathcal{A})\} = \mathrm{negl}(n)\,.$$

The fourth notion we use is pseudorandomness of a function family (PRF). In the definition of the success probability of an adversary against pseudorandomness the adversary gets black-box access to an oracle Box. Box is either initialized with a function from $\mathcal{H}_n$ or a function from the set $\mathcal{AF}(m,n)$ of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$. The goal of the adversary is to distinguish both cases:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{PRF}}_{\mathcal{H}_n}(\mathcal{A}) = \Big| & \mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{H}_n : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1] \\ & -\mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{AF}(m,n) : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1]\Big| \,. \end{aligned} \tag{2.4}$$

Using this success probability, we define a pseudorandom function family the following way.

Definition 2.2 (PRF)**.** *Let $\mathcal{H}_n$ be defined as above. We call $\mathcal{H}_n$ a pseudorandom function family, if it is efficient and for all $q, t = \mathrm{poly}(n)$ the maximum success probability $\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{H}_n; t, q)$ of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, making at most q queries to Box, is negligible in n:*

$$\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{H}_n; t, q) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{PRF}}_{\mathcal{H}_n}(\mathcal{A})\} = \mathrm{negl}(n)\,.$$

These notions are related. On the one hand there exist direct implications. It is known that COLL implies SPR but the inverse does not hold. If for a non-negligible set of elements H of $\mathcal{H}_n$ at least a non-negligible set of elements from the domain collide with at least one other element under H, then $\mathcal{H}_n$ being SPR implies $\mathcal{H}_n$ is also OW. This is the case because if we evaluate a random H on a random element x of the domain and hand the result to a successful inverter $\mathcal{A}$, $\mathcal{A}$ will return x with probability at most $1/2$ if x collides with at least one other value. Thus, with probability $\geq 1/2$ $\mathcal{A}$ will return a different preimage. For a more detailed discussion of hash function notions and their relations see [RS04].

On the other hand, there exist complexity theoretic relations. It is known that a second-preimage resistant hash-function family can be constructed given any one-way function [Rom90]. As the existence of a one-way function is the minimal assumption needed for the existence of complexity theoretic cryptography, assuming the existence of a second-preimage resistant hash function family means making minimal security assumptions. It is also known that the existence of a one-way function implies the existence of a pseudorandom function family. This follows from the result of Håstad et al. [HILL99] who show how to construct a pseudorandom generator from any one-way function and the GGM construction [GGM86] of a pseudorandom function family from any pseudorandom generator. It is noteworthy that current research suggests that it is not possible to construct a collision resistant hash function family from any one-way function. Hence, the existence of a collision resistant hash function family is a strictly stronger assumption than those mentioned before.

2.1.2 Digital Signature Schemes

In the following we present the definitions for digital signature schemes that we use. Regarding security we recall the standard notion of existential unforgeability under adaptive chosen message attacks (EU-CMA) introduced in [GMR88]. While this notion suffices for most applications, sometimes a stronger notion is needed called strong unforgeability under adaptive chosen message attacks (SU-CMA). SU-CMA secure signature schemes have a number of applications, including the construction of chosen-ciphertext secure encryption schemes [CHK04] and group signatures [ACJT00, BBS04]. We start with the definition of digital signature schemes.

Digital Signature Scheme

Let $\mathcal{M}$ be the message space. A digital signature scheme $\textsc{Dss} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ is a triple of probabilistic polynomial time algorithms:

- $\mathsf{Kg}(1^n)$ on input of a security parameter 1^n outputs a private signing key sk and a public verification key pk;
- $\mathsf{Sign}(\mathsf{sk}, M)$ outputs a signature σ under sk for message M, if $M \in \mathcal{M}$;
- $\mathsf{Vf}(\mathsf{pk}, \sigma, M)$ outputs 1 iff σ is a valid signature on M under pk;

such that the following correctness condition is fulfilled:

$$\forall(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{Kg}(1^n), \forall(M \in \mathcal{M}) : \mathsf{Vf}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, M), M) = 1.$$

Throughout this work *signature scheme* always refers to a digital signature scheme.

Existential Unforgeability under Adaptive Chosen Message Attacks

The standard security notion for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) which is defined using the following experiment. By $\textsc{Dss}(1^n)$ we denote a signature scheme with security parameter n.

```
**Experiment** $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathrm{Dss}(1^n)}(\mathcal{A})$
    $(\mathsf{sk}, \mathsf{pk}) \longleftarrow \mathsf{Kg}(1^n)$
    $(M^\star, \sigma^\star) \longleftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
    Let $\{(M_i, \sigma_i)\}_1^q$ be the query-answer pairs of $\mathsf{Sign}(\mathsf{sk}, \cdot)$.
    Return 1 iff $\mathsf{Vf}(\mathsf{pk}, M^\star, \sigma^\star) = 1$ and $M^\star \notin \{M_i\}_1^q$.
```

For the success probability of an adversary $\mathcal{A}$ in the above experiment we write

$$\mathrm{Succ}^{\text{EU-CMA}}_{\mathrm{Dss}(1^n)}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{Exp}^{\text{EU-CMA}}_{\mathrm{Dss}(1^n)}(\mathcal{A}) = 1\right].$$

A signature scheme is called EU-CMA-secure if any PPT adversary has only negligible success probability:

Definition 2.3 (EU-CMA). *Let $n \in \mathbb{N}$,* Dss *a digital signature scheme as defined above. We call* Dss EU-CMA*-secure if for all $q, t = \mathrm{poly}(n)$ the maximum success probability* $\mathrm{InSec}^{\text{EU-CMA}}(\mathrm{Dss}(1^n); t, q)$ *of all possibly probabilistic adversaries $\mathcal{A}$ running in time $\le t$, making at most q queries to* Sign *in the above experiment, is negligible in n:*

$$\mathrm{InSec}^{\text{EU-CMA}}(\mathrm{Dss}(1^n); t, q) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\text{EU-CMA}}_{\mathrm{Dss}(1^n)}(\mathcal{A})\} = \mathrm{negl}(n).$$

An EU-CMA secure one-time signature scheme (OTS) is a Dss that is EU-CMA secure as long as the number of oracle queries of the adversary is limited to one, i.e. $q = 1$. In the case of SU-CMA security the adversary also succeeds if it returns a new signature on a message sent to Sign before. Hence, we obtain a definition for SU-CMA security replacing the last line in $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\mathrm{Dss}(1^n)}(\mathcal{A})$ by

$$\text{Return 1 iff } \mathsf{Vf}(\mathsf{pk}, M^\star, \sigma^\star) = 1 \text{ and } (M^\star, \sigma^\star) \notin \{(M_i, \sigma_i)\}_1^q.$$

2.2 Hash-based Signature Schemes

The signature schemes that we present in this work are hash-based signature schemes. The idea of hash-based signatures was introduced by Merkle [Mer90a] and the results in [BM96, BDE⁺11, BDK⁺07, BDS08, BDS09, BGD⁺06, DOTV08, DSS05, Gar05, HM02, JLMS03, Szy04] improve the Merkle idea in many respects by providing new algorithmic ideas and security proofs. In this section we present the basic ideas of hash based signature schemes. Moreover, we discuss so called tree traversal algorithms. These algorithms play a central role regarding the efficiency of hash based signature schemes and we will refer to this subsection several times. We begin this section with a hash-based one-time signature scheme.

2.2.1 The Lamport-Diffie One-Time Signature Scheme

At the heart of a hash-based signature scheme is a hash-based OTS. The first and most intuitive proposal for such an OTS is Lamport's scheme (sometimes called Lamport-Diffie OTS) [Lam79]. With this scheme the basic ideas for hash-based OTS were introduced. The OTS we present later in this work are based on the same general idea. The scheme uses any one-way function H, i.e. $\mathrm{H} = \mathrm{H}_K \in \mathcal{H}_n$ as defined above, and signs l bit messages. The secret key consists of $2l$ random bit strings

$$\mathsf{sk} = (\mathsf{sk}_{1,0}, \mathsf{sk}_{1,1}, \dots, \mathsf{sk}_{l,0}, \mathsf{sk}_{l,1})$$

of length m. The public key consists of the $2l$ outputs of the one-way function

$$\mathsf{pk} = (\mathsf{pk}_{1,0}, \mathsf{pk}_{1,1}, \dots, \mathsf{pk}_{l,0}, \mathsf{pk}_{l,1}) = (\mathrm{H}(\mathsf{sk}_{1,0}), \mathrm{H}(\mathsf{sk}_{1,1}), \dots, \mathrm{H}(\mathsf{sk}_{l,0}), \mathrm{H}(\mathsf{sk}_{l,1}))$$

when evaluated on the elements of the secret key. Signing a message $M \in \{0,1\}^l$ corresponds to publishing the corresponding elements of the secret key:

$$\sigma = (\sigma_1, \dots, \sigma_l) = (\mathsf{sk}_{1,M_1}, \dots, \mathsf{sk}_{l,M_l}).$$

To verify a signature the verifier checks whether the elements of the signature are mapped to the right elements of the public key using H:

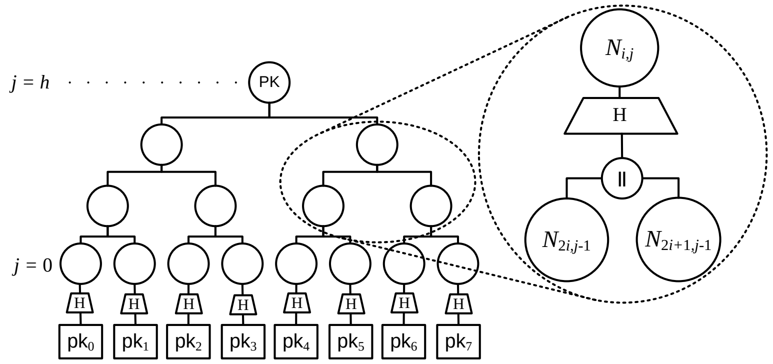
$$(\mathrm{H}(\sigma_1), \dots, \mathrm{H}(\sigma_l)) \overset{?}{=} (\mathsf{pk}_{1,M_1}, \dots, \mathsf{pk}_{l,M_l})$$

The reason for the scheme being one-time is that a signature contains half of the secret key. Hence, an adversary would simply ask for two signatures – namely the signatures on 0^l and 1^l – to be able to sign any message of her choice. However, if an adversary only knows the signature on one message, an existential forgery would contain a preimage for at least one element of the public key that was not contained in the signature before. The scheme can be proven EU-CMA-secure if the function used is one-way [BDS09]. It is also straightforward to prove that the scheme is SU-CMA-secure if the used function is second-preimage resistant.

2.2.2 The Merkle Signature Scheme

Given an OTS like the one from above we now show how to construct a many-time signature scheme. The method was introduced by Merkle in [Mer90a]. For this reason hash-based signature schemes are sometimes also called Merkle Signature Schemes (MSS). The many-time signature schemes presented in this work are based on this method but we use and introduce some improvements for the basic method presented here.

Figure 2.1: The Merkle Tree construction



The idea is to use many OTS key pairs and authenticate the public keys using a binary authentication tree, called Merkle Tree. The root of the tree becomes the new public key and the OTS secret keys become the overall secret key. More specifically, given a tree height h and a collision-resistant hash function $\mathrm{H} = \mathrm{H}_K \in \mathcal{H}_n$ the key generation works as follows. First 2^h OTS key pairs $(\mathsf{pk}_i, \mathsf{sk}_i), 0 \leq i < 2^h$ are generated. The secret key $\mathsf{SK} = (\mathsf{sk}_0, \ldots, \mathsf{sk}_{2^h-1})$ consists of the 2^h OTS secret keys. The 2^h leaves of the tree are the hash values of the 2^h OTS public keys using H: $L_i = \mathrm{H}(\mathsf{pk}_i)$. The nodes $N_{i,j}$ in the tree are computed as the hash value of the concatenation of their child nodes:

$$N_{i,j} = \mathrm{H}(N_{2i,j-1} || N_{2i+1,j-1})$$

for $0 < j \leq h$. The single hash value that represents the root of the tree is the overall public key PK. The construction of the public key is shown in Figure 2.1.

To sign the ith message the ith OTS secret key is used. The signature $\Sigma = (i, \sigma_i, \mathsf{pk}_i, \mathsf{Auth}_i)$ contains the index i, the obtained OTS signature σ_i, the corresponding OTS public key pk_i and the authentication path for pk_i. The authentication path for pk_i is the set of all siblings of the nodes on the path from L_i to the root. Figure 2.2 shows the authentication path for some pk_i. To verify a signature, first the OTS signature is verified. Then a root value is computed using the OTS public key from the signature and the authentication path. If the obtained root value equals the one contained in PK, the signature is accepted. Otherwise it is rejected.

It can be shown that this construction is EU-CMA-secure if the used OTS is EU-CMA-secure and the used hash function is taken from a collision resistant hash function family [BDS09]. The intuition for the proof is that an adversary, that successfully forges a signature, either generated a forgery for one of the OTS key pairs or that it replaced one of the OTS key pairs by one for which it knows the secret key. In the second case, it follows from the pigeonhole principle that the authentication path for the original OTS public key and that for the OTS public key contained in the forgery must collide at some point. This gives us a collision for H.

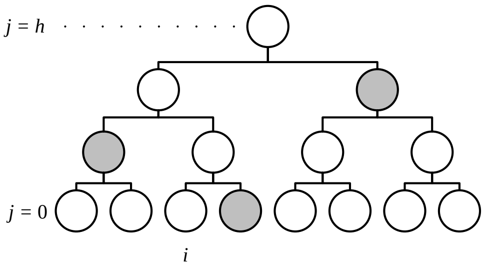
To understand the next subsection, one improvement of this basic scheme is needed. The construction described here results in a secret key that grows linearly in the number of signatures a key pair can be used for. For this reason, all practical MSS constructions use a pseudorandom generator to generate the OTS key pairs. In this case, the OTS key pairs are deleted after key generation and recomputed when needed.

2.2.3 Tree Traversal Algorithms

The most costly part in the signature generation algorithm is the computation of the authentication path described above. For this task so called tree traversal algorithms are used. An authentication path for a tree of height h consists of h nodes, one node on each level $0 \leq j < h$ of the tree. The computation of a node on level j requires the computation of 2^j leaves and $2^j - 1$ computations of inner nodes of the tree. In many cases the authentication paths of two sequential signatures differ only in a few nodes. Therefore, it is possible to reduce the number of nodes that have to be computed by storing the last authentication path in a state. However, in the worst case a new node on each level is required. Hence, $2^h - 1$ leaves and $2^h - h$ inner nodes have to be computed if the nodes are computed when needed.

There exist several proposals for tree traversal algorithms [BKN07, BDS08, Szy04, JLMS03] that are much more efficient than the straightforward approach above. A detailed overview can be found in [BDS09]. In this work we use the BDS algorithm by Buchmann et al. [BDS08] because it is the only algorithm that takes into account that leaf computation requires more effort than the computation of an inner node.

Figure 2.2: The authentication path for leaf i



This difference in computational costs grows if the OTS public keys are not stored, which is the case for all our MSS proposals. The basic building block of BDS is the TreeHash algorithm proposed by Merkle [Mer90a] that can be used to compute a node of the tree. Algorithm 2.1 shows TreeHash as used by BDS. The algorithm works on a stack Stack that is updated each time the algorithm is called. As input the algorithm takes Stack and the index of the next leaf to process. It outputs an updated Stack. The required node is the only remaining node on Stack after 2^j calls to TreeHash for a node on level j. It uses the method LeafCalc as a subroutine. Given an index, LeafCalc computes the corresponding leaf node of the tree. The implementation of LeafCalc depends on the MSS for which BDS is used.

Algorithm 2.1: TreeHash

```
Input: Stack, leaf index $\phi$
Output: Updated Stack
$N$= LeafCalc ($\phi$);
while top node on Stack has same height as $N$ do
    $N \longleftarrow$ H ((Stack.pop()||$N$));
end
Stack.push($N$);
return Stack
```

The BDS algorithm is parameterized by the BDS parameter k defining a time-memory trade-off. It reduces the worst case runtime per signature generation from $2^h - 1$ to $(h-k)/2$ evaluations of TreeHash and hence only $(h-k)/2$ leaf computations. More specifically, the BDS algorithm does three things. First, it uses the fact that a node which is a left child can be computed from values that occurred in an authentication path before, spending only one evaluation of H. Second, it stores the right nodes from the top k levels of the tree during key generation. Third, it distributes the computations for right child nodes among previous authentication path computations. BDS computes one node on every tree level $0 \leq j < h-k$ in parallel using Treehash. The main part of BDS is the scheduling of these $h-k$ computations. During every authentication path computation, BDS distributes $(h-k)/2$ updates among the running computations. This is done such that the computation of all nodes required for authentication path $i+1$ is finished during the ith signature generation. The computation of the next right node on a level starts when the last computed right node becomes part of the authentication path. To achieve this, BDS uses a state $\mathsf{State}_{\mathrm{BDS}}$ of no more than $\left(3h + \left\lfloor \frac{h}{2} \right\rfloor - 3k - 2 + 2^k\right)$ tree nodes that is initialized during key generation. To compute the authentication paths the

BDS algorithm spends $(h - k)/2$ leaf computations and evaluations of TreeHash per signature. For more details see [BDS08].

3 New Variants of the Winternitz One Time Signature Scheme

In this chapter we present the idea of the Winternitz One-Time Signature Scheme (W-OTS) first proposed in [Mer90a] and introduce two new variants of W-OTS. We prove the first variant strongly unforgeable under adaptive chosen message attacks in the standard model if a second-preimage resistant, undetectable one-way function family is used. For the second variant we prove existential unforgeability under adaptive chosen message attacks if it is instantiated using a pseudorandom function family. Towards this end, we introduce the notion of a function chain and give a generic description of W-OTS using this new notion. We introduce two security notions for function chains and use them to prove W-OTS EU-CMA and SU-CMA secure, respectively. Finally, we present two constructions of function chains that fulfill these security properties and lead to the two instantiations mentioned above. We start with the generic description of W-OTS, including the definition of a function chain. Afterwards, we present the security proofs and different chain constructions. The contributions of this chapter were published as parts of [1, 6, 7].

3.1 The Winternitz One-Time Signature Scheme

In this section we present a generic description of W-OTS using function chains, which we also introduce here. Intuitively, a function chain generates an ordered set – i.e. a chain – of values using a function (family) given a start value. The core idea of W-OTS is to use a certain number of such function chains starting from random values. These random values are the secret key. The public key consists of the final outputs of the chains, i.e. the end of each chain. A signature is computed by mapping the message to one intermediate value of each function chain. We start with the definition of function chains. Afterwards we present W-OTS.

3.1.1 Function Chains

In the following we formally introduce the notion of a function chain and two security properties for function chains. A function chain might be seen as some mode of operation for a function (family). More specifically, a function chain consists of two probabilistic polynomial time algorithms: An initialization algorithm $\mathcal{I}$ that generates a public chain key ck and an evaluation algorithm $\mathcal{E}$ that allows to compute the function chain. Actually, the evaluation algorithm does not simply compute the last value of the chain, but given a value of the chain it allows to perform one or more iterations of the function chain to obtain a succeeding value of the chain. There are two important details in our definition. First, the initialization algorithm takes the chain length as input. For this reason the length of a chain is fixed during initialization and the chain key can depend on the length. Furthermore, the evaluation algorithm takes as input which iteration of the chain the given input value belongs to and the output for which iteration of the chain should be computed, with the restriction that only the values of subsequent iterations can be computed. These two details make it possible, that iterations might differ, i.e. they might depend on their position in the chain. The notion of function chains might be interesting on its own, i.e. to formalize methods that make the computation of password digests more expensive to prevent brute-force attacks. Now, we first present the formal notion. For a better understanding, we give a little example afterwards. Then we define the security properties.

Definition 3.1 (Function Chain)**.** *Let $n \in \mathbb{N}$ be the security parameter, domain $\mathcal{D}$ and key space $\mathcal{K}$ be sets whose elements have a description length polynomial in n. A function chain $\mathcal{C} = (\mathcal{I},\mathcal{E})$ is a pair of probabilistic polynomial time algorithms*

- $\mathcal{I}(1^n, \lambda)$*, the initialization algorithm, on input of the security parameter n in unary and a chain length parameter $\lambda \in \mathbb{N}$ returns a public chain key $\mathsf{ck} \in \mathcal{K}$.*
- $\mathcal{E}^{i,j}_{\mathsf{ck}}(X)$*, the evaluation algorithm, on input of a value $X \in \mathcal{D}$, an interval $i, j \in \mathbb{N}, 0 \leq i \leq j \leq \lambda$ and a chain key $\mathsf{ck} \in \mathcal{K}$ returns $Y \in \mathcal{D}$, the jth value of the chain assuming X is the ith value of the chain.*

such that

$$(\forall n, \lambda \in \mathbb{N}), (\forall \mathsf{ck} \longleftarrow \mathcal{I}(1^n, \lambda)), (\forall i, j, m \in \mathbb{N}, 0 \leq i \leq j \leq m \leq \lambda), (\forall X \in \mathcal{D}) :$$

$$\mathcal{E}^{j,m}_{\mathsf{ck}}(\mathcal{E}^{i,j}_{\mathsf{ck}}(X)) = \mathcal{E}^{i,m}_{\mathsf{ck}}(X).$$

The correctness condition above simply says that it is possible to continue the evaluation of $\mathcal{C}$ from any intermediate value without changing the final result, as long as the same chain key is used. Please note that the above definition implies $\mathcal{E}^{i,j}_{\mathsf{ck}}(X) = X$ if $i = j$. To simplify notation, we omit the chain key whenever its value is clear from the context. We also omit the first index i for $\mathcal{E}$ if it is zero, i.e. $\mathcal{E}^{j}_{\mathsf{ck}}(X) = \mathcal{E}^{0,j}_{\mathsf{ck}}(X)$. We assume that the security parameter n, the chain length λ and the domain $\mathcal{D}$ can either be derived from ck or are publicly known.

To get a better idea of this notion, take the following example:

Construction 3.2 (Example)**.** *For a given $n \in \mathbb{N}$, let*

$$\mathcal{F}_n = \{\mathrm{F}_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^n\}$$

be a function family. Then we can construct a function chain with the following two algorithms operating on $\mathcal{K} = \mathcal{D} = \{0,1\}^n$:

- $\mathcal{I}(1^n, \lambda)$ *chooses a function key* $K \xleftarrow{\$} \{0,1\}^n$ *and returns* $\mathsf{ck} = (K, \lambda)$.
- $\mathcal{E}^{i,j}_{\mathsf{ck}}(X)$ *given* $X \in \{0,1\}^n$ *applies* F_K $j - i$ *times on* X*, i.e. it returns* $Y = \mathrm{F}_K^{j-i}(X) \in \{0,1\}^n$.

We leave it to the reader to check the correctness condition for this function chain.

We require a function chain to provide some security property when we use it for W-OTS. Informally, to achieve EU-CMA security we require the function chain to fulfill some kind of one-wayness. For the stronger notion of SU-CMA security, we need something like second-preimage resistance. We now define these two properties. We start with the one-wayness property. In previous works [GKL88, EGM96, DSS05] it was already analyzed what it means for the iteration of a function to be one-way. We define our notion for function chains similar to Dods et al. [DSS05] and reuse their name one-deeper preimage resistance (ODP).

The reasoning behind ODP is as follows. In most applications, including W-OTS, the final output Z of the function chain is used as some kind of verification value, similar to a message digest. Now, the adversary learns some intermediate value Y of the function chain that is the output of the ith iteration. A first idea would be to ask that it is impossible for the adversary to come up with a value X that is the output of a preceding iteration $j < i$ and maps to Y after $i - j$ iterations. But this misses the needs of the above application. The reason is that there might exist many more values that are possible outcomes of an iteration $j < i$, which lead to Z as final output of the chain, but do not lead to Y as the ith intermediate value. For this reason, we say that the adversary quasi inverts the function chain and thereby breaks the ODP property if it comes up with such a X. This is formalized in the following success probability of an adversary $\mathcal{A}$ against the ODP resistance of a function chain $\mathcal{C}(1^n, \lambda)$:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A}) = &\Pr\big[\,\mathsf{ck} \longleftarrow \mathcal{I}(1^n,\lambda); U \xleftarrow{\$} \mathcal{D}, i \xleftarrow{\$} [1,\lambda]; Y \longleftarrow \mathcal{E}^{i}_{\mathsf{ck}}(U),\\ &(X,j) \xleftarrow{\$} \mathcal{A}(\mathsf{ck}, i, Y) : 0 \le j < i; \mathcal{E}^{i,\lambda}_{\mathsf{ck}}(Y) = \mathcal{E}^{j,\lambda}_{\mathsf{ck}}(X)\big]\end{aligned}$$

Using this, we define ODP for function chains as follows

Definition 3.3 (ODP for function chains). *Let $n, \lambda, t \in \mathbb{N}$, $t = \mathrm{poly}(n)$, $\mathcal{C}$ =($\mathcal{I}$,$\mathcal{E}$) a function chain as described in Definition 3.1. We call $\mathcal{C}(1^n, \lambda)$ ODP resistant if the success probability $\mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A})$ of any adversary $\mathcal{A}$ running in time less or equal t is negligible in n:*

$$\mathrm{InSec}^{\mathrm{ODP}}(\mathcal{C}(1^n,\lambda); t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A})\} = \mathrm{negl}(n)\,.$$

To achieve SU-CMA security we will also need some kind of second-preimage resistance that we call second-origin resistance (SO). In this case, the intuition is that it should also be impossible for the adversary to come up with a new Y' which is a possible output of the ith iteration and maps to the same final output $Z' = Z$ as Y. Obviously, it makes no sense to allow $i = \lambda$ in this case as this would mean that we have two contradicting requirements, i.e. $Y \neq Y'$ and $Y = Z = Z' = Y'$. We formalize this using the following success property of an adversary $\mathcal{A}$ against the SO resistance of a function chain $\mathcal{C}(1^n, \lambda)$:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{SO}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A}) = &\Pr\big[\,\mathsf{ck} \longleftarrow \mathcal{I}(1^n,\lambda); U \xleftarrow{\$} \mathcal{D}, i \xleftarrow{\$} [0,\lambda-1]; Y \longleftarrow \mathcal{E}^{i}_{\mathsf{ck}}(U),\\ &(Y') \xleftarrow{\$} \mathcal{A}(\mathsf{ck}, i, Y) : Y \neq Y'; \mathcal{E}^{i,\lambda}_{\mathsf{ck}}(Y) = \mathcal{E}^{i,\lambda}_{\mathsf{ck}}(Y')\big]\end{aligned}$$

Using this, we define SO for function chains as follows

Definition 3.4 (SO for function chains). *Let $n, \lambda, t \in \mathbb{N}$, $t = \mathrm{poly}(n)$, $\mathcal{C}$ =($\mathcal{I}$,$\mathcal{E}$) a function chain as described in Definition 3.1. We call $\mathcal{C}(1^n, \lambda)$ SO resistant if the success probability $\mathrm{Succ}^{\mathrm{SO}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A})$ of any adversary $\mathcal{A}$ running in time less or equal t is negligible in n:*

$$\mathrm{InSec}^{\mathrm{SO}}(\mathcal{C}(1^n,\lambda); t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{SO}}_{\mathcal{C}(1^n,\lambda)}(\mathcal{A})\} = \mathrm{negl}(n)\,.$$

3.1.2 W-OTS

Now, we give a generic description of W-OTS using function chains as defined above. We start describing the parameters of the scheme. Afterwards, we present the three algorithms of the signature scheme.

Parameters: W-OTS is parameterized by security parameter n, the binary message length m, and the Winternitz parameter $w \in \mathbb{N}, w > 1$ that determines the time-memory trade-off. The last two parameters are used to compute

$$\ell_1 = \left\lceil \frac{m}{\log(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2.$$

Moreover, it uses a function chain $\mathcal{C}$. These parameters are publicly known. We can now describe the three algorithms of the scheme.

Key Generation Algorithm ($\mathsf{Kg}(1^n)$)**:** On input of security parameter n in unary the key generation algorithm choses ℓ values $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell) \xleftarrow{\$} \mathcal{D}^\ell$ uniformly at random that form the secret key. Next, Kg initializes the function chain $\mathsf{ck} \longleftarrow \mathcal{I}(1^n, w-1)$ and obtains a chain public key. The public verification key pk is computed as

$$\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (\mathsf{ck}, \mathcal{E}_{\mathsf{ck}}^{w-1}(\mathsf{sk}_1), \ldots, \mathcal{E}_{\mathsf{ck}}^{w-1}(\mathsf{sk}_\ell)) \,.$$

Signature Algorithm ($\mathsf{Sign}(M, \mathsf{sk})$)**:** On input of a message $M \in \{0,1\}^m$ and the secret signing key sk, the signature algorithm first computes a base w representation of M: $M = (b_1 \ldots b_{\ell_1}),\ b_i \in \{0, \ldots, w-1\}$. Next it computes the checksum

$$C = \sum_{i=1}^{\ell_1} (w - 1 - b_i)$$

and represents it as ℓ_2 base w numbers $C = (b_{\ell_1+1}, \ldots, b_\ell)$. The length of the base-w representation of C is at most ℓ_2 since $C \leq \ell_1(w-1)$. We set $B = (b_1, \ldots, b_\ell) = M \parallel C$, the concatenation of the base w representations of M and C. The signature is computed as

$$\sigma = (\sigma_1, \ldots, \sigma_\ell) = (\mathcal{E}_{\mathsf{ck}}^{b_1}(\mathsf{sk}_1), \ldots, \mathcal{E}_{\mathsf{ck}}^{b_\ell}(\mathsf{sk}_\ell).$$

Please note that the checksum guarantees that given the B corresponding to one message M, for any other message $M' \neq M$ the corresponding B' includes at least one $b'_i < b_i$ for $0 < i \leq \ell$.

Verification Algorithm ($\mathsf{Vf}(M, \sigma, \mathsf{pk})$)**:** On input of message M of binary length m, a signature σ and a public verification key pk, the verification algorithm first computes the b_i, $1 \leq i \leq \ell$ as described above. Then it does the following comparison:

$$(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) \stackrel{?}{=} (\mathcal{E}_{\mathsf{ck}}^{b_1, w-1}(\sigma_1), \ldots, \mathcal{E}_{\mathsf{ck}}^{b_\ell, w-1}(\sigma_\ell))$$

If the comparison holds, it returns **true** and **false** otherwise.

3.2 The Security of Generic W-OTS

In this section we analyze the security of the generic W-OTS description from the last section. We show that W-OTS is existentially unforgeable under adaptive chosen message attacks if the used function chain is one-deeper preimage resistant. Furthermore, we show that W-OTS is strongly unforgeable under adaptive chosen message attacks if the function chain is second-origin resistant. More precisely, we prove the following lemma:

Lemma 3.5. *Let $n, w, m \in \mathbb{N}$, $w, m = poly(n)$, $\mathcal{C}$ a function chain. Denote by* $\mathsf{t_{Kg}}$ ($\mathsf{t_{Sign}}, \mathsf{t_{Vf}}$) *the runtime of the W-OTS key generation (signature, verification) algorithm, respectively. Then the insecurity of W-OTS against an* EU-CMA *attack,* $\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}(1^n, w, m); t, 1\right)$*, is bounded by*

$$\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}(1^n, w, m); t, 1\right) \\ \leq \ell w \cdot \mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}(1^n, w-1); t'\right)$$

with $t' = t + \mathsf{t_{Kg}} + \mathsf{t_{Sign}} + \mathsf{t_{Vf}}$.

Moreover, we can bound $\mathrm{InSec}^{\text{SU-CMA}}\left(\textit{W-OTS}(1^n, w, m); t, 1\right)$*, the insecurity of W-OTS against an* SU-CMA *attack, by*

$$\mathrm{InSec}^{\text{SU-CMA}}\left(\textit{W-OTS}(1^n, w, m); t, 1\right) \\ \leq \ell w \cdot \left(\mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}(1^n, w-1); t'\right) + \mathrm{InSec}^{\text{SO}}\left(\mathcal{C}(1^n, w-1); t'\right)\right)$$

with $t' = t + \mathsf{t_{Kg}} + \mathsf{t_{Sign}} + \mathsf{t_{Vf}}$.

The proof of this lemma is rather straight forward. The intuition is that in the case of EU-CMA security, a forgery must contain a one-deeper preimage for one of the ℓ chains. This is guaranteed by the construction of the checksum. In the reduction we simply guess this chain. In case of SU-CMA security, the forgery contains either a one-deeper preimage or a second-origin. Again we guess the chain.

Proof. **Part 1 - EU-CMA:** We begin with the EU-CMA case. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ that can produce existential forgeries for W-OTS$(1^n, w, m)$ running an adaptive chosen message attack in time t and with success probability ϵ greater the claimed bound $\ell w \cdot \mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}(1^n, w-1); t'\right)$. We show how to construct an oracle machine $\mathcal{M}^{\mathcal{A}}$ that breaks the one-deeper preimage resistance of $\mathcal{C}(1^n, w-1)$ in time $t' \approx t$ with success probability greater $\mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}(1^n, w); t\right)$, leading to the required contradiction. A pseudo-code version of $\mathcal{M}^{\mathcal{A}}$ is given as Algorithm 3.1.

Algorithm 3.1: $\mathcal{M}^{\mathcal{A}}$

```
Input: Chain key $\mathsf{ck}$, iteration count $i$ and ODP challenge $Y_c$
Output: A one-deeper preimage $X \in \mathcal{D}$
Run $\mathsf{Kg}(1^n)$ with $w = \lambda + 1$, using $\mathsf{ck}$ as chain key to generate a W-OTS key pair $(\mathsf{sk}, \mathsf{pk})$;
Choose $\alpha \xleftarrow{\$} \{1, ..., \ell\}$;
Replace $\mathsf{pk}_\alpha$ by $\mathsf{pk}'_\alpha \longleftarrow \mathcal{E}_{\mathsf{ck}}^{i,w-1}(Y_c)$;
Run $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$;
if $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$ queries $\mathsf{Sign}$ with message $M$ then
    Compute $B = (b_1, ..., b_\ell)$;
    if $b_\alpha < i$ then
        return fail;
    else
        // Generate signature $\sigma$ of $M$
        Run $\sigma = (\sigma_1, \ldots, \sigma_\ell) \longleftarrow \mathsf{Sign}(M, \mathsf{sk})$;
        Set $\sigma_\alpha = \mathcal{E}_{\mathsf{ck}}^{i,b_\alpha}(Y_c)$;
        Reply to query with $\sigma$;
    end
end
if $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$ returns valid $(\sigma', M')$ then
    compute $B' = (b'_1, ..., b'_\ell)$;
    if $b'_\alpha \geq i$ then
        return fail;
    else
        return $\mathcal{E}_{\mathsf{ck}}^{b'_\alpha,b_\alpha-1}(\sigma_\alpha)$;
    end
else
    return fail;
end
```

On input of a chain key ck, iteration count i and ODP challenge Y_c $\mathcal{M}^{\mathcal{A}}$ generates a W-OTS public key for $w = \lambda + 1$ using ck as chain key. Next, $\mathcal{M}^{\mathcal{A}}$ selects a random chain α and places the challenge there. This is done computing the remaining $w - 1 - i$ iterations to obtain the output of the chain and using it to replace the value in the public key that corresponds to chain α (Line 3). Please note that the resulting public key is distributed in the same way as a public key generated by Kg. The reason is that Y_c is the ith intermediate value of a chain. The difference to a normal key pair is that $\mathcal{M}^{\mathcal{A}}$ now only knows the intermediate values of chain α for iterations $\geq i$. For the remaining chains, all values are known. For this reason, if $\mathcal{A}$ makes a query to the signing oracle, $\mathcal{M}^{\mathcal{A}}$ can answer the query correctly if $b_\alpha \geq i$ (Line 7). In this case, $\mathcal{M}^{\mathcal{A}}$ computes the signature using the secret key values and Y_c (Lines 10&11). Otherwise it returns fail. Now, if $\mathcal{A}$ returns a valid forgery, $\mathcal{M}^{\mathcal{A}}$ only learns something new if $b'_\alpha < i$ (Line 17). In this case, $\mathcal{M}^{\mathcal{A}}$ extracts the one-deeper preimage and returns it (Line 20). Otherwise it returns fail.

Now we compute $\mathcal{M}^{\mathcal{A}}$'s success probability. $\mathcal{M}^{\mathcal{A}}$ succeeds, if the break conditions on lines 7 and 17 are not fulfilled and $\mathcal{A}$ returns a valid forgery. We do the analysis in the reverse order. By assumption, $\mathcal{A}$ succeeds with probability ϵ. Because of the checksum construction, there must exist at least one $1 \leq j \leq w - 1$ with $b'_j < b_j$. To simplify the analysis, we compute the probability that $\alpha = j$ and $b_\alpha = i$, i.e. that we placed Y_c in the perfect position. In this case $b_\alpha \geq i$ and $b'_\alpha < i$ and both break conditions are not fulfilled. This happens with probability $\geq w^{-1}\ell^{-1}$, because we chose α uniformly at random and $b_\alpha > 0$ must hold. Putting things together we get:

$$\mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}(1^n, w-1)}\left(\mathcal{M}^{\mathcal{A}}\right) \geq \frac{1}{w\ell} \cdot \epsilon.$$

The runtime t' of $\mathcal{M}^{\mathcal{A}}$ is the runtime t of $\mathcal{A}$ plus the time needed to run Kg, Sign and Vf ($t' = t + \mathsf{t_{Kg}} + \mathsf{t_{Sign}} + \mathsf{t_{Vf}}$). Using that ϵ is greater than the claimed bound leads to the required contradiction.

Part 2 - SU-CMA: Now we turn to the SU-CMA case. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ that can produce strong forgeries for W-OTS$(1^n, w, m)$ running an adaptive chosen message attack in time t and with success probability ϵ greater than the claimed bound $\ell w \cdot (\mathrm{InSec}^{\mathrm{ODP}}(\mathcal{C}(1^n, w-1); t')$ $+\mathrm{InSec}^{\mathrm{SO}}(\mathcal{C}(1^n, w-1); t'))$. Then there are two mutually exclusive cases. First, $\mathcal{A}$ might still return a forgery for a new message. In this case we can use $\mathcal{M}^{\mathcal{A}}$ to break the ODP resistance of $\mathcal{C}(1^n, w-1)$. Second, $\mathcal{A}$ might return a new signature for the message sent to Sign before. For this case we can build an oracle machine $\mathcal{M}'^{\mathcal{A}}$ that breaks the SO resistance of $\mathcal{C}(1^n, w-1)$. $\mathcal{M}'^{\mathcal{A}}$ takes the same inputs (ck, i, Y_c) as $\mathcal{M}^{\mathcal{A}}$, except that i now is in the range $[0, \lambda - 1]$. It also behaves like $\mathcal{M}^{\mathcal{A}}$ up to the point when $\mathcal{A}$ returns the forgery. At this point $\mathcal{M}'^{\mathcal{A}}$ returns a second origin σ'_α if $Y_c \neq \sigma'_\alpha$ and it returns fail otherwise.

We first analyze the success probability for the two cases. As they are mutually exclusive, they occur with inverse probabilities. Assume $\mathcal{A}$ returns a signature on a new message under the condition that it succeeds with probability p, then the success probability of $\mathcal{M}^{\mathcal{A}}$ is the same as above:

$$\mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}(1^n,w-1)}\left(\mathcal{M}^{\mathcal{A}}\right) \geq \frac{p}{w\ell} \cdot \epsilon.$$

Now under the condition that $\mathcal{A}$ succeeds, it returns a new signature on M with probability $1-p$. Then $b_j = b'_j$ for all $1 \leq j \leq \ell$ and there must exist at least one $\sigma_j \neq \sigma'_j$. With probability $\geq w^{-1}\ell^{-1}$ it holds that $\alpha = j$ and $b_\alpha = i$ as both values are uniformly random. In this case $\mathcal{M}'^{\mathcal{A}}$ can answer the signature query and extract a second origin. By assumption, $\mathcal{A}$ succeeds with probability ϵ and so we get

$$\mathrm{Succ}^{\mathrm{SO}}_{\mathcal{C}(1^n,w-1)}\left(\mathcal{M}'^{\mathcal{A}}\right) \geq \frac{1-p}{w\ell} \cdot \epsilon.$$

The runtime t' of $\mathcal{M}'^{\mathcal{A}}$ is equal to the runtime of $\mathcal{M}^{\mathcal{A}}$ which is the runtime t of $\mathcal{A}$ plus the time needed to run Kg, Sign and Vf ($t' = t + \mathsf{t}_{\mathsf{Kg}} + \mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$). Now we can put the two bounds together and use that $\mathcal{M}^{\mathcal{A}}$'s success probability in both cases is limited by the corresponding insecurity functions. Then we get

$$\epsilon \leq w\ell \cdot \left(\mathrm{InSec}^{\mathrm{ODP}}\left(\mathcal{C}(1^n, w-1); t'\right) + \mathrm{InSec}^{\mathrm{SO}}\left(\mathcal{C}(1^n, w-1); t'\right)\right)$$

which contradicts our initial assumption. □

3.3 A Function Chain using Second-Preimage Resistance

In this section we present the construction of a function chain $\mathcal{C}^+$. The presented function chain $\mathcal{C}^+$ is ODP and SO resistant if the internally used function is second-preimage resistant, one-way and undetectable, i.e. the output of the function is pseudorandom. This function chain $\mathcal{C}^+$ then directly leads to a W-OTS construction W-OTS$^+$ that is SU-CMA secure under these conditions. We now first present the construction for $\mathcal{C}^+$. Afterwards we formally define undetectability and prove that $\mathcal{C}^+$ is ODP and SO resistant.

3.3.1 The $\mathcal{C}^+$ Function Chain

We now describe the function chain $\mathcal{C}^+$. For a given security parameter n it uses a family of functions

$$\mathcal{F}_n : \{\mathrm{F}_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^k\} \tag{3.1}$$

where $k = \text{poly}(n)$. The function chain $\mathcal{C}^+$ works over domain $\mathcal{D} = \{0,1\}^n$ and has key space $\mathcal{K} = \{0,1\}^k \times \{0,1\}^{n\times\lambda}$. The algorithms are instantiated as follows:

$\mathcal{I}(1^n, \lambda)$: On input of the security parameter n in unary and the chain length λ the initialization algorithm choses a function F_K from $\mathcal{F}_n$ by choosing a $K \xleftarrow{\$} \{0,1\}^k$ uniformly at random. Next it chooses a vector $\mathbf{R} = (R_1, \dots, R_\lambda) \xleftarrow{\$} \{0,1\}^{n\times\lambda}$ of λ bit strings, each of length n uniformly at random. We call $\mathbf{R}$ the randomization elements. The algorithm returns $\mathsf{ck} = (K, \mathbf{R})$.

$\mathcal{E}^{i,j}_{\mathsf{ck}}(X)$: On input of value $X \in \mathcal{D}$, interval $0 \le i \le j \le \lambda$ and chain key ck the evaluation function works the following way. According to the definition of a function chain, $\mathcal{E}$ returns X if $i = j$ ($\mathcal{E}^{i,i}_{\mathsf{ck}}(X) = X$). For $i < j$ we define $\mathcal{E}$ recursively by

$$\mathcal{E}^{i,j}_{\mathsf{ck}}(X) = \mathrm{F}_K(\mathcal{E}^{i,j-1}_{\mathsf{ck}}(X) \oplus R_j),$$

i.e. in every round, the function first takes the bitwise xor of the intermediate value and the corresponding randomization element R_j and evaluates F_K on the result afterwards.

Correctness of $\mathcal{C}^+$ directly follows from the construction of $\mathcal{E}$ above.

3.3.2 Preliminaries

We now provide two definitions used in this section. In the following we use the (distinguishing) advantage of an adversary which we now define.

Definition 3.6 (Advantage)**.** *Given two distributions $\mathcal{X}$ and $\mathcal{Y}$, we define the advantage* $\mathrm{Adv}_{\mathcal{X},\mathcal{Y}}(\mathcal{A})$ *of an adversary $\mathcal{A}$ in distinguishing between these two distributions as*

$$\mathrm{Adv}_{\mathcal{X},\mathcal{Y}}(\mathcal{A}) = |\mathsf{Pr}[1 \longleftarrow \mathcal{A}(\mathcal{X})] - \mathsf{Pr}[1 \longleftarrow \mathcal{A}(\mathcal{Y})]|.$$

We now define undetectability. In what follows, we only consider the families $\mathcal{F}_n$ defined in the last section (Equation 3.1). Intuitively, a function family is undetectable if its output can not be distinguished from uniformly random values. This is what we require from a pseudorandom generator, which in contrast to $\mathcal{F}_n$ has to be length expanding. Here we define it for length preserving function families.

To define undetectability, assume the following two distributions over $\{0,1\}^n \times \{0,1\}^k$. A sample (U, K) from the first distribution $\mathcal{D}_{\text{UD},\mathcal{U}}$ is obtained by sampling $U \xleftarrow{\$} \{0,1\}^n$ and $K \xleftarrow{\$} \{0,1\}^k$ uniformly at random from the respective domain. A sample (U, K) from the second distribution $\mathcal{D}_{\text{UD},\mathcal{F}}$ is obtained by sampling $K \xleftarrow{\$} \{0,1\}^k$ and then evaluating F_K on a uniformly random bit string, i.e. $U \longleftarrow \mathrm{F}_K(\mathcal{U}_n)$. The advantage of an adversary $\mathcal{A}$ against the undetectability of $\mathcal{F}_n$ is then defined as the distinguishing advantage for these two distributions:

$$\mathrm{Adv}^{\text{UD}}_{\mathcal{F}_n}(\mathcal{A}) = \mathrm{Adv}_{\mathcal{D}_{\text{UD},\mathcal{U}},\mathcal{D}_{\text{UD},\mathcal{F}}}(\mathcal{A})$$

Using this we define undetectability as:

Definition 3.7 (Undetectability (UD)). *Let $n \in \mathbb{N}, t = \mathrm{poly}(n)$, $\mathcal{F}_n$ a family of functions as described above. We call $\mathcal{F}_n$ undetectable, if* $\mathrm{InSec}^{\text{UD}}(\mathcal{F}_n; t)$ *the advantage of any adversary $\mathcal{A}$ against the undetectability of $\mathcal{F}_n$ running in time less or equal t is negligible in n:*

$$\mathrm{InSec}^{\text{UD}}(\mathcal{F}_n; t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Adv}^{\text{UD}}_{\mathcal{F}_n}(\mathcal{A})\} = \mathrm{negl}(n)\,.$$

Undetectability was already used by Dods et al. [DSS05] in security proofs for W-OTS.

3.3.3 Security Proof

We now show that $\mathcal{C}^+$ is an ODP and SO resistant function chain if $\mathcal{F}_n$ is a second-preimage resistant, undetectable one-way function family. More formally we prove the following lemma:

Lemma 3.8. *Let $n, \lambda, t \in \mathbb{N}$, $\mathcal{F}_n$ as defined above be a second-preimage resistant, undetectable one-way function family. Then* $\mathrm{InSec}^{\text{ODP}}(\mathcal{C}^+(1^n, \lambda); t)$*, the* ODP *resistance of $\mathcal{C}^+$ is bounded by*

$$\begin{aligned}&\mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}^+(1^n, \lambda); t\right)\\ &\qquad \le \lambda \cdot \mathrm{InSec}^{\text{UD}}(\mathcal{F}_n; t^\star) + \mathrm{InSec}^{\text{OW}}(\mathcal{F}_n; t') + \lambda \cdot \mathrm{InSec}^{\text{SPR}}(\mathcal{F}_n; t'')\end{aligned}$$

with $t' = t + \lambda$, $t'' = t + 2\lambda$ and $t^\star = t + 2\lambda - 1$. Moreover, if $\mathcal{F}_n$ is a second-preimage resistant function family, $\mathrm{InSec}^{\text{SO}}(\mathcal{C}^+(1^n, \lambda); t)$*, the* SO *resistance of $\mathcal{C}^+$ is bounded by*

$$\mathrm{InSec}^{\text{SO}}\left(\mathcal{C}^+(1^n, \lambda); t\right) \le \lambda \cdot \mathrm{InSec}^{\text{SPR}}(\mathcal{F}_n; t')$$

with $t' = t + 2\lambda$.

This leads to our first main result. We call W-OTS using $\mathcal{C}^+$ as the function chain W-OTS$^+$. Combining the above result with Lemma 3.5 we obtain the following theorem on the security of W-OTS$^+$:

Theorem 3.9. *Let $n, w, m \in \mathbb{N}$, $w, m = \mathrm{poly}(n)$, $\mathcal{F}_n$ as defined above a second-preimage resistant, undetectable one-way function family. Then, the insecurity of W-OTS$^+$ against an* EU-CMA *attack,* $\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}^+(1^n, w, m); t, 1\right)$*, is bounded by*

$$\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}^+(1^n, w, m); t, 1\right) \\ \leq \ell w^2 \cdot \mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right) + \ell w \cdot \mathrm{InSec}^{\text{OW}}\left(\mathcal{F}_n; t'\right) + \ell w^2 \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{F}_n; t''\right)$$

with $t' = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + w$*,* $t'' = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + 2w$ *and* $t^\star = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + 2w - 1$*. Moreover, the insecurity of W-OTS$^+$ against an* SU-CMA *attack is bounded by*

$$\mathrm{InSec}^{\text{SU-CMA}}\left(\textit{W-OTS}^+(1^n, w, m); t, 1\right) \\ \leq \ell w^2 \cdot \mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right) + \ell w \cdot \mathrm{InSec}^{\text{OW}}\left(\mathcal{F}_n; t'\right) + \left(\ell w^2 + w\right) \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{F}_n; t''\right)$$

with $t' = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + w$*,* $t'' = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + 2w$ *and* $t^\star = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + 2w - 1$*.*

It seems natural to assume that the existence of a function that combines one-wayness, undetectability and second-preimage resistance is equivalent to the existence of a one-way function. As the function has to be one-way itself, the one direction is trivial. On the other hand, we know that second-preimage resistant functions exist if a one-way function exists [Rom90] and we know the same for undetectable functions, i.e. pseudorandom generators [HILL99]. We leave the question if this also implies the existence of a function family that combines all three properties for future work. If this was the case, it would mean that W-OTS$^+$ has minimal security requirements.

Before we present the proof, we now give some intuition. We begin with the ODP resistance. Recall, the adversary is given a chain key, an index i and the output of the ith iteration Y. Then it has to return a one-deeper preimage X of Y. Intuitively, if X leads to Y as the output of the ith iteration, it also leads to a real preimage of Y under F_K that we try to extract using the preimage challenge as Y. If X leads a different value than Y as output for the ith iteration, the values in the chains starting from Y and X must collide at some point, as they lead the same final output. This also means a collision for F_K. By manipulating the randomization elements in the chain key, we can use this second case to extract a second-preimage with high probability. The undetectability is used to show that the adversary still succeeds with sufficiently high probability although the Y is not an output of the ith iteration of the chain but a uniformly random value.

For the case of SO resistance, we proceed similar to the above, but we know in advance that only the second case can appear. For this reason, we do not have to place a preimage challenge in the chain and can use a Y that is really the output of the ith iteration of the chain. Hence, we also do not need the undetectability in this case.

Proof of Lemma 3.8.
Part 1 - SO resistance: We first prove the SO case, as we can reuse the reduction for the ODP case. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ that can break the SO resistance of $\mathcal{C}^{+}(1^n, \lambda)$ running in time $\leq t$ and with success probability $\epsilon_{\mathcal{A}} = \text{Succ}^{\text{SO}}_{\mathcal{C}^{+}(1^n,\lambda)}(\mathcal{A})$ greater than the claimed bound on $\text{InSec}^{\text{SO}}(\mathcal{C}^{+}(1^n, \lambda); t)$. We first show how to construct an oracle machine $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ that breaks the second-preimage resistance of $\mathcal{F}_n$ using $\mathcal{A}$. A pseudo-code description of $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ is given as Algorithm 3.2.

Algorithm 3.2: $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$

```
Input: Security parameter $n$, function key $K$, second-preimage resistance
       challenge $X_c$.
Output: A value $X$ that is a second-preimage for $X_c$ under $\mathrm{F}_K$ or fail.
Sample $\mathbf{R} \xleftarrow{\$} \{0,1\}^{n \times \lambda}$ to obtain a chain key $\mathsf{ck} = (K, \mathbf{R})$ for length $\lambda$;
Choose index $\alpha \xleftarrow{\$} \{0, \ldots, \lambda - 1\}$ uniformly at random;
Sample $U \xleftarrow{\$} \{0,1\}^n$ and compute $Y \longleftarrow \mathcal{E}^{\alpha}_{\mathsf{ck}}(U)$;
Choose index $\beta \xleftarrow{\$} \{\alpha + 1, \ldots, \lambda\}$ uniformly at random;
Obtain $\mathbf{R}'$ from $\mathbf{R}$, replacing $R_\beta$ by $\mathcal{E}^{\alpha,\beta-1}_{\mathsf{ck}}(Y) \oplus X_c$ ;
Set $\mathsf{ck}' = (K, \mathbf{R}')$;
Run $Y' \longleftarrow \mathcal{A}(\mathsf{ck}', \alpha, Y)$ ;
if $Y' \neq Y$ and $\mathcal{E}^{\alpha,\lambda}_{\mathsf{ck}}(Y') = \mathcal{E}^{\alpha,\lambda}_{\mathsf{ck}}(Y)$ then
    if $X' = \mathcal{E}^{\alpha,\beta-1}_{\mathsf{ck}}(Y') \oplus R_\beta \neq X_c$ and $\mathrm{F}_K(X') = \mathcal{E}^{\alpha,\beta}_{\mathsf{ck}}(Y) = \mathrm{F}_K(X_c)$ then
        return second-preimage $X'$ ;
    end
end
return fail;
```

The oracle machine $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ first imitates the initialization algorithm to obtain a valid chain key $\mathsf{ck} = (K, \mathbf{R})$ for length λ using the given function key K. Then, $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ selects an iteration index α uniformly at random to construct the SO challenge. Next, $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ samples a uniformly random element $U \xleftarrow{\$} \{0,1\}^n$ from the domain of the function chain and computes the output Y of the αth iteration. Then, another intermediate value β between α and the end of the chain is selected uniformly at random. $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ places the second-preimage challenge at the input to F_K during the βth iteration of the chain, replacing the randomization element R_β (Line 5). A manipulated chain key ck' is constructed using the new set of randomization elements. Then $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ runs $\mathcal{A}$ on input $(\mathsf{ck}', \alpha, Y)$.

If $\mathcal{A}$ returns a second origin Y', $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ tries to extract a second-preimage. Otherwise, $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ returns fail. Assume $\mathcal{A}$ returned a second origin. In this case the chains continued from Y and Y' must collide at some position between $\alpha + 1$ and λ according to the pigeonhole principle as they have the same final output. If they collide at position β for the first time, a second-preimage for X_c can be extracted (Line 10). Otherwise $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ aborts.

Now, we compute the success probability of $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$. As $\mathcal{A}$'s input distribution is correct, it will return a second-origin with probability $\epsilon_{\mathcal{A}}$. $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$ returns a second-preimage for X_c if the two chains collide for the first time at position β. This happens with probability greater λ^{-1} as β was chosen uniformly at random from within the interval $[\alpha + 1, \lambda]$.

Using that $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$'s success probability is limited by the corresponding insecurity function for the second-preimage resistance of $\mathcal{F}_n$, we can bound the success probability of $\mathcal{A}$ if called by $\mathcal{M}^{\mathcal{A}}_{\text{SPR}}$:

$$\epsilon_{\mathcal{A}} \leq \lambda \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{F}_n; t'\right). \tag{3.2}$$

The time $t' = t + 2\lambda$ is an upper bound for the runtime of $\mathcal{A}$ plus the time needed to compute all values for two chains of length $\leq \lambda$. This leads the required contradiction.

Part 2 - ODP resistance: We now prove the ODP case. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ that can break the ODP resistance of $\mathcal{C}^+(1^n, \lambda)$ running in time $\leq t$ and with success probability $\epsilon_{\mathcal{A}} = \mathrm{Succ}^{\text{ODP}}_{\mathcal{C}^+(1^n,\lambda)}(\mathcal{A})$ greater than the claimed bound on $\mathrm{InSec}^{\text{ODP}}\left(\mathcal{C}^+(1^n, \lambda); t\right)$. If $\mathcal{A}$ returns a one-deeper preimage (j, X) on input (ck, i, Y) there are two mutually exclusive cases that appear with complementary probability. Either $\mathcal{E}^{j;i}_{\mathsf{ck}}(X) = Y$ holds, i.e. Y is the ith intermediate value of the chain continued from X, or it does not hold. Denote the two cases by $c1$, i.e. the condition holds, and $c2$ otherwise. In the following we show how to construct an oracle machine $\mathcal{M}^{\mathcal{A}}_{\text{OW}}$, that breaks the one-wayness of $\mathcal{F}_n$ using $\mathcal{A}$

in case $c1$. A pseudo-code description of $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ is given as Algorithm 3.3. Otherwise, in case $c2$, we can use $\mathcal{M}^{\mathcal{A}}_{\mathrm{SPR}}$ from the SO proof above to extract a second-preimage. For this to work, we only have to replace Line 7 of $\mathcal{M}^{\mathcal{A}}_{\mathrm{SPR}}$ by

$$(j, X) \longleftarrow \mathcal{A}(\mathsf{ck}', \alpha, Y); Y' \longleftarrow \mathcal{E}^{j,\alpha}_{\mathsf{ck}}(X);$$

i.e. transfer the one-deeper preimage into a second-origin.

Algorithm 3.3: $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$

```
Input: Security parameter $n$, function key $K$, one-way challenge $Y_c$.
Output: A value $X$ that is a preimage of $Y_c$ under $\mathrm{F}_K$ or fail.
Sample $\mathbf{R} \xleftarrow{\$} \{0,1\}^{n \times \lambda}$ to obtain a chain key $\mathsf{ck} = (K, \mathbf{R})$ for length $\lambda$ ;
Choose index $\alpha \xleftarrow{\$} \{1, \ldots, \lambda\}$ uniformly at random ;
Run $(X, j) \longleftarrow \mathcal{A}(\mathsf{ck}, \alpha, Y_c)$ ;
if $j < \alpha$ and $\mathcal{E}^{\alpha,\lambda}_{\mathsf{ck}}(Y_c) = \mathcal{E}^{j,\lambda}_{\mathsf{ck}}(X)$ then
    // $\mathcal{M}^{\mathcal{A}}_{\mathrm{ow}}$ is used for the case $\mathcal{E}^{j,\alpha}_{\mathsf{ck}}(X) = Y_c$, so the following
       always works
    return preimage $\mathcal{E}^{j,\alpha-1}_{\mathsf{ck}}(X) \oplus R_\alpha$ ;
end
return fail;
```

The oracle machine $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ first imitates the initialization algorithm to obtain a valid chain key $\mathsf{ck} = (K, \mathbf{R})$ for length λ using the given function key K. Then, $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ selects the position to place its challenge in the chain key. Therefor, it samples an index α uniformly at random to select an intermediate value of this chain. Then $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ runs $\mathcal{A}$ on input $(\mathsf{ck}, \alpha, Y_c)$, i.e. giving Y_c to the adversary as the output of the αth iteration of the chain. If $\mathcal{A}$ returns a one-deeper preimage (j, X), $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ can always extract a preimage. This holds as $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ is used for the case that $\mathcal{A}$ returns one-deeper preimages that fulfill $\mathcal{E}^{j,\alpha}_{\mathsf{ck}}(X) = Y_c$. In any other case, $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ returns fail.

Now we compute the success probability of $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ conditioned on case $c1$, i.e. we assume whenever $\mathcal{A}$ succeeds, it returns a one-deeper preimage that fulfills $\mathcal{E}^{j,\alpha}_{\mathsf{ck}}(X) = Y_c$. Hence, if $\mathcal{A}$ succeeds, $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ returns a preimage of Y_c under F_K with probability 1. As our modifications might have changed the input distribution of $\mathcal{A}$, it does not necessarily succeed with probability $\epsilon_{\mathcal{A}}$. For the moment we denote the probability that $\mathcal{A}$ returns a one-deeper preimage when run by $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$ as $\epsilon'_{\mathcal{A}}$. Using that $\mathcal{M}^{\mathcal{A}}_{\mathrm{OW}}$'s success probability against the one-wayness of $\mathcal{F}_n$ is bound by the corresponding insecurity function, we can bound the success probability of $\mathcal{A}$ in case $c1$ if called by $\mathcal{M}^{\mathcal{A}}_{\text{OW}}$:

$$\epsilon'_{\mathcal{A}} \leq \text{InSec}^{\text{ow}}\left(\mathcal{F}_n; t'\right) \tag{3.3}$$

where the time $t' = t + \lambda$ is an upper bound for the runtime of $\mathcal{A}$ plus the time needed to compute all values for one chain of length $\leq \lambda$.

As a second step, we bound the difference between the success probability $\epsilon'_{\mathcal{A}}$ of $\mathcal{A}$ when called by $\mathcal{M}^{\mathcal{A}}_{\text{OW}}$ and its success probability $\epsilon_{\mathcal{A}}$ in the original experiment. If the first is greater than the latter this would already lead a contradiction. Hence, we assume $\epsilon'_{\mathcal{A}} \leq \epsilon_{\mathcal{A}}$ in what follows. Please note, that among $\mathcal{A}$'s inputs only the distribution of Y_c might differ from the distribution in the real game. We define two distributions $\mathcal{D}_{\mathcal{M}}$ and $\mathcal{D}_{\text{ODP}}$ over $\{0, \ldots, \lambda\} \times \{0,1\}^n \times \{0,1\}^{(n \times \lambda)} \times \{0,1\}^k$. A sample $(\alpha, U, \mathbf{R}, K)$ follows $\mathcal{D}_{\mathcal{M}}$ if the entries $\alpha \xleftarrow{\$} \{0, \ldots, \lambda\}$, $U \xleftarrow{\$} \{0,1\}^n$, $\mathbf{R} \xleftarrow{\$} \{0,1\}^{n \times \lambda}$ and $K \xleftarrow{\$} \{0,1\}^k$ are chosen uniformly at random. A sample $(\alpha, U, \mathbf{R}, K)$ follows $\mathcal{D}_{\text{ODP}}$ if α, $\mathbf{R}$ and K are chosen uniformly at random but $U = \mathcal{E}^{\alpha}_{\mathsf{ck}}(\mathcal{U}_n)$ with $\mathsf{ck} = (K, \mathbf{R})$. Thus, the two distributions only differ in the way U is chosen. We now construct an oracle machine $\mathcal{M}'^{\mathcal{A}}$ that uses the possibly different behavior of $\mathcal{A}$ when given differently distributed inputs to distinguish between $\mathcal{D}_{\text{ODP}}$ and $\mathcal{D}_{\mathcal{M}}$. Using $\mathcal{M}'^{\mathcal{A}}$ we can then upper bound $\epsilon_{\mathcal{A}}$ by a function of the distinguishing advantage of $\mathcal{M}'^{\mathcal{A}}$ and $\epsilon'_{\mathcal{A}}$. Afterwards we use a hybrid argument to bound the distinguishing advantage of $\mathcal{M}'^{\mathcal{A}}$ using the undetectability of $\mathcal{F}_n$.

The oracle machine $\mathcal{M}'^{\mathcal{A}}$ works the following way. On input of a sample $(\alpha, U, \mathbf{R}, K)$ that is either chosen from $\mathcal{D}_{\mathcal{M}}$ or from $\mathcal{D}_{\text{ODP}}$, $\mathcal{M}'^{\mathcal{A}}$ runs $\mathcal{A}$ on input (ck, α, U) with $\mathsf{ck} = (K, \mathbf{R})$. Whenever $\mathcal{A}$ returns a valid one-deeper preimage, $\mathcal{M}'^{\mathcal{A}}$ returns 1 and 0 otherwise. The runtime of $\mathcal{M}'^{\mathcal{A}}$ is bounded by the runtime of $\mathcal{A}$ plus no more than λ evaluations of $\mathcal{E}$. So we get $t'' = t + \lambda$ as an upper bound.

Now, we compute the distinguishing advantage $\text{Adv}_{\mathcal{D}_{\mathcal{M}}, \mathcal{D}_{\text{ODP}}}\left(\mathcal{M}'^{\mathcal{A}}\right)$ of $\mathcal{M}'^{\mathcal{A}}$. If the sample is taken from $\mathcal{D}_{\mathcal{M}}$, the distribution of $\mathcal{A}$'s inputs generated by $\mathcal{M}'^{\mathcal{A}}$ is the same as the distribution of the inputs generated by $\mathcal{M}^{\mathcal{A}}_{\text{OW}}$. Hence, $\mathcal{M}'^{\mathcal{A}}$ outputs 1 with probability

$$\mathsf{Pr}\left[(\alpha, U, \mathbf{R}, K) \longleftarrow \mathcal{D}_{\mathcal{M}} : 1 \longleftarrow \mathcal{M}'^{\mathcal{A}}(\alpha, U, \mathbf{R}, K)\right] = \epsilon'_{\mathcal{A}}.$$

If the sample was taken from $\mathcal{D}_{\text{ODP}}$, $\mathcal{A}$'s inputs generated by $\mathcal{M}'^{\mathcal{A}}$ follow the same distribution as those in the ODP game and so $\mathcal{M}'^{\mathcal{A}}$ outputs 1 with probability

$$\mathsf{Pr}\left[(\alpha, U, \mathbf{R}, K) \longleftarrow \mathcal{D}_{\text{ODP}} : 1 \longleftarrow \mathcal{M}'^{\mathcal{A}}(\alpha, U, \mathbf{R}, K)\right] = \epsilon_{\mathcal{A}}.$$

Hence, the distinguishing advantage of $\mathcal{M}'^{\mathcal{A}}$ is

$$\text{Adv}_{\mathcal{D}_{\text{ODP}}, \mathcal{D}_{\mathcal{M}}}\left(\mathcal{M}'^{\mathcal{A}}\right) = \left|\epsilon_{\mathcal{A}} - \epsilon'_{\mathcal{A}}\right|.$$

As mentioned above, we only have to consider the case $\epsilon_{\mathcal{A}} \geq \epsilon'_{\mathcal{A}}$. Therefore, we obtain the following bound on $\epsilon_{\mathcal{A}}$:

$$\epsilon_{\mathcal{A}} = \mathrm{Adv}_{\mathcal{D}_{\text{ODP}},\mathcal{D}_{\mathcal{M}}}\left(\mathcal{M}'^{\mathcal{A}}\right) + \epsilon'_{\mathcal{A}} \tag{3.4}$$

In our last step, we limit the distinguishing advantage of $\mathcal{M}'^{\mathcal{A}}$. We use a hybrid argument to show that this advantage is bound by the undetectability of $\mathcal{F}_n$. For a given $\alpha \in \{0, \ldots, \lambda\}$, we define the hybrids

$$H_j = (\alpha, \mathcal{E}^{j,\alpha}_{\mathsf{ck}}(\mathcal{U}_n), \mathbf{R}, K),$$

with $\mathbf{R} \xleftarrow{\$} \{0,1\}^{n\times\lambda}, K \xleftarrow{\$} \{0,1\}^k$ for $0 \leq j \leq \alpha$. Given an adversary $\mathcal{B}$ that can distinguish between H_0 and H_α with advantage $\epsilon_{\mathcal{B}}$, a hybrid argument yields that there must exist two consecutive hybrids that $\mathcal{B}$ distinguishes with advantage $\geq \epsilon_{\mathcal{B}}/\alpha$. Assume these two hybrids are H_γ and $H_{\gamma+1}$. Then we can construct an oracle machine $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ that uses $\mathcal{B}$ to distinguish between $\mathcal{D}_{\text{UD},\mathcal{U}}$ and $\mathcal{D}_{\text{UD},\mathcal{F}}$ as defined in the preliminaries and thereby attacking the undetectability of $\mathcal{F}_n$. Given a distinguishing challenge (U, K), $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ selects $\mathbf{R} \xleftarrow{\$} \{0,1\}^{n\times\lambda}$, sets $\mathsf{ck} = (K, \mathbf{R})$, computes $X = \mathcal{E}^{\gamma+1,\alpha}_{\mathsf{ck}}(U)$, runs $b \longleftarrow \mathcal{B}(\alpha, X, \mathbf{R}, K)$ and outputs b.

Let's analyze the advantage $\mathrm{Adv}^{\text{UD}}_{\mathcal{F}_n}\left(\mathcal{M}^{\mathcal{B}}_{\text{UD}}\right)$ of $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$. If the sample is taken from $\mathcal{D}_{\text{UD},\mathcal{U}}$, U is uniformly random and $X = \mathcal{E}^{\gamma+1,\alpha}_{\mathsf{ck}}(U)$ is distributed exactly like the second element of $H_{\gamma+1}$. Otherwise, if the sample is taken from $\mathcal{D}_{\text{UD},\mathcal{F}}$, then $U \longleftarrow \mathrm{F}_K(\mathcal{U}_n)$ is an output of F_K and we get

$$\begin{aligned} X &= \mathcal{E}^{\gamma+1,\alpha}_{\mathsf{ck}}(\mathrm{F}_K(\mathcal{U}_n)) = \mathcal{E}^{\gamma,\alpha}_{\mathsf{ck}}(\mathcal{U}_n \oplus R_{\gamma+1}) \\ &= \mathcal{E}^{\gamma,\alpha}_{\mathsf{ck}}(\mathcal{U}_n) = H_{\gamma_{(2)}} \end{aligned}$$

where $H_{\gamma_{(2)}}$ denotes the second element of H_γ. Here we used the fact, that the xor of a uniformly distributed variable and a fixed value leads again to a uniformly distributed variable. Summing up, the input of $\mathcal{B}$, produced by $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ is either distributed like H_γ or like $H_{\gamma+1}$, depending on $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$'s distinguishing challenge. Hence, the advantage of $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ is exactly that of $\mathcal{B}$ distinguishing between these two hybrids. Thus, we get

$$\mathrm{Adv}^{\text{UD}}_{\mathcal{F}_n}\left(\mathcal{M}^{\mathcal{B}}_{\text{UD}}\right) \geq \epsilon_{\mathcal{B}}/\alpha.$$

As the advantage of $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ is bounded by the undetectability of $\mathcal{F}_n$ per assumption, $\mathcal{M}'^{\mathcal{A}}$ does exactly what we assume $\mathcal{B}$ to do and the runtime of $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ is that of $\mathcal{B}$ plus at most λ evaluations of $\mathcal{E}$, we get

$$\mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right) \geq \mathrm{Adv}^{\text{UD}}_{\mathcal{F}_n}\left(\mathcal{M}^{\mathcal{B}}_{\text{UD}}\right) \geq \frac{\epsilon_{\mathcal{B}}}{\alpha} = \frac{\mathrm{Adv}_{\mathcal{D}_{\text{ODP}},\mathcal{D}_{\mathcal{M}}}\left(\mathcal{M}'^{\mathcal{A}}\right)}{\alpha}$$

where $t^\star = t'' + \lambda - 1 = t + 2\lambda - 1$ is the runtime of $\mathcal{M}_{\text{UD}}^{\mathcal{B}}$. As $\alpha \in \{0, \dots, \lambda\}$, we obtain the following bound on the advantage of $\mathcal{M}'^{\mathcal{A}}$:

$$\mathrm{Adv}_{\mathcal{D}_{\text{ODP}}, \mathcal{D}_{\mathcal{M}}}\left(\mathcal{M}'^{\mathcal{A}}\right) \leq \lambda \cdot \mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right). \tag{3.5}$$

Putting equations (3.3), (3.4) and (3.5) together we obtain a final bound on $\epsilon_{\mathcal{A}}$ for case $c1$:

$$\epsilon_{\mathcal{A}} \leq \lambda \cdot \mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right) + \mathrm{InSec}^{\text{OW}}\left(\mathcal{F}_n; t'\right)$$

with $t' = t + \lambda$ and $t^\star = t + 2\lambda - 1$.

The success probability of $\mathcal{M}_{\text{SPR}}^{\mathcal{A}}$ conditioned on case $c2$ is exactly the bound from the SO resistance proof, because case $c2$ implies that the returned one-deeper preimage leads directly to a second origin. So, we get for case $c2$

$$\epsilon_{\mathcal{A}} \leq \lambda \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{F}_n; t''\right)$$

where the time $t'' = t + 2\lambda$ is an upper bound for the runtime of $\mathcal{A}$ plus the time needed to compute all values for two chains of length $\leq \lambda$.

As the two cases are mutually exclusive and the probabilities that they occur add up to 1, we can apply a union bound and get

$$\epsilon_{\mathcal{A}} \leq \lambda \cdot \mathrm{InSec}^{\text{UD}}\left(\mathcal{F}_n; t^\star\right) + \mathrm{InSec}^{\text{OW}}\left(\mathcal{F}_n; t'\right) + \lambda \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{F}_n; t''\right)$$

with $t' = t + \lambda$, $t'' = t + 2\lambda$ and $t^\star = t + 2\lambda - 1$. This contradicts the initial assumption □

3.4 A Function Chain using Pseudorandom Function Families

In this section we present $\mathcal{C}^{\$}$, a function chain that is ODP resistant if the used function family is pseudorandom. Using this function chain we obtain a W-OTS variant W-OTS$^{\$}$ that is EU-CMA secure if the used function family is pseudorandom. As the existence of pseudorandom function families is known to be equivalent to the existence of a one-way function [HILL99, GGM86], this means that W-OTS$^{\$}$ has minimal security assumptions. We first present the function chain $\mathcal{C}^{\$}$. Afterwards we state some preliminaries to finally prove $\mathcal{C}^{\$}$ to be ODP resistant.

3.4.1 The $\mathcal{C}^{\$}$ Function Chain

We now describe the function chain $\mathcal{C}^{\$}$. For a given security parameter n it uses a family of functions

$$\mathcal{F}_n = \{\mathrm{F}_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^n\} \tag{3.6}$$

parameterized by a key $K \in \{0,1\}^n$ and a security parameter n. The function chain $\mathcal{C}^{\$}$ works over domain and key space $\mathcal{D} = \mathcal{K} = \{0,1\}^n$. The algorithms are instantiated as follows:

$\mathcal{I}(1^n, \lambda)$: On input of the security parameter n in unary and the chain length λ the initialization algorithm choses a value $R \xleftarrow{\$} \{0,1\}^n$ uniformly at random and returns $\mathsf{ck} = R$. We assume that λ is publicly known.

$\mathcal{E}^{i,j}_{\mathsf{ck}}(X)$: On input of value $X \in \mathcal{D}$, interval $0 \leq i \leq j \leq \lambda$ and chain key ck the evaluation function works the following way. According to the definition of a function chain, $\mathcal{E}$ returns X if $i = j$ ($\mathcal{E}^{i,i}_{\mathsf{ck}}(X) = X$). For $i < j$ we define $\mathcal{E}$ recursively by

$$\mathcal{E}^{i,j}_{\mathsf{ck}}(X) = \mathrm{F}_{\mathcal{E}^{i,j-1}_{\mathsf{ck}}(X)}(R),$$

i.e. in every round, the output of the previous round is used to select a function from $\mathcal{F}_n$ which then is evaluated on input R. One might think of it as a random walk through the function family starting from X.

Correctness follows immediately from the iterative nature of the evaluation algorithm.

3.4.2 Preliminaries

In the following we define a new security notion for function families required for our reduction. We call the notion *key one-wayness* (KOW) which states that it is hard to find a key K such that the function F_K maps a given input X to a given output Y. We also state two lemmas about the relation between this notion and the pseudorandomness property, which will be useful for the security proof.

Towards defining KOW, we define the success probability of an adversary $\mathcal{A}$ against the key one-wayness of a function family $\mathcal{F}_n$ as

$$\begin{aligned}\mathrm{Succ}^{\mathrm{KOW}}_{\mathcal{F}_n}(\mathcal{A}) = \mathsf{Pr}\Big[&(X, K) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^n, Y \leftarrow \mathrm{F}_K(X),\\ &K' \longleftarrow \mathcal{A}(X, Y) : Y = \mathrm{F}_{K'}(X)\Big].\end{aligned}$$

Based on this we can define KOW:

Definition 3.10 (Key One-Wayness (KOW))**.** *Let $n, t \in \mathbb{N}$, $t = \operatorname{poly}(n)$, $\mathcal{F}_n$ be a family of functions as in (3.6). We call $\mathcal{F}_n$* KOW*, if* $\operatorname{InSec}^{\text{KOW}}(\mathcal{F}_n; t)$*, the success probability of any adversary $\mathcal{A}$ that runs in time at most t, is negligible in n:*

$$\operatorname{InSec}^{\text{KOW}}(\mathcal{F}_n; t) \overset{def}{=} \max_{\mathcal{A}} \left\{ \operatorname{Succ}^{\text{KOW}}_{\mathcal{F}_n}(\mathcal{A}) \right\} = \operatorname{negl}(n)\,.$$

A key collision of a function family $\mathcal{F}_n$ is defined as a pair of distinct keys (K, K') such that $\mathrm{F}_K(X) = \mathrm{F}_{K'}(X)$ holds for some $X \in \{0,1\}^n$. In our proofs we make use of an upper (κ) and a lower (κ') bound on the number of key collisions that occur in the family $\mathcal{F}_n$. We define these bounds as follows:

Definition 3.11. *Let $\mathcal{F}_n$ be a family of functions as in (3.6). We define the upper bound on the number of key collisions in $\mathcal{F}_n$ as the maximum number of keys that map the same input value to the same output value:*

$$\kappa(\mathcal{F}_n) \overset{def}{=} \max_{S \subseteq \{0,1\}^n} \Big\{ |S| \mid (\exists X, Y \in \{0,1\}^n), (\forall K \in S) : \mathrm{F}_K(X) = Y \Big\}.$$

We define the lower bound on the number of key collisions in $\mathcal{F}_n$ accordingly as

$$\kappa'(\mathcal{F}_n) \overset{def}{=} \min_{K' \in \{0,1\}^n} \max_{S \subseteq \{0,1\}^n, K' \in S} \Big\{ |S| \mid (\exists X, Y \in \{0,1\}^n), (\forall K \in S) : \mathrm{F}_K(X) = Y \Big\}.$$

Please note that $\kappa, \kappa' \geq 1$ per definition. We write κ (κ') instead of $\kappa(\mathcal{F}_n)$ $(\kappa'(\mathcal{F}_n))$ where $\mathcal{F}_n$ is clear from the context. The values κ and κ' restrict the number of different images Y some preimage X can be mapped to by functions in $\mathcal{F}_n$, i.e.

$$\frac{2^n}{\kappa} \leq \Big| \{ \mathrm{F}_K(X) : K \in \{0,1\}^n \} \Big| \leq \frac{2^n}{\kappa'} \tag{3.7}$$

for all $X \in \{0,1\}^n$. Also, given $Y \overset{\$}{\longleftarrow} \{0,1\}^n$ the probability that there exists a key K and preimage X such that $\mathrm{F}_K(X) = Y$ holds is at least $1/\kappa$.

To make our security proof meaningful we will need a bound on κ. The following lemma states a relation between κ and the insecurity of a pseudorandom function family. Please recall, that the time t is counted in terms of evaluations of F. We assume, that a call to the oracle Box in the PRF game takes the same time as an evaluation of F.

Lemma 3.12. *Let $b, n \in \mathbb{N}, b \leq n$, $\mathcal{F}_n$ as in (3.6) be a* PRF *with insecurity function*

$$\operatorname{InSec}^{\text{PRF}}(\mathcal{F}_n; t) \leq \frac{t}{2^b}$$

and $\kappa(\mathcal{F}_n)$ as in Definition 3.11. Then

$$\kappa(\mathcal{F}_n) \leq 2^{n-b} + 1.$$

Proof. Assume $\kappa > 2^{n-b}+1$ and let (X, Y) be a pair where there exist κ keys mapping X to Y. We show how to build an adversary $\mathcal{A}$ against the pseudorandomness of $\mathcal{F}_n$. First, $\mathcal{A}$ queries Box with X. If $\mathsf{Box}(X) = Y$ then $\mathcal{A}$ returns 1 and 0 otherwise. Clearly $\mathcal{A}$ runs in time $t = 1$. Furthermore, we have $\mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{F}_n : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1] = \kappa/2^n > 2^{-b} + 2^{-n}$ and $\mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{AF} : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1] = 2^{-n}$ and therefore $\mathrm{Succ}^{\mathrm{PRF}}_{\mathcal{F}_n}(\mathcal{A}) > 2^{-b}$ which contradicts the assumption on the insecurity of $\mathcal{F}_n$. □

Following the definition of κ and κ', $\kappa' \geq 1$ always holds. The above lemma implies that for a good pseudorandom function family, i.e. a pseudorandom function family with $b = n$ bit security, $\kappa = 2$.

The following lemma states that the KOW property is implied by the PRF property. In other words, an efficient attacker against the KOW property leads to an efficient attacker against the pseudorandomness.

Proposition 3.13 (PRF ⇒ KOW). *Let $\mathcal{F}_n$ be a function family as in Equation 3.6. Then the pseudorandomness and the key one-wayness of $\mathcal{F}_n$ are related as follows:*

$$\mathrm{InSec}^{\mathrm{KOW}}(\mathcal{F}_n; t) \leq \frac{1}{\frac{1}{\kappa} - \frac{1}{2^n}} \cdot \mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; t+2, 2).$$

Proof. Assume there exists an adversary $\mathcal{A}$ against the key one-wayness of $\mathcal{F}_n$, i.e. that given a pair $(X, Y) \in \{0,1\}^n \times \{0,1\}^n$ finds a key K satisfying $Y = \mathrm{F}_K(X)$ in time t with probability $\epsilon = \mathrm{InSec}^{\mathrm{KOW}}(\mathcal{F}_n; t)$. Then we can construct an oracle machine $\mathcal{M}^{\mathcal{A}}$ against the pseudorandomness of $\mathcal{F}_n$ using $\mathcal{A}$ the following way: $\mathcal{M}^{\mathcal{A}}$ queries $\mathsf{Box}(\cdot)$ with a random value $X \xleftarrow{\$} \{0,1\}^n$. After receiving the answer Y, $\mathcal{M}^{\mathcal{A}}$ runs $K \longleftarrow \mathcal{A}(X, Y)$ to obtain a key K. Then $\mathcal{M}^{\mathcal{A}}$ queries Box with a second random value $X' \xleftarrow{\$} \{0,1\}^n$. $\mathcal{M}^{\mathcal{A}}$ returns 1 if $\mathsf{Box}(X') = Y' = \mathrm{F}_K(X')$ and 0 otherwise.

We now analyze the success probability of $\mathcal{M}^{\mathcal{A}}$. In case $\mathsf{Box} \xleftarrow{\$} \mathcal{F}_n$, the probability that $\mathcal{A}$ outputs a key K such that $\mathrm{F}_K(X) = Y$ holds is ϵ per assumption. The probability that $\mathsf{Box}(X') = Y' = \mathrm{F}_K(X')$ holds is at least $1/\kappa$, because at least one of the κ functions in $\mathcal{F}_n$ mapping X to Y also maps X' to Y'. In case $\mathsf{Box} \xleftarrow{\$} \mathcal{AF}(n, n)$, the probability that $\mathcal{A}$ outputs a key K such that $\mathrm{F}_K(X) = Y$ holds is at most ϵ. The probability that $\mathsf{Box}(X') = Y' = \mathrm{F}_K(X')$ holds is $1/2^n$, because from the $2^{n(2^n-1)}$ functions in $\mathcal{AF}(n, n)$ mapping X to Y, only $2^{n(2^n-2)}$ also map X' to Y'. In summary we get $\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; t+2, 2) \geq \mathrm{Succ}^{\mathrm{PRF}}_{\mathcal{F}_n}\left(\mathcal{M}^{\mathcal{A}}\right) \geq \epsilon\,(1/\kappa - 1/2^n)$. □

3.4.3 Security Proof

We now proof that $\mathcal{C}^{\$}$ is an ODP resistant function chain, if $\mathcal{F}_n$ is a pseudorandom function family. More specifically, we prove the following lemma:

Lemma 3.14. *Let $n, \lambda, t \in \mathbb{N}$, $\mathcal{F}_n$ as in Equation (3.6) a pseudorandom function family. Then,* $\mathrm{InSec}^{\mathrm{ODP}}\left(\mathcal{C}^{\$}(1^n, \lambda); t\right)$, *the* ODP *resistance of* $\mathcal{C}^{\$}$ *is bounded by*

$$\mathrm{InSec}^{\mathrm{ODP}}\left(\mathcal{C}^{\$}(1^n, \lambda); t\right) \leq \frac{\kappa(\mathcal{F}_n)^{\lambda}}{\left(\frac{1}{\kappa(\mathcal{F}_n)} - \frac{1}{2^n}\right)} \cdot \mathrm{InSec}^{\mathrm{PRF}}\left(\mathcal{F}_n; t', 2\right)$$

with $t' = t + \lambda + 2$.

This leads to our second main result. We call W-OTS using $\mathcal{C}^{\$}$ as function chain W-OTS$^{\$}$. Combining the above result with Lemma 3.5 we obtain the following theorem on the security of W-OTS$^{\$}$:

Theorem 3.15. *Let $n, w, m, t \in \mathbb{N}$, $w, m = \mathrm{poly}(n)$, $\mathcal{F}_n$ as in Equation (3.6) a pseudorandom function family. Then, the insecurity of W-OTS$^{\$}$ against an* EU-CMA *attack,* $\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}^{\$}(1^n, w, m); t, 1\right)$, *is bounded by*

$$\begin{aligned}&\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{W-OTS}^{\$}(1^n, w, m); t, 1\right)\\ &\qquad \leq \ell w \kappa(\mathcal{F}_n)^{w-1} \frac{1}{\left(\frac{1}{\kappa(\mathcal{F}_n)} - \frac{1}{2^n}\right)} \cdot \mathrm{InSec}^{\mathrm{PRF}}\left(\mathcal{F}_n; t', 2\right)\end{aligned}$$

with $t' = t + t_{\mathsf{Kg}} + t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + w + 1$.

The intuition behind the proof is that a one-deeper preimage for challenge (ck, i, Y) will lead a function key for $\mathcal{F}_n$ used in iteration $i - 1$. Using a combinatorial argument, this key maps R to Y with high enough probability. Thus, we can use an adversary against the ODP resistance of $\mathcal{C}^{\$}$ to construct a KOW adversary. Using Proposition 3.13 we can limit the maximum success probability of this KOW adversary by the insecurity of the pseudorandomness of $\mathcal{F}_n$.

Proof. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ that can break the ODP resistance of $\mathcal{C}^{\$}(1^n, \lambda)$ running in time $\leq t$ and with success probability $\epsilon_{\mathcal{A}} = \mathrm{Succ}^{\mathrm{ODP}}_{\mathcal{C}+(1^n,\lambda)}(\mathcal{A})$ greater than the claimed bound on $\mathrm{InSec}^{\mathrm{ODP}}\left(\mathcal{C}^{\$}(1^n, \lambda); t\right)$. We first show how to construct an oracle machine $\mathcal{M}^{\mathcal{A}}$ that breaks the key one-wayness of $\mathcal{F}_n$ using $\mathcal{A}$. A pseudo-code description of $\mathcal{M}^{\mathcal{A}}$ is given as Algorithm 3.4.

The goal of $\mathcal{M}^{\mathcal{A}}$ is to produce a key K such that $\mathrm{F}_K(X_c) = Y_c$ for X_c, Y_c provided as input. First, $\mathcal{M}^{\mathcal{A}}$ sets X_c to be the chain key. Then $\mathcal{M}^{\mathcal{A}}$ chooses a random index α within the range $[1, \ldots, \lambda]$ using the uniform distribution. Next, $\mathcal{M}^{\mathcal{A}}$ calls $\mathcal{A}(\mathsf{ck}, \alpha, Y_c)$ claiming that Y_c is an αth intermediate value of the chain (Line 3). If $\mathcal{A}$ returns a one-deeper preimage (j, X'), $\mathcal{M}^{\mathcal{A}}$ computes a candidate key $K \leftarrow$

Algorithm 3.4: $\mathcal{M}^{\mathcal{A}}$

```
   Input: Security parameter $n$, KOW challenge $(X_c, Y_c)$ as in Definition 3.10
   Output: $K$, such that $\mathrm{F}_K(X_c) = Y_c$ or fail
 1 Set $\mathsf{ck} = X_c$;
 2 Choose index $\alpha \in [1, \dots, \lambda]$ uniformly at random;
 3 Run $(j, X') \longleftarrow \mathcal{A}(\mathsf{ck}, \alpha, Y_c)$;
 4 if $j < \alpha$ and $\mathcal{E}_{\mathsf{ck}}^{\alpha,\lambda}(Y_c) = \mathcal{E}_{\mathsf{ck}}^{j,\lambda}(X')$ then
 5     Compute $K \leftarrow \mathcal{E}_{\mathsf{ck}}^{j,\alpha-1}(X')$;
 6     if $\mathrm{F}_K(X) \neq Y$ then
 7         return fail;
 8     end
 9     return $K$;
10 end
11 return fail;
```

$\mathcal{E}_{\mathsf{ck}}^{j,\alpha-1}(X')$ as the output of iteration $\alpha - 1$, continuing the chain from X'. Then, $\mathcal{M}^{\mathcal{A}}$ checks whether $\mathrm{F}_K(X) = Y_c$ holds (Line 6). If the condition holds, $\mathcal{M}^{\mathcal{A}}$ returns the key K. Otherwise $\mathcal{M}^{\mathcal{A}}$ returns fail.

We now compute the success probability of $\mathcal{M}^{\mathcal{A}}$. The probability that $\mathcal{A}$ succeeds in Line 3 is at least $\epsilon_{\mathcal{A}}$ by definition. This probability holds under the condition that Y_c resembles a regular output of the αth iteration of the chain under ck. This is the case if there exists a value W such that $\mathcal{E}_{\mathsf{ck}}^{\alpha}(W) = Y_c$. This happens with probability at least $1/\kappa^{\alpha}$ according to Definition 3.11. The probability that $Y_c = \mathrm{F}_K(X)$ holds in Line 6 is at least $1/\kappa^{\lambda-\alpha}$. This is because there exist at most $\kappa^{\lambda-\alpha}$ different values that are mapped to $\mathcal{E}_{\mathsf{ck}}^{\alpha,\lambda}(Y_c)$ after $\lambda - \alpha$ iterations of the chain and at least one of them is Y_c.

In summary we have $\mathrm{Succ}_{\mathcal{F}_n}^{\mathrm{KOW}}\left(\mathcal{M}^{\mathcal{A}}\right) \geq \epsilon_{\mathcal{A}}/(\kappa^{\alpha}\kappa^{\lambda-\alpha}) = \epsilon_{\mathcal{A}}/(\kappa^{\lambda})$ and $\mathcal{M}^{\mathcal{A}}$ runs in time $t' \leq t + \lambda$ as $\mathcal{M}^{\mathcal{A}}$ has to compute λ intermediate values of the chain, at most. We can use Proposition 3.13 and the pseudorandomness of $\mathcal{F}_n$ to limit $\mathcal{M}^{\mathcal{A}}$'s success probability:

$$\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; t'', 2) \geq \epsilon_{\mathcal{A}}(1/\kappa - 1/2^n)/(\kappa^{\lambda})$$

with $t'' = t + \lambda + 2$ which states the required contradiction. □

4 XMSS

In this chapter we present our main construction: An efficient forward secure signature scheme using minimal security assumptions. We call the scheme eXtended Merkle Signature Scheme (XMSS). The section is structured as follows. First, we give a description of an EU-CMA secure construction in Section 4.1 and prove its security in Section 4.2. In Section 4.3 we show how to change the construction to achieve forward security and give a proof that this property is indeed achieved. We finally analyze the theoretical performance of the scheme in Section 4.4, presenting theoretical runtimes and sizes. The contributions of this chapter were published as parts of [2].

4.1 The eXtended Merkle Signature Scheme XMSS

In this section we describe the basic construction of XMSS. This construction achieves standard security. The forward secure version differs only in few details. The construction is based on the concept of a Merkle signature scheme, as described in Section 2.2.2. To minimize storage requirements, pseudorandom key generation is used to generate the OTS secret keys on the fly. As OTS we use one of our W-OTS constructions. In the following we describe the scheme for the case of W-OTS$^\$$. We explicitly comment on differences that occur using W-OTS$^+$ in the end. However, whenever a description holds for W-OTS in general, we write W-OTS. Only if something is specific for using W-OTS$^\$$ we explicitly write W-OTS$^\$$. As tree traversal algorithm we use the BDS algorithm from Section 2.2.3. We start the description with the parameters used by XMSS, afterwards we give a description of the building blocks, namely the XMSS Tree, the leaf construction, and the pseudorandom key generation. Then we describe the algorithms for key generation, signature generation and verification.

Parameters. For security parameter $n \in \mathbb{N}$, XMSS uses a function family $\mathcal{F}_n = \{\mathrm{F}_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$, and a hash function H, chosen uniformly

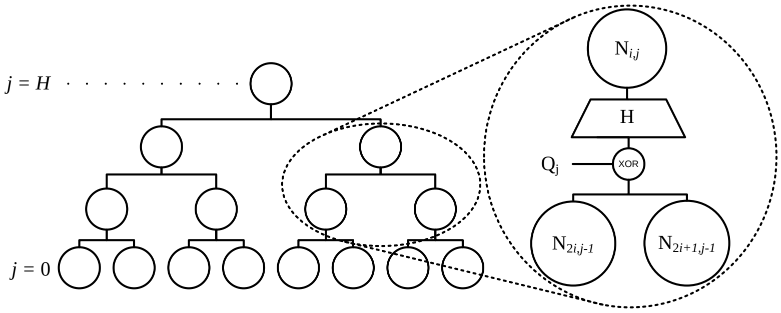


Figure 4.1: The XMSS tree construction

at random from the family $\mathcal{H}_n = \{\mathrm{H}_K : \{0,1\}^{2n} \to \{0,1\}^n | K \in \{0,1\}^n\}$. It is parameterized by the binary message length $m \in \mathbb{N}$, the tree height $h \in \mathbb{N}$, the BDS parameter $k \in \mathbb{N}, k < h, k - h$ is even, and the Winternitz parameter $w \in \mathbb{N}, w > 1$. XMSS can be used to sign 2^h messages of m bits. Those parameters are publicly known.

XMSS Tree. The XMSS tree is a modification of the classical Merkle Hash Tree proposed in [DOTV08]. Figure 4.1 shows the construction. The XMSS tree is a binary tree of height h that makes use of the hash function H. It has $h+1$ levels. The leaves are on level 0. The root is on level h. The nodes on level j, $0 \leq j \leq h$, are denoted by $N_{i,j}$, $0 \leq i < 2^{h-j}$. To construct the tree, h bit masks $Q_j \in \{0,1\}^{2n}$, $0 < j \leq h$, are used. $N_{i,j}$, for $0 < j \leq h$, is computed as

$$N_{i,j} = \mathrm{H}((N_{2i,j-1} || N_{2i+1,j-1}) \oplus Q_j).$$

The usage of the bitmasks is the main difference to the other Merkle tree constructions. It is borrowed from [BR97] and allows to replace the collision resistant hash function family by a second-preimage resistant one.

Leaf Construction. The leaves of the XMSS tree are the hash values of the W-OTS public keys. More specifically, they are the hash values of $(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$. The chain key $\mathsf{ck} = \mathsf{pk}_0$ becomes part of the XMSS public key. To avoid the need of a collision resistant hash function, another XMSS tree is used to construct the leaves. It is called L-tree. The ℓ leaves of an L-tree are the ℓ bit strings $(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$ from the corresponding verification key. As ℓ is not necessarily a power of 2, there might not be sufficiently many leaves to get a complete binary tree. Therefore the construction is modified. A left node that has no right sibling is lifted to a higher level of the L-tree until it becomes the right sibling of another node. In this construction, the same hash function H as above but new bitmasks are used. The bitmasks are the same for all L-trees. As L-trees have height $\lceil \log \ell \rceil$, additional $\lceil \log \ell \rceil$ bitmasks are required.

Pseudorandom Key Generation. To reduce the secret key size, the W-OTS key pairs are generated using pseudorandom generators (PRG). The way this is done is the difference between the EU-CMA and the forward-secure XMSS construction. For the EU-CMA secure construction, the pseudorandom key generation allows to reduce the secret key size from 2^h W-OTS secret keys of ℓn bits each to a single n bit seed value (ignoring the BDS state).

The W-OTS secret keys are computed using a seed $\textsc{Seed} \in \{0,1\}^n$, the pseudorandom function family $\mathcal{F}_n$, and the pseudorandom generator GEN which for $\lambda \in \mathbb{N}, \mu \in \{0,1\}^n$ yields

$$\mathsf{GEN}_\lambda(\mu) = \mathrm{F}_\mu(1)||\ldots||\mathrm{F}_\mu(\lambda).$$

For $i \in \{1,\ldots,2^h\}$ the i-th W-OTS secret key is computed as

$$\mathsf{sk}_i \longleftarrow \mathsf{GEN}_\ell(\mathrm{F}_{\textsc{Seed}}(i)).$$

Key Generation Algorithm. The key generation algorithm takes as input all of the above parameters. Then the whole XMSS Tree has to be constructed to obtain the value of the root node. We now detail this procedure. First, the bitmasks $(Q_1,\ldots,Q_{h+\lceil \log \ell \rceil})$ and the value R for the W-OTS$^\$$ chain key are chosen uniformly at random. Then, $\textsc{Seed}$ is chosen uniformly at random and it is stored as part of the secret key SK. The tree is constructed using the $\textsc{TreeHash}$ algorithm, described in Section 2.2.3. The method $\textsc{LeafCalc}$ to compute the leafs of the tree is implemented the following way. Given index ϕ, the W-OTS secret key sk_ϕ is generated as $\mathsf{sk}_\phi \longleftarrow \mathsf{GEN}_\ell(\mathrm{F}_{\textsc{Seed}}(\phi))$. Next, the W-OTS key generation is used to compute the W-OTS public key, which in turn is used to compute the corresponding leaf using an L-tree. In the end, the W-OTS key pair is deleted. After all 2^h leaves were processed by $\textsc{TreeHash}$, the only value on Stack is the root of the tree, which is stored in the public key PK.

During root computation, the BDS state $\mathsf{State}_{\mathrm{BDS}}$ is initialized. The initial XMSS secret key is $\mathsf{SK} = (\textsc{Seed}, \mathsf{State}_{\mathrm{BDS}})$. The XMSS public key consists of the bitmasks $(Q_1,\ldots,Q_{h+\lceil \log \ell \rceil})$, the value X, and the root of the tree.

Signature Generation Algorithm. The signature generation algorithm takes as input a message M of binary length m, the secret key SK and the index i. It outputs an updated secret key SK' and a signature Σ on the message M. To sign the ith message (we start counting from 0), the ith W-OTS key pair is used. The signature $\Sigma = (i, \sigma, \mathsf{Auth})$ contains the index i, the W-OTS signature σ, and the authentication path for the leaf $N_{0,i}$. We now explain how a signature is generated. On input of the ith message, the ith W-OTS secret key sk_i is pseudorandomly generated as described above. Next, sk_i is used to generate the one-time signature σ on M. Then the authentication path is computed using BDS, which results in a changed $\mathsf{State}'_{\mathrm{BDS}}$. The updated secret key SK' contains $\textsc{Seed}$ and $\mathsf{State}'_{\mathrm{BDS}}$.

Signature Verification Algorithm. The signature verification algorithm takes as input a signature $\Sigma = (i, \sigma, \mathsf{Auth})$, a message $M \in \{0,1\}^m$ and the XMSS public key PK. To verify the signature, the values $B = (b_1, \ldots, b_\ell)$ are computed as described in the W-OTS signature generation, using M. Then the ith verification key is computed as in W-OTS signature verification:

$$(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (\mathcal{E}^{b_1, w-1}(\sigma_1), \ldots, \mathcal{E}^{b_\ell, w-1}(\sigma_\ell)).$$

The corresponding leaf $N_{0,i}$ of the XMSS tree is constructed using an L-tree. This leaf and the authentication path are used to compute the path $(P_0, \ldots, P_h)$ to the root of the XMSS tree, where $P_0 = N_{0,i}$ and

$$P_j = \begin{cases} \mathrm{H}((P_{j-1} || \mathsf{Auth}_{j-1}) \oplus B_j), & \text{if } \lfloor i/2^j \rfloor \equiv 0 \mod 2 \\ \mathrm{H}((\mathsf{Auth}_{j-1} || P_{j-1}) \oplus B_j), & \text{if } \lfloor i/2^j \rfloor \equiv 1 \mod 2 \end{cases}$$

for $0 \leq j \leq h$. If P_h is equal to the root of the XMSS tree given in the public key, the signature is accepted. Otherwise, it is rejected.

Using W-OTS$^+$. As mentioned above, things slightly change when we use W-OTS$^+$ instead of W-OTS$^\$$. However, there are only three differences. First, we need another function F' randomly chosen from a family $\mathcal{F}'_n : \{\mathrm{F}_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^k\}$ as described in Section 3.3 using the uniform distribution. Second, the W-OTS algorithms use the $\mathcal{C}^+$ algorithms instead of the $\mathcal{C}^\$$ algorithms. Third, the public key changes. As we assume F' to be publicly known, we do not have to store the corresponding key but we need the $w-1$ randomization elements $\mathbf{R}$. We can reuse the bitmasks that are already part of the secret key. As the bitmasks are bit strings of length $2n$, this works fine if $w - 1 \leq 2(h + \lceil \log \ell \rceil)$. Otherwise, we reuse the bit masks for the first $2(h + \lceil \log \ell \rceil)$ randomization elements and add $w - 1 - 2(h + \lceil \log \ell \rceil)$ uniformly random bit strings to the public key.

4.2 Standard Security

In this section we prove that XMSS as described in the last section achieves EU-CMA security in the standard model and discuss the minimality of the assumptions we use. We first provide the needed preliminaries.

4.2.1 Preliminaries

We now present some security notions that we need for our proofs. We defined signature schemes and EU-CMA security in Section 2.1.2. Here we extend this definitions to the case of key evolving signature schemes. This is required to formally cover MSS type signature schemes. We also recall the notion of a pseudorandom generator. Please note that in our proofs we measure algorithmic runtimes as the number of evaluations of functions from $\mathcal{F}_n$ and $\mathcal{H}_n$.

Key Evolving Signature Schemes XMSS is a stateful signature scheme. This is not covered by the standard definition of digital signature schemes. To capture this formally we follow the definition from [BM99] of key evolving signature schemes. In a key evolving signature scheme, the lifetime of a keypair is divided into several time periods, say p. While the public key pk is fixed, the scheme operates on p different secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{p-1}$, one per time period. A key evolving signature scheme contains a key update algorithm that is called at the end of each time period and updates the secret key. The end of a time period might be determined by time, i.e. a period is one day or something else, like the maximum number of signatures a secret key can be used for. The latter is the case for XMSS, where a period ends after signing one message and the key update algorithm is automatically called after each signature generation. In contrast to an ordinary signature scheme, the key generation algorithm of a key evolving signature scheme takes as an additional input the maximal number of periods p and outputs the public key pk and the first secret key sk_0. Using a key evolving signature scheme, a signature (σ, i) on a message contains the index i of the period of the used secret key. The validation of a signature (σ, i) only succeeds, if the signature is a valid signature for time period i under public key pk. We summarize this in the following more formal definition.

Definition 4.1 (Key Evolving Signature Scheme)**.** *A key evolving signature scheme is a quadruple of probabilistic polynomial time algorithms* $\textsc{Kes} = (\mathsf{Kg}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vf})$*. It is parameterized by a security parameter* $n \in \mathbb{N}$ *and the number of time periods* $p \in \mathbb{N}, p = \text{poly}(n)$ *and operates on the following finite sets with description length polynomial in* n*: The secret key space* $\mathcal{KS} = \mathcal{KS}_0 \times \ldots \times \mathcal{KS}_{p-1}$ *consisting of*

p sets, the public key space $\mathcal{KP}$, the message space $\mathcal{M}$, and the signature space Σ. The runtime of the algorithms is polynomial in n and the algorithms are defined as follows:

> $\mathsf{Kg}(1^n, p)$*: The key generation algorithm on input of the security parameter $n \in \mathbb{N}$ in unary and the number of time periods $p \in \mathbb{N}$ outputs an initial private signing key $\mathsf{sk}_0 \in \mathcal{KS}_0$ and a public verification key $\mathsf{pk} \in \mathcal{KP}$.*
>
> $\mathsf{KUpd}(\mathsf{sk}, i)$*: The key update algorithm on input of an index $i \in \mathbb{N}$ and a secret signing key $\mathsf{sk} \in \mathcal{KS}$ outputs the private signing key $\mathsf{sk}' \in \mathcal{KS}_{i+1}$ for the next time period if $i < p - 1$ and $\mathsf{sk} \in \mathcal{KS}_i$. If $i \geq p - 1$ it outputs the empty string. In all other cases it returns* fail.
>
> $\mathsf{Sign}(\mathsf{sk}, M, i)$*: The signature algorithm on input of a signature key $\mathsf{sk} \in \mathcal{KS}$, a message $M \in \mathcal{M}$, and an index $i \in \mathbb{N}$ outputs the signature $(\sigma, i) \in \Sigma$ of the message M if $i < T$ and $\mathsf{sk} \in \mathcal{KS}_i$. It returns* fail*, otherwise.*
>
> $\mathsf{Vf}(\mathsf{pk}, M, (\sigma, i))$*: The verification algorithm on input of a public key $\mathsf{pk} \in \mathcal{KP}$, a message $M \in \mathcal{M}$, and a signature $(\sigma, i) \in \Sigma$ outputs 1 iff (σ, i) is a valid signature on M under public key pk for time period i and 0 otherwise.*

Now, let $\mathsf{KUpd}(\mathsf{sk}_0)^i = \mathsf{KUpd}(\ldots \mathsf{KUpd}(\mathsf{sk}_0, 0) \ldots, i-1)$ denote the computation of the key for time period i starting from sk_0. The following condition must hold: For all $M \in \mathcal{M}$, $(\mathsf{pk}, \mathsf{sk}_0) \longleftarrow \mathsf{Kg}(1^n, p)$, and $i < p$: $\mathsf{Vf}(M, (\mathsf{Sign}(M, \mathsf{KUpd}(\mathsf{sk}_0)^i), i), \mathsf{pk}) = 1$.

A digital signature scheme (Dss) is a key evolving signature scheme with only one period and a key update algorithm that always returns the empty string. XMSS is a key evolving signature scheme with $p = 2^h$ for $h \in \mathbb{N}$. The XMSS key update algorithm consists of increasing the index i in the secret key and running the BDS algorithm to prepare the next authentication path. This is done after every signature.

The standard security model for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) as described in Section 2.1.2. We translate it to the setting of key evolving signature schemes, using the following experiment. Let $\textsc{Kes}(1^n, p)$ denote a key evolving signature scheme with security parameter n and number of periods p. The experiment has two phases. During the chosen message attack phase (**cma**), the adversary is allowed to collect signatures on messages of her choice like in the EU-CMA model. In contrast to the EU-CMA model, the adversary might do this up to p times, once for each time period. The adversary algorithm $\mathcal{A}$ is given oracle access to an instance of a signature oracle Sign initialized with secret key sk_i and index i, denoted by $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_i,\cdot,i)}$. Afterwards, in the forgery phase (**forge**), the adversary has to come up with an existential forgery like in the EU-CMA model. The **state** variable allows the adversary to keep a state and the **out** variable allows the adversary to switch from the **cma** to the **forge** phase.

```
Experiment $\mathsf{Exp}^{\text{EU-CMA}}_{\text{Kes}(1^n,p)}(\mathcal{A})$
    $i \longleftarrow 0$, state ⟵ null, out ⟵ null, $(\mathsf{sk}_0, \mathsf{pk}) \longleftarrow \mathsf{Kg}(1^n, p)$
    While $i < p$ And out $\neq$ halt
        $(\mathbf{out}, \mathbf{state}) \longleftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_i,\cdot,i)}(1^n, \mathbf{cma}, \mathsf{pk}, \mathbf{state})$
        i++; $\mathsf{sk}_i \longleftarrow \mathsf{KUpd}(\mathsf{sk}_{i-1}, i)$
    $(M^\star, \sigma^\star, i^\star) \longleftarrow \mathcal{A}(1^n, \mathbf{forge}, \mathbf{state})$
    Return 1 iff $\mathsf{Vf}(\mathsf{pk}, M^\star, (\sigma^\star, i^\star)) = 1$
        And Sign was not queried for a signature on $M^\star$
```

For the success probability of an adversary $\mathcal{A}$ in the above experiment we write

$$\mathrm{Succ}^{\text{EU-CMA}}_{\text{Kes}(1^n,p)}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\text{EU-CMA}}_{\text{Kes}(1^n,p)}(\mathcal{A}) = 1\right].$$

When we talk about the runtime of an adversary $\mathcal{A}$ in the above experiment, it refers to the sum of runtimes over all executions of $\mathcal{A}$ in the experiment. Now we can define EU-CMA security for key evolving signature schemes.

Definition 4.2 (EU-CMA for Kes). *Let $n, q \in \mathbb{N}$, $t, q = \mathrm{poly}(n)$, Kes a key evolving signature scheme. Fix $p \in \mathbb{N}$. We call Kes EU-CMA-secure, if the maximum success probability $\mathrm{InSec}^{\text{EU-CMA}}(\text{Kes}(1^n, T); t, q)$ of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, making at most q queries to each instance of* Sign *in the above experiment, is negligible in n:*

$$\mathrm{InSec}^{\text{EU-CMA}}(\text{Kes}(1^n, T); t, q) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\text{EU-CMA}}_{\text{Kes}(1^n,T)}(\mathcal{A})\} = \mathrm{negl}(n).$$

Please note that for a Dss this translates to the initial notion again.

Pseudorandom Generators Pseudorandom generators (PRG) are functions that stretch a random input to a longer pseudorandom output. We follow the notion of [BY03]: Let $n \in \mathbb{N}$, $b = \mathrm{poly}(n)$, $b > n$, $G_n : \{0,1\}^n \to \{0,1\}^b$ and $\mathcal{A}$ an adversary that given a b-bit string returns a bit. The notion is defined using the two following experiments, one where $\mathcal{A}$ gets a random string as input and another one where the input of $\mathcal{A}$ is an output of the PRG:

Experiment $\mathsf{Exp}_{G_n}^{\text{PRG}-1}(\mathcal{A})$	**Experiment** $\mathsf{Exp}_{G_n}^{\text{PRG}-0}(\mathcal{A})$
$X \xleftarrow{\$} \{0,1\}^n; C \longleftarrow G_n(X)$	$C \xleftarrow{\$} \{0,1\}^b$
$g \longleftarrow \mathcal{A}(C)$	$g \longleftarrow \mathcal{A}(C)$
Return g	**Return** g

The success probability of an adversary $\mathcal{A}$ against the security of PRG G_n is defined as the ability of the adversary to distinguish both experiments:

$$\mathrm{Succ}_{G_n}^{\text{PRG}}(\mathcal{A}) = \left|\Pr\left[\mathsf{Exp}_{G_n}^{\text{PRG}-1}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Exp}_{G_n}^{\text{PRG}-0}(\mathcal{A}) = 1\right]\right|.$$

Now we define secure pseudorandom generators.

Definition 4.3 (PRG)**.** *Let $n, t \in \mathbb{N}$, $t = \mathrm{poly}(n)$, G_n as above. We call G_n a secure pseudorandom generator, if* $\mathrm{InSec}^{\text{PRG}}(G_n; t)$*, the maximum success probability of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, is negligible in n:*

$$\mathrm{InSec}^{\text{PRG}}(G_n; t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}_{G_n}^{\text{PRG}}(\mathcal{A})\} = \mathrm{negl}(n).$$

4.2.2 XMSS is Existentially Unforgeable under Chosen Message Attacks

In the following, we prove XMSS EU-CMA secure in the standard model and discuss some implications of this result. We prove the following theorem for XMSS when using W-OTS$^\$$:

Theorem 4.4. *If $\mathcal{H}_n$ is a second-preimage resistant hash function family and $\mathcal{F}_n$ a pseudorandom function family, then XMSS is existentially unforgeable under adaptive chosen message attacks.*

Before we give the proof of Theorem 4.4, we want to highlight one implication of this result: The security assumptions of XMSS are minimal. From [Rom90] it is known that the minimal security assumption for complexity based cryptography, namely the existence of a one-way function, is the necessary condition for the existence of a secure digital signature scheme. As already mentioned in Section 2.1.1, in [Rom90] the construction of a target-collision resistant hash function family from a one-way function is presented. Since target-collision resistant hash function families are second-preimage resistant (see [RS04]), this implies that second-preimage resistant hash function families can be constructed from secure digital signature schemes. In [HILL99] the construction of a pseudorandom generator from a one-way function is presented. In [GGM86] pseudorandom function families are obtained from pseudorandom generators. It follows that secure signature schemes yield pseudorandom

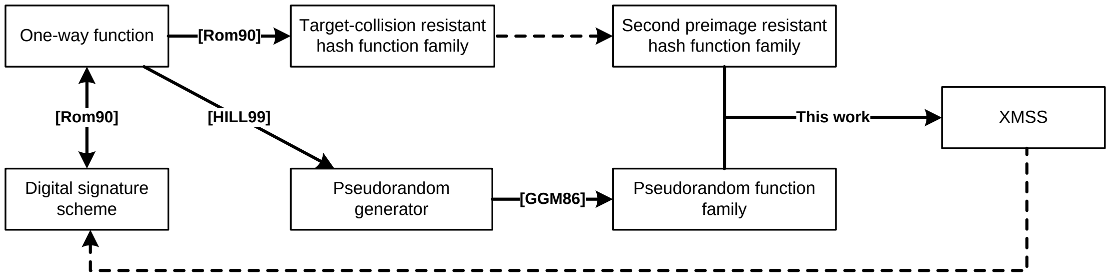

One-way function
[Rom90]
Target-collision resistant hash function family
Second preimage resistant hash function family
[Rom90]
[HILL99]
This work
XMSS
Digital signature scheme
Pseudorandom generator
[GGM86]
Pseudorandom function family

Figure 4.2: Existential relations between primitives. An arrow with a solid line from A to B says, that B can be constructed from A. A dashed line from A to B says, that a primitive that fulfills A also fulfills B.

function families. Those constructions imply that there exists a secure instance of XMSS if there exists any secure digital signature scheme and therefore complexity based cryptography at all. This implies that the security requirements for XMSS are minimal. The relations between the primitives are also displayed in Figure 4.2.

Next, we give the proof of Theorem 4.4. The proof uses another view on the construction of XMSS. Look at XMSS the following way: XMSS uses W-OTS with pseudorandom key generation. The ℓn-bit W-OTS secret keys are generated using GEN and an n-bit (pseudo-)random input. This variant of W-OTS is used with the XMSS-Tree construction to obtain a many-time signature scheme. The 2^h n-bit inputs for the key generation are again generated using GEN and a random n-bit string. In our proof we iteratively show that each of these constructions is secure with the last construction being XMSS.

Proof of Theorem 4.4. First we look at the key generation algorithm Kg in more detail. Kg uses the PRG $\mathsf{GEN}_\lambda(\mu) = \mathrm{F}_\mu(0)||\dots||\mathrm{F}_\mu(\lambda-1)$ from the last section. The W-OTS secret key is generated using $\mathsf{GEN}_\ell(\mu)$ where μ in turn is the ith n-bit string of the output of $\mathsf{GEN}_{2^h}(\textsc{Seed})$. We show that GEN_λ is a secure PRG if the used function family is pseudorandom.

Claim 4.5. *Let $n, \lambda \in \mathbb{N}, \mu \in \{0,1\}^n$, $\mathcal{F}_n = \{\mathrm{F}_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$ be a pseudorandom function family with insecurity function* $\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; t, q)$. *Then* $\mathsf{GEN}_\lambda : \{0,1\}^n \rightarrow \{0,1\}^{\lambda n}$,

$$\mathsf{GEN}_\lambda(\mu) = \mathrm{F}_\mu(0)||\dots||\mathrm{F}_\mu(\lambda-1)$$

is a PRG with insecurity function

$$\mathrm{InSec}^{\mathrm{PRG}}(\mathsf{GEN}_\lambda; t) = \mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; (t+\lambda), \lambda).$$

Proof of Claim 4.5. For the sake of contradiction assume there exists an adversary $\mathcal{A}$ distinguishing the output of GEN_λ from a uniformly random λn bit string. Then we can build an oracle machine $\mathcal{M}^\mathcal{A}$ that given access to $\mathcal{A}$ distinguishes $\mathcal{F}_n$ from $\mathcal{AF}(n,n)$. $\mathcal{M}^\mathcal{A}$ queries Box with $0,\dots,\lambda-1$ and hands the concatenation of the results to $\mathcal{A}$. Then $\mathcal{M}^\mathcal{A}$ simply forwards $\mathcal{A}$'s output. $\mathcal{M}^\mathcal{A}$ succeeds with the same probability as $\mathcal{A}$. □

Now, we show that one can replace the random input of the key generation algorithm by a pseudorandom one. So if we look at W-OTS using $\mathsf{GEN}_\ell(\mu)$ to generate the secret key from one n-bit string and assume that μ is chosen uniformly at random for the moment, then the following claim tells us, that this is almost as secure as using ℓn random bits. Furthermore it tells us, that we can use n random bits and GEN_{2^h} to generate the 2^h bit strings of length n that are used to generate the 2^h W-OTS key pairs of XMSS.

Claim 4.6. *Let $n, n', q, t, p \in \mathbb{N}$, $G_n : \{0,1\}^n \to \{0,1\}^{\lambda n}$ be a PRG with insecurity function $\mathrm{InSec}^{\mathrm{PRG}}(G_n;t)$ that stretches n bit random input to λn bit pseudorandom output and let $\textsc{Kes} = (\mathsf{Kg}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vf})$ be a key evolving signature scheme with insecurity function $\mathrm{InSec}^{\text{EU-CMA}}\big(\textsc{Kes}(1^{n'},p);t,q\big)$ that needs no more than λn bits of random input for key generation and key update. Further, let $\textsc{Kes}^\star = (\mathsf{Kg}^\star, \mathsf{KUpd}^\star, \mathsf{Sign}, \mathsf{Vf})$ be the variant of $\textsc{Kes}$ that uses G_n to generate the λn bits required for key generation. Then $\textsc{Kes}^\star$ is a key evolving signature scheme with insecurity function*

$$\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}^\star(1^{n'},p);t,q\right) = \mathrm{InSec}^{\mathrm{PRG}}(G_n;t') + \mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right),$$

where $t' = t + \mathsf{t}_{\mathsf{Kg}} + p(q\mathsf{t}_{\mathsf{Sign}} + t_{\mathsf{KUpd}^\star}) + \mathsf{t}_{\mathsf{Vf}}$.

The intuition behind the proof for the above claim is the following. If the success probability of an adversary against the scheme with pseudorandom key generation differs from its success probability against the original scheme, this difference can only be caused by the pseudorandom key generation. In the proof we show how to use this difference to distinguish between outputs of the PRG and random strings.

Proof of Claim 4.6. We want to bound the success probability of any adversary $\mathcal{A}$ that runs within time t, making at most q queries to each instance of Sign, i.e. we want to limit the insecurity function $\mathrm{InSec}^{\text{EU-CMA}}\big(\textsc{Kes}^\star(1^{n'},p);t,q\big)$. Given such an adversary, we can build an oracle machine $\mathcal{M}^\mathcal{A}$ distinguishing the output of G_n from random λn bit strings as described in algorithm 4.1.

We construct $\mathcal{M}^{\mathcal{A}}$ in the following way. On input of a challenge $C \in \{0,1\}^{\lambda n}$, $\mathcal{M}^{\mathcal{A}}$ computes a key pair $(\mathsf{pk}, \mathsf{sk}_0)$ for $\textsc{Kes}^\star$ using C instead of the output of G_n. Next $\mathcal{M}^{\mathcal{A}}$ calls $\mathcal{A}^{\mathsf{Sign}=\mathcal{M}}(1^n, \mathbf{cma}, \mathsf{pk}, \mathbf{state})$ for each time period $i < p$ or until $\mathcal{A}$ indicates to switch to the **forge** phase. At the end of each period the key update algorithm is called using C instead of the output of G_n. If $\mathcal{A}$ queries the oracle Sign for a signature during time period i, $\mathcal{M}^{\mathcal{A}}$ computes the signature using sk_i. $\mathcal{M}^{\mathcal{A}}$ answers up to q queries per time period. If $\mathcal{A}$ returns a valid forgery, $\mathcal{M}^{\mathcal{A}}$ returns 1 and 0 otherwise. $\mathcal{M}^{\mathcal{A}}$ runs in time $t + \mathsf{t}_{\mathsf{Kg}} + p(q\mathsf{t}_{\mathsf{Sign}} + t_{\mathsf{KUpd}^\star}) + \mathsf{t}_{\mathsf{Vf}}$.

Algorithm 4.1: $\mathcal{M}^{\mathcal{A}}$

```
Input: Security parameter $n'$ and challenge $C \in \{0,1\}^{\lambda n'}$
Output: $g \in \{0,1\}$
compute $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{Kg}(1^{n'}, p)$ using $C$ as the randomness of $\mathsf{Kg}^\star$;
out $\longleftarrow$ null, state $\longleftarrow$ null, $i \longleftarrow 0$;
while $i < p$ and out $\neq$ halt do
    run $(\mathbf{out}, \mathbf{state}) \longleftarrow \mathcal{A}^{\mathsf{Sign}=\mathcal{M}}(1^n, \mathbf{cma}, \mathsf{pk}, \mathbf{state})$;
    if $\mathcal{A}$ queries Sign in time period $i$ then
        answer up to $q$ queries using $\mathsf{sk}_i$
    end
    run $\mathsf{sk}_i \longleftarrow \mathsf{KUpd}^\star(\mathsf{sk}_i, i)$;
    i++;
end
if $(M^\star, \sigma^\star, i^\star) \longleftarrow \mathcal{A}(1^n, \textit{forge}, \textit{state})$ is a valid forgery then
    return $g = 1$
else
    return $g = 0$
end
```

Now we calculate the success probability of $\mathcal{M}^{\mathcal{A}}$. If $\mathcal{M}^{\mathcal{A}}$ is in $\mathsf{Exp}_{G_n}^{\text{PRG}-1}$, C is pseudorandom output of G_n. Hence, $\mathcal{A}$ succeeds with probability $\mathrm{Succ}^{\text{EU-CMA}}_{\textsc{Kes}^\star(1^{n'},p)}(\mathcal{A})$ by definition and we get

$$\mathsf{Pr}\left[\mathsf{Exp}_{G_n}^{\text{PRG}-1}(\mathcal{M}^{\mathcal{A}}) = 1\right] = \mathrm{Succ}^{\text{EU-CMA}}_{\textsc{Kes}^\star(1^{n'},p)}(\mathcal{A}) \,.$$

If $\mathcal{M}^{\mathcal{A}}$ is in $\mathsf{Exp}_{G_n}^{\text{PRG}-0}$, C is chosen uniformly at random. In this case $\mathcal{A}$ succeeds with probability $\leq \mathrm{InSec}^{\text{EU-CMA}}\big(\textsc{Kes}(1^{n'}, p); t, q\big)$. Otherwise $\mathcal{A}$ would be a forger for $\textsc{Kes}$ that running in time t succeeds with probability greater than

$\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right)$, which would contradict the assumption on the security of $\textsc{Kes}$. So we get

$$\mathsf{Pr}\left[\mathsf{Exp}_{G_n}^{\text{PRG}-0}(\mathcal{M}^{\mathcal{A}})=1\right]\leq \mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right).$$

Altogether this leads to

$$\begin{aligned}\mathrm{InSec}^{\text{PRG}}\left(G_n;t'\right) &\geq \mathrm{Succ}_{G_n}^{\text{PRG}}\left(\mathcal{M}^{\mathcal{A}}\right)\\ &= \left|\mathsf{Pr}\left[\mathsf{Exp}_{G_n}^{\text{PRG}-1}(\mathcal{M}^{\mathcal{A}})=1\right]-\mathsf{Pr}\left[\mathsf{Exp}_{G_n}^{\text{PRG}-0}(\mathcal{M}^{\mathcal{A}})=1\right]\right|\\ &\geq \mathrm{Succ}_{\textsc{Kes}^{\star}(1^{n'},p)}^{\text{EU-CMA}}\left(\mathcal{A}\right)-\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right)\end{aligned}$$

and therefore

$$\mathrm{Succ}_{\textsc{Kes}^{\star}(1^{n'},p)}^{\text{EU-CMA}}\left(\mathcal{A}\right)\leq \mathrm{InSec}^{\text{PRG}}\left(G_n;t'\right)+\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right)$$

with $t'=t+\mathsf{t}_{\mathsf{Kg}}+p(q\mathsf{t}_{\mathsf{Sign}}+t_{\mathsf{KUpd}^{\star}})+\mathsf{t}_{\mathsf{Vf}}$. As this holds for any adversary $\mathcal{A}$ running in time $\leq t$, making at most q queries to each instance of Sign we get

$$\begin{aligned}&\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'},p);t,q\right)\\ &\qquad\qquad\leq \mathrm{InSec}^{\text{PRG}}\left(G_n;t'\right)+\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'},p);t,q\right)\end{aligned}$$

□

In [DOTV08] the authors give an exact security proof for an MSS where the Merkle tree is replaced by the XMSS-Tree construction. Using W-OTS as OTS, we obtain the following insecurity function for the EU-CMA-security of this XMSS-Tree construction:

$$\begin{aligned}&\mathrm{InSec}^{\text{EU-CMA}}\left(\text{XMSS-Tree}(1^n,2^h);t,q=1\right)\\ &\quad\leq 2\cdot\max\left\{(2^{h+\log\ell}-1)\mathrm{InSec}^{\text{SPR}}\left(\mathcal{H}_n;t'\right),2^h\cdot\mathrm{InSec}^{\text{EU-CMA}}\left(\text{W-OTS}(1^n);t',1\right)\right\}\end{aligned}$$

with $t'=t+2^h\cdot t_{\mathsf{Sign}}+t_{\mathsf{Vf}}+t_{\mathsf{Kg}}$.

Now we can combine all this to conclude the proof. We use Claim 4.6 with the insecurity functions of W-OTS$^{\$}$ and GEN_{ℓ}. This gives us the insecurity function for W-OTS$^{\$}$ with pseudorandom key generation. We insert this in the insecurity function for XMSS-Tree. Finally we apply Claim 4.6 again, this time using the obtained insecurity function for XMSS-Tree with W-OTS with pseudorandom key generation and GEN_{2^h}. Altogether this leads to

$$\begin{aligned}
&\mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}(1^n, 2^h); t, 1\right)\\
&\qquad\leq \mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n; (t' + 2^h), 2^h\right)\\
&\qquad+ 2 \cdot \max \left\{ \begin{array}{l} (2^{h+\log \ell} - 1) \cdot \mathrm{InSec}^{\text{SPR}}\left(\mathcal{H}_n; t'\right), \\ 2^h \Big(\mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n; (t' + \ell), \ell\right) \\ \quad + \frac{\ell w \kappa^{w-1}}{\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)} \cdot \mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n; t' + w + 1, 2\right) \Big) \end{array} \right\}
\end{aligned} \tag{4.1}$$

where $t' = t + 2^h \cdot t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + t_{\mathsf{Kg}}$. This concludes the proof. □

4.3 Forward Security

In this section we show that XMSS is forward secure if we slightly modify the key generation process. Namely, we now use a forward secure PRG to generate the seeds for the W-OTS keys. This was also used by Krawczyk [Kra00] to reduce the key size of his generic forward secure signature scheme based on a certification tree. However, it turns out that the proof becomes much more complicated in our case as we have to deal with the rather static structure of a hash tree instead of the dynamic structure of a certification tree. Before we describe the modification and state our main theorem, we provide the used definitions.

4.3.1 Preliminaries

In the following we define stateful pseudorandom generators and the notion of forward security for these generators, but first we formally define forward secure signature schemes.

Forward Secure Signature Schemes The notion of forward security is a security notion for key evolving signature schemes as defined in the last section. We follow the definition of [BM99]. Again, we define the notion using an experiment which is given below. This experiment differs only slightly from the one used to define EU-CMA-security for key evolving signature schemes. The difference is that the adversary now gets the ability to break in. This means that during the **cma** phase, the adversary is allowed to indicate to the experiment that it wants to break in, setting the **out** variable to **breakin**. In this case, the experiment switches from the **cma** phase to the **forge** phase and the adversary is given the secret key sk_{i-1} of the current time period (Please note that the last two statements in the while loop are increasing the index i and updating the secret key. Hence the last key used during the **cma** phase has now index $i-1$). As an existential forgery for the current or an upcoming time period would be trivial, the adversary has to come up with an existential forgery for a past time period.

```
Experiment $\mathsf{Exp}^{\mathrm{FSSIG}}_{\mathrm{K{\small ES}}(1^n,p)}(\mathcal{A})$
    $i \longleftarrow 0$, $\mathbf{state} \longleftarrow \mathbf{null}$, $(\mathsf{sk}_0, \mathsf{pk}) \longleftarrow \mathsf{Kg}(1^n, p)$
    While $i < p$ And $\mathbf{out} \neq \mathbf{breakin}$
        $(\mathbf{out}, \mathbf{state}) \longleftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_i,\cdot,i)}(1^n, \mathbf{cma}, \mathsf{pk}, \mathbf{state})$
        i++, $\mathsf{sk}_i \longleftarrow \mathsf{KUpd}(\mathsf{sk}_{i-1}, i)$
    $(M^\star, \sigma^\star, i^\star) \longleftarrow \mathcal{A}(1^n, \mathbf{forge}, \mathbf{state}, \mathsf{sk}_{i-1})$
    If $\mathsf{Vf}(\mathsf{pk}, M^\star, (\sigma^\star, i^\star)) = 1$, $\mathsf{Sign}(\mathsf{sk}_{i^\star}, \cdot, i^\star)$ was not queried for a signature on $M^\star$
        And $i^\star < i - 1$ Return 1
    Return 0
```

For the success probability of an adversary $\mathcal{A}$ in the above experiment we write

$$\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{K{\small ES}}(1^n,p)}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathrm{FSSIG}}_{\mathrm{K{\small ES}}(1^n,p)}(\mathcal{A}) = 1\right].$$

When we talk about the runtime of an adversary $\mathcal{A}$ in the above experiment, it refers to the sum of runtimes over all executions of $\mathcal{A}$ in the experiment. Now we can define FSSIG for key evolving signature schemes.

Definition 4.7 (FSSIG). *Let $n, q \in \mathbb{N}$, $t = \mathrm{poly}(n)$,* K{\small ES} → Kes *a key evolving signature scheme. Fix $p \in \mathbb{N}$. We call* $\mathrm{K{\small ES}}(1^n, p)$ FSSIG*-secure, if* $\mathrm{InSec}^{\mathrm{FSSIG}}(\mathrm{K{\small ES}}(1^n, p); t, q)$*, the maximum success probability of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, making at most q queries to each instance of* Sign *in the above experiment, is negligible in n:*

$$\mathrm{InSec}^{\mathrm{FSSIG}}(\mathrm{K{\small ES}}(1^n, p); t, q) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{K{\small ES}}(1^n,p)}(\mathcal{A})\} = \mathrm{negl}(n).$$

Note, that forward security defined as above implies EU-CMA-security.

Forward Secure Pseudorandom Bit Generators Informally, a forward secure PRG is a stateful PRG that starts from a random initial state. Given a state, it outputs a new state and some output bits. Even if an adversary manages to learn the secret state of a forward secure PRG, it is unable to distinguish the former outputs from random bit strings. More formally, a stateful PRG is a function $G_n : \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^b$, for $n, b \in \mathbb{N}, b = \mathrm{poly}(n)$, that on input of a state State_i of length n outputs a new state State_{i+1} and b output bits $\mathrm{O{\small UT}}_{i+1}$. Forward security for a stateful PRG that is used to produce no more than $\widetilde{n}$ outputs is defined using the two following experiments $\mathsf{Exp}_{G_n}^{\text{FSPRG}-1}(\mathcal{A})$ and $\mathsf{Exp}_{G_n}^{\text{FSPRG}-0}(\mathcal{A})$ which are based on the ones from [BY03]. In both experiments the adversary $\mathcal{A}$ is allowed to collect up to $\widetilde{n}$ bit strings during the **find** phase. In the first experiment these bit strings are outputs of G_n, in the second experiment these bit strings are chosen uniformly at random. The adversary can keep a history using the variable h. The adversary can switch to the **guess** phase setting $d = \textbf{guess}$. In the **guess** phase, the adversary gets the current state of G_n and has to output one bit, indicating wether the bit strings were uniformly random or generated by G_n:

```
Experiment $\mathsf{Exp}_{G_n}^{\text{FSPRG}-1}(\mathcal{A})$
    $\mathsf{State}_0 \overset{\$}{\longleftarrow} \{0,1\}^n$
    $i \longleftarrow 0; h, d \longleftarrow \textbf{null}$
    Repeat
        $i \longleftarrow i + 1$
        $(\textsc{Out}_i, \mathsf{State}_i) \longleftarrow G_n(\mathsf{State}_{i-1})$
        $(d, h) \overset{\$}{\longleftarrow} \mathcal{A}(1^n, \textbf{find}, \textsc{Out}_i, h)$
    Until $(d = \textbf{guess})$ Or $(i = \widetilde{n})$
    $g \overset{\$}{\longleftarrow} \mathcal{A}(1^n, \textbf{guess}, \mathsf{State}_i, h)$
    Return $g$
```

```
Experiment $\mathsf{Exp}_{G_n}^{\text{FSPRG}-0}(\mathcal{A})$
    $\mathsf{State}_0 \overset{\$}{\longleftarrow} \{0,1\}^n$
    $i \longleftarrow 0; h, d \longleftarrow \textbf{null}$
    Repeat
        $i \longleftarrow i + 1$
        $(\textsc{Out}_i, \mathsf{State}_i) \longleftarrow G_n(\mathsf{State}_{i-1})$
        $\textsc{Out}_i \overset{\$}{\longleftarrow} \{0,1\}^b$
        $(d, h) \overset{\$}{\longleftarrow} \mathcal{A}(1^n, \textbf{find}, \textsc{Out}_i, h)$
    Until $(d = \textbf{guess})$ Or $(i = \widetilde{n})$
    $g \overset{\$}{\longleftarrow} \mathcal{A}(1^n, \textbf{guess}, \mathsf{State}_i, h)$
    Return $g$
```

The success probability of an adversary $\mathcal{A}$ is denoted by

$$\mathrm{Succ}_{\mathsf{GEN}}^{\text{FSPRG}}(\mathcal{A}) = \left|\Pr\left[\mathsf{Exp}_{G_n}^{\text{FSPRG}-1}(\mathsf{Dis}) = 1\right] - \Pr\left[\mathsf{Exp}_{G_n}^{\text{FSPRG}-0}(\mathsf{Dis}) = 1\right]\right|.$$

When we talk about the runtime of an adversary $\mathcal{A}$ in the above experiment, it refers to the sum of runtimes over all executions of $\mathcal{A}$ in the experiment as in the case of forward secure signature schemes. Now we can define forward security for a stateful PRG.

Definition 4.8 (FSSIG). *Let $n, \widetilde{n} \in \mathbb{N}$, $t = \mathrm{poly}(n)$, G_n a stateful PRG as defined above. We call G_n FSPRG-secure, if $\mathrm{InSec}^{\text{FSPRG}}(G_n; t)$, the maximum success probability of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, in the experiment above, is negligible in n:*

$$\mathrm{InSec}^{\text{FSPRG}}(G_n; t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}_{G_n}^{\text{FSPRG}}(\mathcal{A})\} = \mathrm{negl}(n).$$

4.3.2 XMSS is Forward Secure

In the following we describe the modifications needed to make XMSS forward secure. Then we state our main theorem and prove it. To make XMSS forward secure we use a forward secure PRG FsGen when generating the seeds for the W-OTS secret keys. Starting from a random input Seed = State_0 of length n, FsGen uses F_n and the previous state State_{i-1} to generate n bits of pseudorandom output $\textsc{Out}_i$ and a new state State_i of length n:

$$\textsc{FsGen}(\mathsf{State}_{i-1}) = (\mathsf{State}_i||\textsc{Out}_i) = (f_{\mathsf{State}_{i-1}}(0)||f_{\mathsf{State}_{i-1}}(1)).$$

The generation of the W-OTS secret keys from the seeds still utilizes GEN_ℓ. The secret key of the resulting forward secure XMSS contains the current state State_i instead of Seed. In contrast to the construction from Section 4.1, the seeds for the W-OTS signature keys are not easily accessible from State_i using one evaluation of F_n. To compute the authentication path, the tree traversal algorithm needs to compute several W-OTS keys before they are needed. This is very expensive using FsGen. Luckily this problem is already addressed in [BDS08]. We use their solution that requires to store $2h$ states of FsGen in the secret key.

For XMSS with the modified key generation from above using W-OTS$^\$$, we proof the following security theorem.

Theorem 4.9. *If $\mathcal{H}_n$ is a second-preimage resistant hash function family and $\mathcal{F}_n$ a pseudorandom function family, then XMSS with the modified key generation described above is a forward secure digital signature scheme.*

Informally the proof works the following way. First, we state that FsGen is a forward secure PRG using a result from [BY03]. In a second step, we show that for arbitrary but fixed h, XMSS is forward secure if the seeds for the W-OTS secret keys are generated using FsGen. The idea behind the proof is very close to the one of Claim 4.6. But this time it is more complicated to upper bound the success probability in the case of random bit strings.

Proof of Theorem 4.9. First, we revisit a result from [BY03] about the security of FsGen. There the authors show that if $\mathcal{F}_n$ is a pseudorandom function family with insecurity function $\mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; t, q)$, then FsGen is a forward secure PRG with insecurity function

$$\mathrm{InSec}^{\mathrm{FSPRG}}(\textsc{FsGen}; t) = 2\widetilde{n} \cdot \mathrm{InSec}^{\mathrm{PRF}}(\mathcal{F}_n; (t + 2\widetilde{n}), 2).$$

Now we show that XMSS is forward secure, if the seeds for the W-OTS secret keys are generated using FsGen.

Claim 4.10. *Let $n, n', h \in \mathbb{N}$,* FsGen *as described above. Let XMSS ′ be the version of XMSS where the 2^h n'-bit seeds for the W-OTS key generation are chosen uniformly at random with insecurity function* $\mathrm{InSec}^{\text{EU-CMA}}\left(\mathit{XMSS}\,'1^{n'}, 2^h); t, q = 1\right)$. *Further, let XMSS* be the modified version of XMSS that uses* FsGen *to generate the 2^h n-bit seeds required for W-OTS key generation. Then XMSS* is a forward secure signature scheme with insecurity function*

$$\begin{aligned}\mathrm{InSec}^{\text{FSSIG}}\left(\mathit{XMSS}^\star(1^{n'}, 2^h); t, 1\right) \;\leq\; & 2^h \cdot \mathrm{InSec}^{\text{FSPRG}}\left(\textsc{FsGen}; t'\right) \\ & + \mathrm{InSec}^{\text{EU-CMA}}\left(\mathit{XMSS}\,'(1^{n'}, 2^h); t, 1\right)\end{aligned}$$

$t' = t + \mathsf{t}_{\mathsf{Kg}^\star} + 2^h \mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$.

Proof of claim. We want to limit the success probability of any adversary $\mathcal{A}$ that tries to break the forward security of XMSS*. More specifically, we want to find an upper bound for the insecurity function $\mathrm{InSec}^{\text{FSSIG}}\left(\mathrm{XMSS}^\star(1^{n'}, 2^h); t, 1\right)$. Therefore we assume $\mathcal{A}$ runs within time t, making at most 1 query to each instance of Sign. Given such an adversary, we can build an oracle machine $\mathcal{M}^{\mathcal{A}}$ distinguishing the output of FsGen from truly random outputs, given black box access to $\mathcal{A}$.

We construct $\mathcal{M}^{\mathcal{A}}$ the following way. $\mathcal{M}^{\mathcal{A}}$ chooses a value $\alpha \xleftarrow{\$} \{1, \ldots, 2^h\}$ uniformly at random. During the **find** phase of the fsprg experiment, $\mathcal{M}^{\mathcal{A}}$ collects α outputs $\textsc{Out}_1, \ldots, \textsc{Out}_\alpha$ before switching to the **guess** phase. In the **guess** phase $\mathcal{M}^{\mathcal{A}}$ is given State_α. Now, $\mathcal{M}^{\mathcal{A}}$ uses FsGen and State_α to compute another $2^h - \alpha$ outputs $\textsc{Out}_{\alpha+1}, \ldots, \textsc{Out}_{2^h}$. Then $\mathcal{M}^{\mathcal{A}}$ uses $\textsc{Out}_1, \ldots, \textsc{Out}_{2^h}$ instead of the output of FsGen to generate a XMSS public key pk. Note, that to generate the W-OTS key pair for time period i, $\textsc{Out}_{i+1}$ is used. Next $\mathcal{M}^{\mathcal{A}}$ calls $\mathcal{A}^{\mathsf{Sign}=\mathcal{M}}(1^n, \mathbf{cma}, \mathsf{pk}, \mathbf{state})$ for each time period $i < \alpha$ until $\mathcal{A}$ indicates to break in. If $\mathcal{A}$ queries $\mathcal{M}^{\mathcal{A}}$ as the oracle Sign during period i, $\mathcal{M}^{\mathcal{A}}$ computes the queried signature using $\textsc{Out}_{i+1}$ to generate the corresponding W-OTS secret key. If $\mathcal{A}$ indicates to break in during a time period $i \neq \alpha - 1$ or does not indicate to break in in time period $i = \alpha - 1$, $\mathcal{M}^{\mathcal{A}}$ returns 0. If $\mathcal{A}$ indicates that it wants to break in at time period $i = \alpha - 1$, $\mathcal{M}^{\mathcal{A}}$ runs $\mathcal{A}$ in the **forge** phase with input $\mathsf{sk}_i = (\mathsf{State}_\alpha, \textsc{Out}_\alpha)$. This is all secret information that exists in time period $i = \alpha - 1$. If $\mathcal{A}$ returns a valid forgery for a time period $j < i$, then $\mathcal{M}^{\mathcal{A}}$ returns 1 and 0 otherwise. Altogether $\mathcal{M}^{\mathcal{A}}$ runs in time $\leq t' = t + \mathsf{t}_{\mathsf{Kg}^\star} + 2^h \mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$.

Now we calculate the success probability of $\mathcal{M}^{\mathcal{A}}$ in distinguishing the output of FsGen from uniformly random outputs. The probability that $\mathcal{A}$ wants to break in in time period $i = \alpha - 1$ is at least 2^{-h} as α is chosen uniformly at random. Now, if $\mathcal{M}^{\mathcal{A}}$ is run in $\mathsf{Exp}^{\text{FSPRG}-1}_{\textsc{FsGen}}(\mathcal{M}^{\mathcal{A}})$, the $\textsc{Out}_i$, $1 \leq i \leq 2^h$ are pseudorandom outputs of FsGen. Hence, $\mathcal{A}$ succeeds with probability $\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^\star(1^{n'},2^h)}(\mathcal{A})$ per definition. As $\mathcal{M}^{\mathcal{A}}$ returns 1 if $\mathcal{A}$ is successful we get

$$\Pr\left[\mathsf{Exp}^{\mathrm{FSPRG}-1}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}}) = 1\right] = 2^{-h} \cdot \mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^\star(1^{n'},2^h)}(\mathcal{A})\,.$$

If $\mathcal{M}^{\mathcal{A}}$ is in $\mathsf{Exp}^{\mathrm{FSPRG}-0}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}})$, the Out_i, $1 \leq i \leq \alpha$ are chosen uniformly at random. The remaining Out_i, $\alpha + 1 \leq i \leq 2^h$ are pseudorandom outputs of FsGen. Again, the probability that $\mathcal{A}$ wants to break in in time period $i = \alpha - 1$ is at least 2^{-h} as α is chosen uniformly at random. And again $\mathcal{M}^{\mathcal{A}}$ returns 1 if $\mathcal{A}$ succeeds. We will need an upper bound for the probability that $\mathcal{M}^{\mathcal{A}}$ returns 1. Hence we have to limit $\mathcal{A}$'s success probability for the case that $\mathcal{A}$ breaks in in time period $i = \alpha - 1$. We will show that in this case, $\mathcal{A}$ succeeds with probability $\leq \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}'(1^{n'}, 2^h); t, 1\right)$. For the moment assume this is true. Then we get

$$\Pr\left[\mathsf{Exp}^{\mathrm{FSPRG}-0}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}}) = 1\right] \leq 2^{-h} \cdot \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}'(1^{n'}, 2^h); t, 1\right)\,.$$

Putting all of this together, we get

$$\begin{aligned}
&\mathrm{InSec}^{\mathrm{FSPRG}}\left(\mathrm{FsGen}; t'\right)\\
&\geq \mathrm{Succ}^{\mathrm{FSPRG}}_{\mathrm{FsGen}}\left(\mathcal{M}^{\mathcal{A}}\right)\\
&= \left|\Pr\left[\mathsf{Exp}^{\mathrm{FSPRG}-1}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{FSPRG}-0}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}}) = 1\right]\right|\\
&\geq 2^{-h} \cdot \mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^\star(1^{n'},2^h)}(\mathcal{A}) - 2^{-h} \cdot \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}'(1^{n'}, 2^h); t, 1\right)
\end{aligned}$$

and therefore

$$\begin{aligned}
&\mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}^\star(1^{n'}, 2^h); t, 1\right) \leq \mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^\star(1^{n'},2^h)}(\mathcal{A})\\
&\leq 2^h \cdot \mathrm{InSec}^{\mathrm{FSPRG}}\left(\mathrm{FsGen}; t'\right) + \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}'(1^{n'}, 2^h); t, 1\right)\,.
\end{aligned}$$

This is the claimed result. Nevertheless, we still have to show that if $\mathcal{M}^{\mathcal{A}}$ is in $\mathsf{Exp}^{\mathrm{FSPRG}-0}_{\mathrm{FsGen}}(\mathcal{M}^{\mathcal{A}})$, $\epsilon_{\mathcal{A}}$, the success probability of $\mathcal{A}$ conditioned on the event that $\mathcal{M}^{\mathcal{A}}$ correctly guesses the time period $\mathcal{A}$ wants to break in, is limited by

$$\epsilon_{\mathcal{A}} \leq \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}'(1^{n'}, 2^h); t, 1\right)\,.$$

We do this, showing how to build an oracle machine $\hat{\mathcal{M}}^{\mathcal{A}}$ that behaves exactly like $\mathcal{M}^{\mathcal{A}}$, from $\mathcal{A}$'s point of view. In contrast to $\mathcal{M}^{\mathcal{A}}$, $\hat{\mathcal{M}}^{\mathcal{A}}$ uses $\mathcal{A}$ either to forge a signature for W-OTS with pseudorandom key generation (W-OTS*) or to find a second-preimage for a random function h from $\mathcal{H}_n$. Next, we describe $\hat{\mathcal{M}}^{\mathcal{A}}$.

$\hat{\mathcal{M}}^{\mathcal{A}}$ receives as input a second-preimage challenge, consisting of a preimage X_c and a function key K identifying a function H from $\mathcal{H}_n$ as well as a W-OTS* public key pk_c. Furthermore, $\hat{\mathcal{M}}^{\mathcal{A}}$ gets access to the corresponding signing oracle for pk_c. Like $\mathcal{M}^{\mathcal{A}}$, $\hat{\mathcal{M}}^{\mathcal{A}}$ chooses $\alpha \xleftarrow{\$} \{1, \dots, 2^h\}$ uniformly at random. Additionally, $\hat{\mathcal{M}}^{\mathcal{A}}$ chooses $\beta \xleftarrow{\$} \{0, \alpha - 1\}$ uniformly at random. Next, $\hat{\mathcal{M}}^{\mathcal{A}}$ generates 2^h W-OTS* key pairs. This is done in a way simulating the $\mathsf{Exp}^{\text{FSPRG}-0}_{\text{FsGen}}(\mathcal{M}^{\mathcal{A}})$ case: For the first α key pairs $\hat{\mathcal{M}}^{\mathcal{A}}$ uses a random seed. Then, $\hat{\mathcal{M}}^{\mathcal{A}}$ uses FsGen to compute State_α using a random seed and uses FsGen starting from State_α to generate the seeds for the remaining key pairs. Afterwards, $\hat{\mathcal{M}}^{\mathcal{A}}$ replaces the key pair on position β by pk_c. As $\beta \leq \alpha$ and pk_c corresponds to a W-OTS* key pair where the seed is chosen at random, the first α W-OTS* key pairs are now generated using random seeds and the remaining W-OTS* key pairs are generated using FsGen, exactly as in the case of $\mathcal{M}^{\mathcal{A}}$.

Next, $\hat{\mathcal{M}}^{\mathcal{A}}$ computes the XMSS-Tree starting from the bit strings of the W-OTS* public keys, using H. During the XMSS-Tree computation, $\hat{\mathcal{M}}^{\mathcal{A}}$ chooses a random node from the set of all ancestor nodes of the bit strings of the first α W-OTS* public keys. Then, $\hat{\mathcal{M}}^{\mathcal{A}}$ chooses the bit masks for the level of this node such that for this node the input to H is X_c. Then, $\hat{\mathcal{M}}^{\mathcal{A}}$ uses the resulting XMSS public key and starts to interact with $\mathcal{A}$ exactly the same way as $\mathcal{M}^{\mathcal{A}}$ does. Especially $\hat{\mathcal{M}}^{\mathcal{A}}$ aborts if $\mathcal{A}$ does not break in in time period $i = \alpha - 1$. $\hat{\mathcal{M}}^{\mathcal{A}}$ can answer all signature queries using the generated secret keys or the signing oracle for pk_c in time period $i = \beta$.

If $\mathcal{A}$ returns a valid forgery $(M', (j, \sigma', \mathsf{Auth}'))$ for time period $j < \alpha - 1$, $\hat{\mathcal{M}}^{\mathcal{A}}$ computes the W-OTS* public key pk'_j using the signature σ'. Now, there are two mutual exclusive cases:

(Case 1) If $\mathsf{pk}'_j = \mathsf{pk}_j$, σ' is an existential forgery for W-OTS*. So, if $j = \beta$ $\hat{\mathcal{M}}^{\mathcal{A}}$ returns (M, σ'), otherwise $\hat{\mathcal{M}}^{\mathcal{A}}$ aborts.

(Case 2) If $\mathsf{pk}'_j \neq \mathsf{pk}_j$, by the pigeon hole principle, there must be one node on the paths from pk'_j and pk_j to the root, where the paths collide the first time. As this node is an output of H and the inputs are different, $\hat{\mathcal{M}}^{\mathcal{A}}$ found a collision. If one of the inputs is X_c, $\hat{\mathcal{M}}^{\mathcal{A}}$ returns the second-preimage. Otherwise $\hat{\mathcal{M}}^{\mathcal{A}}$ aborts. $\hat{\mathcal{M}}^{\mathcal{A}}$ runs in time $t' = t + 2^h \cdot t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + t_{\mathsf{Kg}}$.

Now we compute the success probability of $\hat{\mathcal{M}}^{\mathcal{A}}$. Per assumption $\mathcal{A}$ breaks in in time period $i = \alpha - 1$. From $\mathcal{A}$'s point of view, $\hat{\mathcal{M}}^{\mathcal{A}}$ behaves exactly as $\mathcal{M}^{\mathcal{A}}$. Hence, $\mathcal{A}$ returns a valid forgery with probability $\epsilon_{\mathcal{A}}$. In case 1, $\hat{\mathcal{M}}^{\mathcal{A}}$ succeeds with probability $\Pr[j = \beta] = \frac{1}{\alpha}$. But the success probability of $\hat{\mathcal{M}}^{\mathcal{A}}$ for this case is also upper bounded by its success probability against the EU-CMA-security of W-OTS*, which is bound by $\text{InSec}^{\text{EU-CMA}}$ (W-OTS$(1^n, 1); t', 1)$. Now we analyze case 2. We write Ancestors_α for the set of all ancestor nodes of the bit strings of the first α W-OTS* public keys. Then $\hat{\mathcal{M}}^{\mathcal{A}}$ succeeds with probability $\frac{1}{|\text{Ancestors}_\alpha|}$. But the success probability of $\hat{\mathcal{M}}^{\mathcal{A}}$ in case 2 is also upper bounded by the second-preimage resistance of $\mathcal{H}_n$, $\text{InSec}^{\text{SPR}}\left(\mathcal{H}_n; t'\right)$. One of these cases appears with probability at least $\frac{1}{2}$. Summing up we get

$$\epsilon_{\mathcal{A}} \leq 2 \cdot \max \left\{ \begin{array}{l} (\alpha+1) \cdot \text{InSec}^{\text{EU-CMA}}\left(\text{W-OTS}(1^n, T=1); t', q=1\right), \\ |\text{Ancestors}_\alpha| \cdot \text{InSec}^{\text{SPR}}\left(\mathcal{H}_n; t'\right) \end{array} \right\}.$$

The right part of the equation takes its maximum value for $\alpha = 2^h$. Comparing this with the result from [DOTV08] given in the proof of Theorem 4.4 we see that the right part of the equation for $\alpha = 2^h$ is exactly $\text{InSec}^{\text{EU-CMA}}\left(\text{XMSS}'(1^{n'}, 2^h); t, 1\right)$. This concludes the claim. □

Combining this with the above result for FsGen yields that the maximum success probability over all adversaries running in time $\leq t$, making at most 1 query to each instance of Sign, in attacking the forward security of XMSS* , $\text{InSec}^{\text{FSSIG}}\left(\text{XMSS}^\star; t, 1\right)$, is bounded by

$$\begin{aligned} &\text{InSec}^{\text{FSSIG}}\left(\text{XMSS}^\star; t, 1\right) \\ &\qquad \leq 2^{2h+1} \cdot \text{InSec}^{\text{PRF}}\left(\mathcal{F}_n; (t'+2), 2\right) \\ &\qquad + \ 2 \cdot \max \left\{ \begin{array}{l} (2^{h+\log \ell} - 1) \cdot \text{InSec}^{\text{SPR}}\left(\mathcal{H}_n; t'\right), \\ 2^h \Big(\text{InSec}^{\text{PRF}}\left(\mathcal{F}_n; (t'+\ell), \ell\right) \\ \quad + \frac{\ell w \kappa^{w-1}}{\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)} \cdot \text{InSec}^{\text{PRF}}\left(\mathcal{F}_n; (t'+w+1), 2\right) \Big) \end{array} \right\}, \end{aligned} \tag{4.2}$$

with $t' = t + 2^h \cdot t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + t_{\mathsf{Kg}}$. This concludes the proof. □

4.4 Theoretical Performance

In this section we discuss the theoretical performance of XMSS. We show that the performance of XMSS is closely related to that of the used function families. Hence, XMSS and its forward secure variant are efficient if $\mathcal{H}_n$ is an efficient second-preimage resistant hash function family and $\mathcal{F}_n$ an efficient pseudorandom function family. Efficient here refers to the runtimes and space requirements for sufficiently secure parameters. As the forward secure variant of XMSS is slightly less performant and requires more space, we do the analysis for the forward secure variant. For practical parameters and runtimes see Chapters 6 and 7.

The tree traversal algorithm used to compute the nodes of the authentication path has a huge influence on the runtime of the signature algorithm as well as on the storage requirements for the state. For the BDS algorithm it has been shown in [BDS09] that for $h, k \geq 2$, $h-k$ is even, it requires $\left(5h + \left\lfloor \frac{h}{2} \right\rfloor - 5k - 2 + 2^k\right) \cdot n$ bits for its state, including the secret key. Further it requires no more than $(h-k)/2+1$ leaf computations in the XMSS tree, $3(h-k-1)/2+1$ evaluations of $\mathcal{H}_n$, and $h-k$ calls to FsGen per signature. A leaf computation consists of evaluating FsGen once, evaluating GEN_ℓ once, computing the W-OTS public key, and computing the L-tree. The $h-k$ extra calls to FsGen are used to compute upcoming states of FsGen. There is no need to compute the output bits, because only the next state is required. Therefore this requires only $h-k$ evaluations of functions from $\mathcal{F}_n$.

The runtime of all three algorithms of XMSS is dominated by the time required to evaluate elements of $\mathcal{F}_n$ and $\mathcal{H}_n$. We ignore the computational overhead for adding the bitmasks, control flow, and computing the base w representation of the message as it is negligible for every practical choice of $\mathcal{F}_n$ and $\mathcal{H}_n$. We write t_{H} (t_{F}) for the runtime of functions from $\mathcal{H}_n$ ($\mathcal{F}_n$, respectively). Using a simple counting argument we obtain the following result:

For one call to the XMSS signature algorithm, the runtime is bounded by

$$\mathsf{t}_{\mathsf{Sign}} \leq t_{\mathrm{H}} \left(\frac{\ell+3}{2} \cdot (h-k) + \ell - \frac{1}{2} \right) + t_{\mathrm{F}} \left(\frac{\ell w + 4}{2} \cdot (h-k) + \ell w + 2 \right).$$

For one call to the XMSS signature verification algorithm, the runtime is bounded by

$$\mathsf{t}_{\mathsf{Vf}} \leq t_{\mathrm{H}}(h+\ell) + t_{\mathrm{F}}(\ell w).$$

For one call to the XMSS key generation algorithm, the runtime is bounded by

$$\mathsf{t}_{\mathsf{Kg}} \leq t_{\mathrm{H}} \left(2^h(\ell+1) \right) + t_{\mathrm{F}} \left(2^h(2+\ell(w+1)) \right).$$

The space requirements for the internal state of Sign and Kg (including the secret key) are determined by the space requirements of the tree traversal algorithm plus the space requirements for the current state of FsGen and the index. Vf needs no internal state. Hence, the space used by XMSS, using the BDS algorithm, is at most

$$|\mathsf{SK}| \leq \left(5h + \left\lfloor \frac{h}{2} \right\rfloor - 5k - 2 + 2^k + 1 \right) \cdot n + |i| \text{ bits},$$

where $|i|$ denotes the maximal binary length of the index. $2(h-k)n + n$ of these bits have to be kept secret. For the remaining bits there is no secrecy requirement. The public key has a size of

$$|\mathsf{PK}| = 2n(h + \lceil \log \ell \rceil + 1) \text{ bits}.$$

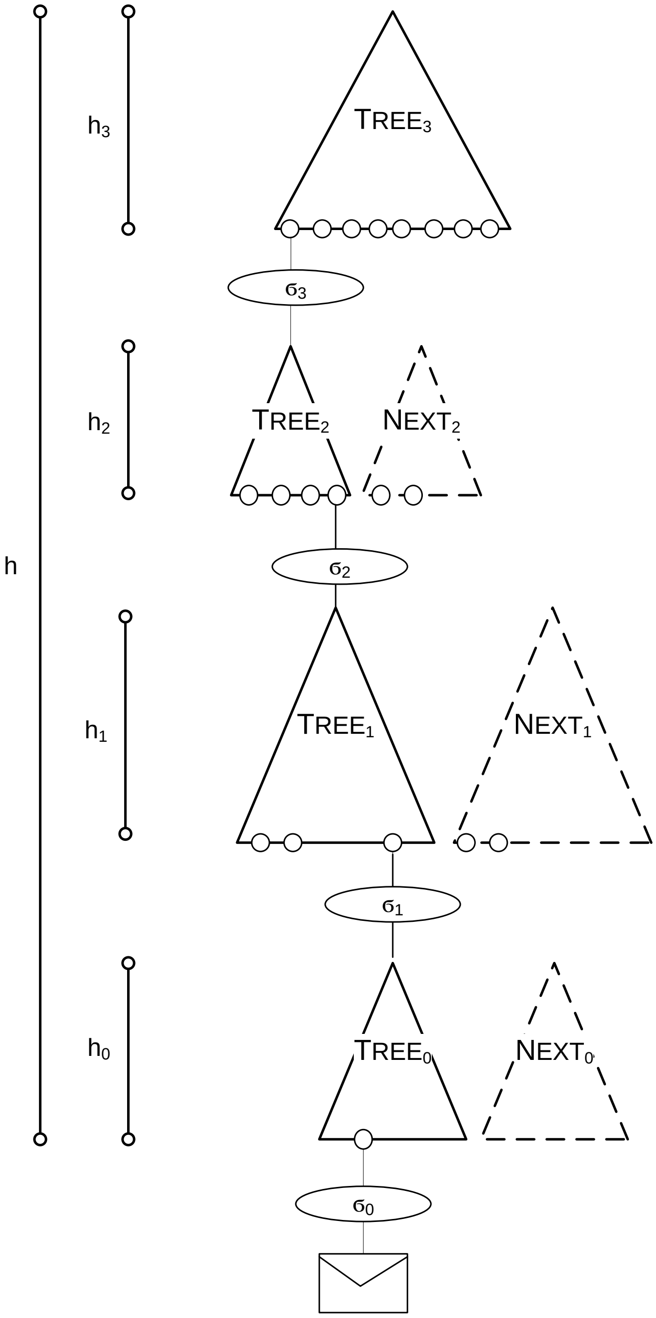
5 XMSS^{MT}– XMSS with Virtually Unlimited Signature Capacity

In this section we show how to extend XMSS in a way that one key pair can be used to sign a virtually unlimited number of messages. We call the scheme Multi Tree XMSS (XMSS^{MT}). Another important benefit of XMSS^{MT} is that key generation time is reduced from $\mathcal{O}(2^h)$ to $\mathcal{O}(2^{h/d})$ for some integer parameter d that can be chosen almost freely within $[1, h-1]$. The construction is based on the idea of tree chaining introduced in [BGD$^+$06]. We also apply and improve the distributed signature generation technique proposed in [BDK$^+$07] to further decrease the worst case signing time. Again, we use W-OTS$^\$$ in our description. A variant using W-OTS$^+$ is obtained, applying the same changes as for XMSS. We start with a description of the construction in Section 5.1. Then we discuss its security in Section 5.2 and finally analyze its correctness and theoretical performance in Section 5.3. The contributions of this chapter were published as parts of [4, 12].

5.1 Multi Tree XMSS - XMSS^{MT}

In this section we introduce XMSS^{MT}. For a better understanding, we first give a brief description of the scheme. For XMSS^{MT} we extend the general concept of a hash based signature scheme as used for XMSS using many layers of trees (here and in the following we use tree synonym for a XMSS key pair, as it better depicts the concept). Roughly speaking, we build a certification tree of XMSS key pairs. The trees on the lowest layer are used to sign the messages whereas the trees on higher layers are used to sign the roots of the trees on the layer below. The public key contains only the root of the tree on the top layer. A signature consists of all the signatures on the way to the highest tree. A graphical representation of the scheme is shown in Figure 5.1. In the following we describe the construction in detail, starting with the used parameters. Afterwards we describe the algorithms of the

Figure 5.1: A schematic representation of a XMSS^{MT} instance with four layers



scheme. The building blocks are the same as for XMSS described in the previous chapter.

Parameters. For security parameter $n \in \mathbb{N}$, XMSS^{MT} uses a pseudorandom function family $\mathcal{F}_n = \{\mathrm{F}_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$ and a second-preimage resistant hash function H, chosen uniformly at random from the family $\mathcal{H}_n = \{\mathrm{H}_K : \{0,1\}^{2n} \rightarrow \{0,1\}^n | K \in \{0,1\}^n\}$, like XMSS. Further parameters are the number of layers $d \in \mathbb{N}$, the binary message length m and one parameter set per layer. For a layer $0 \leq \delta \leq d-1$ a parameter set contains the tree height $h_\delta \in \mathbb{N}$, the BDS parameter $k_\delta \in \mathbb{N}$ with the restrictions $k_\delta < h_\delta$ and $h_\delta - k_\delta$ is even and the Winternitz parameter $w_\delta \in \mathbb{N}, w_\delta \geq 2$. To enable improved distributed signature generation we require $(h_{\delta+1} - k_{\delta+1})/2 + 3 \leq 2^{h_\delta - k_\delta + 1} - 2$ for $0 \leq \delta < d-1$, as well as $(h_0 - k_0)/2 \geq d-1$. Let $h = \sum_{\delta=0}^{d-1} h_\delta$, a XMSS^{MT} key pair can be used to sign 2^h messages of m bits. These parameters are publicly known.

Key Generation. The XMSS^{MT} key generation algorithm takes as input all of the above parameters. First, the $\max_{0\leq\delta\leq d-1}\{h_\delta + \lceil\log \ell_\delta\rceil\}$ bitmasks and the value X used by W-OTS$^\$$ are chosen uniformly at random where ℓ_δ denotes the parameter ℓ for W-OTS on layer δ. The same bitmasks and X are used for all layers. Then, the root of the first XMSS tree on each layer is computed. This is done in an ordered way, starting from layer 0. For the tree $\textsc{Tree}_\delta$ on layer δ the initial state of $\textsc{FsGen}$, $S_{0,\delta}$ is chosen uniformly at random and a copy of it is stored as part of the secret key SK. The tree is constructed the same way as in the XMSS key generation algorithm to compute $\textsc{Root}_\delta$. When $\textsc{Root}_\delta$, $0 \leq \delta < d-1$, is computed, it is signed using the first W-OTS key pair of $\textsc{Tree}_{\delta+1}$, which is computed next. This signature $\sigma_{\delta+1}$ can be extracted while $\textsc{Tree}_{\delta+1}$ is generated and hence does not need any additional computation. Then $\sigma_{\delta+1}$ is stored as part of SK. If the highest layer $d-1$ is reached, $\textsc{Root}_{d-1}$ is stored in the public key PK. During the computation of $\textsc{Root}_\delta$, the state of the BDS algorithm $\mathsf{State}_{\mathrm{BDS},\delta}$ is initialized as for XMSS.

Finally, the data structures for the next trees are initialized: For the next tree $\textsc{Next}_\delta$ on each layer $0 \leq \delta < d-1$ a $\textsc{FsGen}$ state $S_{n,\delta}$ is chosen uniformly at random and a new $\textsc{TreeHash}$ stack $\mathsf{Stack}_{next,\delta}$ is initialized. Also storage for a BDS state $\mathsf{State}_{\mathrm{BDS},n,\delta}$ is reserved. Summing up, SK consists of the states $(S_{0,\delta}, \mathsf{State}_{\mathrm{BDS},\delta}), 0 \leq \delta \leq d-1$ and the $d-1$ signatures σ_δ, $0 < \delta \leq d-1$. Additionally, it contains $d-1$ $\textsc{FsGen}$ states $S_{n,\delta}$, $d-1$ $\textsc{TreeHash}$ stacks $\mathsf{Stack}_{next,\delta}$ and $d-1$ BDS states $\mathsf{State}_{\mathrm{BDS},n,\delta}$ for the next trees $\textsc{Next}_\delta$ on layer $0 \leq \delta < d-1$. The public key PK consists of the $\max_{0\leq\delta\leq d-1}\{h_\delta + \lceil\log \ell_\delta\rceil\}$ bitmasks, the value X and $\textsc{Root}_{d-1}$.

Signature generation. The signature generation algorithm takes as input an m bit message M, the secret key SK, and the index i, indicating that this is the ith message signed with this keypair. The signature generation algorithm consists of two phases. First, M is signed. A XMSS^{MT} signature $\Sigma = (i, \sigma_0, \mathsf{Auth}_0, \sigma_1, \mathsf{Auth}_1, \ldots, \sigma_{d-1}, \mathsf{Auth}_{d-1})$ contains the index i, the W-OTS signature σ_0 on the message M, the corresponding authentication path for $\textsc{Tree}_0$ and the W-OTS signatures on the roots of the currently used trees together with the corresponding authentication paths. The only thing that has to be computed is σ_0 — the W-OTS signature on message M using the ith W-OTS key pair on the lowest layer. All authentication paths and the W-OTS signatures on higher layers are already part of the current secret key.

The second phase is used to update the secret key. Therefore BDS is initialized with $\mathsf{State}_{\mathrm{BDS},0}$ and receives $(h_0 - k_0)/2$ updates. If not all of these updates are needed to update $\mathsf{State}_{\mathrm{BDS},0}$, i.e. all scheduled node computations are finished and there are still updates left, the remaining updates are used for the upper trees. On the upper layers, not only the BDS state has to be updated. While one leaf is used, one leaf in the next tree must be computed, i.e. the root computation has to receive one update. Moreover, $h_\delta - k_\delta$ $\textsc{FsGen}$ states must be updated. This means that remaining updates from layer zero are first used to update $\mathsf{State}_{\mathrm{BDS},1}$. If all scheduled node computations in $\mathsf{State}_{\mathrm{BDS},1}$ are finished, one update is used for the root computation of $\textsc{Next}_1$. The next update is used for the $\textsc{FsGen}$ states. If layer 1 does not need any more updates, the remaining updates are forwarded to layer 2 and so on, until either all updates are used or all tasks are done. Finally, one leaf of the next tree is computed, i.e. the root computation for the next tree on layer 0 receives one update.

A special case occurs if $i \mod 2^{h_0} = 2^{h_0} - 1$. In this case, the last W-OTS key pair of the current $\textsc{Tree}_0$ was used. This means that for the next signature a new tree is needed on every layer δ with $i \mod 2^{h_\delta} = 2^{h_\delta} - 1$. For all these layers, $\mathsf{Stack}_{next,\delta}$ already contains the root of $\textsc{Next}_\delta$. So, $\textsc{Tree}_{\delta+1}$ is used to sign $\textsc{Root}_\delta$. Each signature is counted as one update. In case not all updates are needed, remaining updates are forwarded to the first layer that did not get a new tree. In SK, $\mathsf{State}_{\mathrm{BDS},\delta}$, S_δ, and Σ_δ are replaced by the newly computed data. Afterwards, new data structures for the next tree on layer δ are initialized and used to replace the ones in SK. Finally, the signature SIG, the updated secret key SK and $i + 1$ are returned.

Signature verification. The signature verification algorithm takes as input a signature $\Sigma = (i, \sigma_0, \mathsf{Auth}_0, \sigma_1, \mathsf{Auth}_1, \ldots, \sigma_{d-1}, \mathsf{Auth}_{d-1})$, the message M and the public key PK. To verify the signature, roughly speaking the XMSS verification algorithm is first used for the signature of the lowest layer. The root value obtained in the last step of the XMSS signature verification is then used as message for the next layer and so on, until a root node for layer $d-1$ is obtained. This value is then compared to the value in the public key.

More specifically, σ_0 and M are used to compute the corresponding W-OTS public key. The corresponding leaf $N_{0,j}$ of $\textsc{Tree}_0$ is constructed and used together with Auth_0 to compute the path $(P_0, \ldots, P_{h_0})$ to the root of $\textsc{Tree}_0$, where $P_0 = N_{0,j}$, $j = i \mod 2^{h_0}$ and

$$P_c = \begin{cases} \mathrm{H}((P_{c-1}||\mathsf{Auth}_{c-1,0}) \oplus B_c), & \text{if } \lfloor j/2^c \rfloor \equiv 0 \mod 2 \\ \mathrm{H}((\mathsf{Auth}_{c-1,0}||P_{c-1}) \oplus B_c), & \text{if } \lfloor j/2^c \rfloor \equiv 1 \mod 2 \end{cases}$$

for $0 \leq c \leq h_0$. This process is then iterated for $1 \leq \delta \leq d-1$, using the output of the last iteration $P_{h_{\delta-1}} = \textsc{Root}_{\delta-1}$ as message and σ_δ, Auth_δ and $j = \left\lfloor i/2^{\sum_{b=0}^{\delta-1} h_b} \right\rfloor \mod 2^{h_\delta}$. If the output of the last iteration $P_{h_{d-1}}$ equals the root value contained in PK, $\textsc{Root}_{d-1}$, the signature is assumed to be valid and the algorithm returns 1. In any other case it returns 0.

5.2 Security

In this section we show that XMSS^{MT} is secure. More specifically, we prove the following theorem:

Theorem 5.1. *If $\mathcal{H}_n$ is a second-preimage resistant hash function family and $\mathcal{F}_n$ a pseudorandom function family, then XMSS^{MT} is a forward secure signature scheme.*

The reasoning is as follows. From a theoretical point of view, XMSS^{MT} can be seen as a certification tree using XMSS. The proof is a straightforward combination of the result from the last section about the forward security of XMSS and a result from [MMM02]. For this reason we only sketch the proof for the special case of XMSS^{MT}.

Proof Sketch. Let's look at XMSS^{MT} the following way. Ignoring all algorithmic improvements, XMSS^{MT} uses d differently parameterized versions of XMSS. Denote them as XMSS$_0, \ldots,$ XMSS$_{d-1}$. Now one instance of XMSS$_{d-1}$ is used to sign the roots of $2^{h_{d-1}}$ instances of XMSS$_{d-2}$, one per leaf, and so on. The leaves of the XMSS$_0$ instances are used to sign the messages.

Now assume there exists an adversary $\mathcal{A}$ that breaks the forward security of XMSS^{MT} with probability $\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^{MT}(1^n,2^h)}(\mathcal{A})$ running in time t. Then we can construct an oracle machine $\mathcal{M}^{\mathcal{A}}$ that breaks the forward security of one out of the d XMSS versions used. As input $\mathcal{M}^{\mathcal{A}}$ receives one public key for every used XMSS version XMSS$_\delta$ and access to a corresponding signature oracle Sign_δ, for $0 \leq \delta \leq d$ that automatically performs a key update after every signature query. $\mathcal{M}^{\mathcal{A}}$ places the challenge instance for a layer at a random position on this layer. If a challenge instance is required to sign a message or a root, the corresponding signing oracle is used. For the remaining XMSS instances $\mathcal{M}^{\mathcal{A}}$ generates the key pairs and uses them to sign. When $\mathcal{A}$ indicates to **breakin**, $\mathcal{M}^{\mathcal{A}}$ hands over all the secret key information. If necessary, $\mathcal{M}^{\mathcal{A}}$ itself indicates a **breakin** for some of the challenge instances. If $\mathcal{A}$ returns a valid forgery $(M, \Sigma = (i, \sigma_0, \mathsf{Auth}_0, \sigma_1, \mathsf{Auth}_1, \ldots, \sigma_{d-1}, \mathsf{Auth}_{d-1}))$, $\mathcal{M}^{\mathcal{A}}$ starts a verification. Assume, $\mathcal{M}^{\mathcal{A}}$ stored the Root of all trees used before **breakin** was indicated. According to the forward security notion, M is a new message. Now during verification $\mathcal{M}^{\mathcal{A}}$ compares the Root on every layer with the one it generated itself or the Root in the public key of the challenge instance, respectively. If the two Root nodes match on layer δ, $\mathcal{M}^{\mathcal{A}}$ found a forgery for the XMSS instance on this layer. With probability $2^{-\sum_{j=\delta+1} d-1 h_j}$ the current instance on layer δ is the challenge instance and $\mathcal{M}^{\mathcal{A}}$ succeeds. Otherwise $\mathcal{M}^{\mathcal{A}}$ aborts.

Analyzing $\mathcal{M}^{\mathcal{A}}$, we get that $\mathcal{M}^{\mathcal{A}}$ runs in time $t' \leq t + \mathsf{t}_{\mathsf{Kg}} + 2^h \mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$. With probability $\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^{MT}(1^n,2^h)}(\mathcal{A})$ $\mathcal{A}$ outputs a forgery. Then we got d mutually exclusive cases case$_\delta$, for $0 \leq \delta < d$, i.e. the Root nodes match for the first time on layer δ. As stated above, if the Root nodes match for the first time on layer δ, with probability $2^{-\sum_{j=\delta+1}^{d-1} h_j}$ the current instance on layer δ is the challenge instance. So for each case$_\delta$ we get a bound

$$\mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}(1^n,2^{h_\delta})}\left(\mathcal{M}^{\mathcal{A}}\right) \geq 2^{-\sum_{j=\delta+1}^{d-1} h_j} \mathrm{Succ}^{\mathrm{FSSIG}}_{\mathrm{XMSS}^{MT}(1^n,2^h)}(\mathcal{A})$$

and hence

$$\begin{aligned} &\mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}^{MT}(1^n, 2^h); t, 1\right) \\ &\qquad \leq \sum_{0=\delta}^{d-1} 2^{\sum_{j=\delta+1}^{d-1} h_j} \cdot \mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}(1^n, 2^{h_\delta}); t, 1\right). \end{aligned}$$

Replacing $\mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}(1^n, 2^{h_\delta}); t, 1\right)$ by the results from the last section leads the claimed result. □

5.3 Analysis

In the following we provide an analysis of XMSS^{MT}. We first discuss its algorithmic correctness, i.e. we show that all node computations, root computations, root signatures and FsGen updates finish in time. Afterwards, we discuss key and signature sizes as well as theoretical runtimes of the algorithms.

5.3.1 Correctness

In the following we show that the $(h_0 - k_0)/2$ updates per signature suffice to finish all computations in time. A tree on layer δ needs $(h_\delta - k_\delta)/2$ updates to continue the node computations for upcoming authentication paths and one update for the computation of the root of the next tree $\textsc{Next}_\delta$, to sign the root of $\textsc{Next}_{\delta-1}$ and to update the FsGen states, respectively. This makes $3 + (h_\delta - k_\delta)/2$ updates. Please recall that we require $(h_{\delta+1} - k_{\delta+1})/2 + 3 \leq 2^{h_\delta - k_\delta + 1} - 2$ for $0 \leq \delta < d - 1$ as well as $(h_0 - k_0)/2 \geq d - 1$. The proof works by induction over layer δ.

Base case ($\delta = 0$): Per construction a tree on layer 0 receives $(h_0 - k_0)/2$ updates for the node computations per signature. The remaining tasks (computation of $\textsc{Next}_0$, FsGen updates and generating one signature) are executed without explicitly using updates and instead of a root node a message is signed. This is the same as if it would receive the three additional updates and use them to fulfill these tasks.

Inductive step ($\delta - 1 \Rightarrow \delta$): While one OTS signature on layer δ is used as part of the XMSS^{MT} signature, the OTS key pairs of a whole tree $\textsc{Tree}_{\delta-1}$ on layer $\delta - 1$ are used. Now, per assumption $\textsc{Tree}_{\delta-1}$ receives $3 + (h_{\delta-1} - k_{\delta-1})/2$ updates per signature. Three of these updates are directly used to update the computation of the root of $\textsc{Next}_{\delta-1}$, to update the FsGen states and to sign the next root node on layer $\delta - 2$ (or the message, if $\delta = 1$). So there remain $(h_{\delta-1} - k_{\delta-1})/2$ updates for BDS per signature. This makes a total of $2^{h_{\delta-1}}(h_{\delta-1} - k_{\delta-1})/2$ updates for the whole tree.

For all authentication paths of $\textsc{Tree}_{\delta-1}$, the BDS algorithm has to compute all right nodes of $\textsc{Tree}_{\delta-1}$ that are on a height $< h_{\delta-1} - k_{\delta-1}$ once. The only exceptions are the two first right nodes on every level as these nodes are stored during initialization. The number of required updates for $2 \leq k_{\delta-1} \leq h_{\delta-1}$ is

$$\sum_{j=0}^{h_{\delta-1}-k_{\delta-1}-1} (2^{h_{\delta-1}-j-1} - 2)2^j = (h_{\delta-1} - k_{\delta-1})2^{h_{\delta-1}-1} - 2^{h_{\delta-1}-k_{\delta-1}+1} + 2.$$

Hence, there are

$$(h_{\delta-1} - k_{\delta-1})2^{h_{\delta-1}-1} - (h_{\delta-1} - k_{\delta-1})2^{h_{\delta-1}-1} + 2^{h_{\delta-1}-k_{\delta-1}+1} - 2 = 2^{h_{\delta-1}-k_{\delta-1}+1} - 2$$

unused updates, which are forwarded to $\textsc{Tree}_\delta$. As we require $(h_\delta - \delta)/2 + 3 \leq 2^{h_{\delta-1}-k_{\delta-1}+1} - 2$, $\textsc{Tree}_\delta$ receives the necessary number of updates per signature.

There would still occur a problem, if the number of updates per signature were smaller than $d-1$. The reason is that this would mean that the roots of the new trees on different layers could not always be signed using the updates of the last signature before the change. In this case the private storage would grow, as we would need some intermediate storage for the new signatures. This is the reason why the second condition ($(h_0 - k_0)/2 \geq d-1$) is needed.

Please note that the condition $(h_\delta - \delta)/2 + 3 \leq 2^{h_{\delta-1}-k_{\delta-1}+1} - 2$ implies that we do not allow $k_{\delta-1} = h_{\delta-1}$. The reason is that even if all nodes in the current tree are stored, we need at least two updates per signature to update the computation of the root of $\textsc{Next}_\delta$ and to sign the next root node on layer $\delta - 1$. As we do not need to run BDS, there is no need to update the FsGen states. If now $k_{\delta-1} = h_{\delta-1}$, we cannot guarantee that layer $\delta - 1$ leaves enough unused updates to execute these necessary tasks. For parameters with $k_{\delta-1} = h_{\delta-1}$ a more detailed analysis over all layers of a key pair would be required.

5.3.2 Theoretical Performance

First we look at the sizes. A signature contains the index and d pairs of W-OTS signature and authentication path. Hence a signature takes $24+n\cdot\sum_{\delta=0}^{d-1}(\ell_\delta+h_\delta)$ bits, assuming we reserve three bytes for the index. The public key contains the bitmasks, X and $\textsc{Root}_{d-1}$. Thus, the public key size is $n\cdot(\max_{0\leq\delta\leq d-1}\{h_\delta+\lceil\log \ell_\delta\rceil\}+2)$ bits. The secret key contains a single FsGen state as well as one BDS state which in turn consists of $2(h_\delta - k_\delta)$ FsGen states and no more than $(3h_\delta + \lfloor\frac{h_\delta}{2}\rfloor - 3k_\delta - 2 + 2^{k_\delta})$ tree nodes per currently used tree $\textsc{Tree}_\delta$ [BDS08]. In addition, it contains the $d-1$ W-OTS signatures $\sigma_1, \dots, \sigma_{d-1}$ which have a total size of $n \cdot \sum_{\delta=1}^{d-1}\ell_\delta$ bits and the data structures for upcoming trees. These data structures do not require a full BDS state, as only those arrays in $\mathsf{State}_{\mathrm{BDS},n,\delta}$ are needed that are filled during initialization. Moreover, the space to store the k top levels of nodes can be shared with the corresponding space in $\mathsf{State}_{\mathrm{BDS},\delta}$. Thus, these structures require only $(h_\delta - k_\delta + 1)$ FsGen states (one for building the tree and the remaining as storage for the BDS state) and no more than $3h_\delta - k_\delta + 1$ tree nodes per $\textsc{Next}_\delta$, $0 \leq \delta < d-1$. The total secret key size is

$$n \cdot \left(\sum_{\delta=0}^{d-1}\left[\left(5h_\delta + \left\lfloor\frac{h_\delta}{2}\right\rfloor - 5k_\delta - 2 + 2^{k_\delta}\right) + 1\right] + \sum_{\delta=0}^{d-2}(\ell_{\delta+1} + 4h_\delta - 2k_\delta + 2)\right) \text{bits.}$$

For the runtimes we only look at the worst case times and get the following. During key generation, the first tree on each layer has to be computed. This means, that each of the 2^{h_δ} W-OTS key pairs has to be generated, including the execution of the

PRGs. Furthermore, to obtain the root, the leaves of the trees have to be computed as well as all internal nodes of the tree. If key generation generates the trees in order, starting from the first one, the W-OTS signatures on the roots of lower trees need no additional computation as the signature can be extracted while the corresponding W-OTS key pair is generated. The key generation time is

$$\mathsf{t}_{\mathsf{Kg}} \leq t_{\mathrm{H}} \left(\sum_{\delta=0}^{d-1} \left(2^{h_\delta} (\ell_\delta + 1) \right) \right) + t_{\mathrm{F}} \left(\sum_{\delta=1}^{d-1} \left(2^{h_\delta} (2 + \ell_\delta (w_\delta + 1)) \right) \right),$$

where t_{H} and t_{F} denote the runtimes of one evaluation of H and F, respectively. During one call to Sign, a W-OTS signature on the message must be generated, including generation of the key $(t_{\mathrm{F}}(2 + \ell_0(w_0 + 1)))$, the BDS algorithm receives $(h_0 - k_0)/2$ updates, one leaf of the next tree on layer 0 must be computed and the BDS algorithm updates $h_0 - k_0$ upcoming seeds $(t_{\mathrm{F}}(h_0 - k_0))$. The worst case signing time is bounded by

$$\mathsf{t}_{\mathsf{Sign}} \leq \max_{\delta \in [0, d-1]} \left\{ \begin{array}{l} t_{\mathrm{H}} \left(\frac{h_0 - k_0 + 2}{2} \cdot (h_\delta - k_\delta + \ell_\delta) + h_0 \right) \\ + t_{\mathrm{F}} \left(\frac{h_0 - k_0 + 4}{2} \cdot (\ell_\delta (w_i + 1)) + h_0 - k_0 \right) \end{array} \right\}.$$

Signature verification consists of computing d W-OTS public keys and the corresponding leafs plus hashing to the root. Summing up verification takes

$$\mathsf{t}_{\mathsf{Vf}} \leq \sum_{\delta=0}^{d-1} \left(t_{\mathrm{H}} \left(\ell_\delta + h_\delta \right) + t_{\mathrm{F}} \left(\ell_\delta w_\delta \right) \right)$$

in the worst case.

6 Choosing Optimal Parameters for XMSS*

In this chapter we show how to choose parameters for XMSS*, i.e. XMSS and XMSS^{MT}. More specifically we show how to select parameters that yield a provably secure instantiation on the one hand and result in provably optimal performance for a given use case on the other hand. Towards this end, we first show how to compute the security level of XMSS* for a given parameter set. Afterwards we show how to model parameter selection as a linear optimization problem. Here, we only discuss the more complicated case of XMSS^{MT}. The model for XMSS is obtained using the same techniques but the equations from Section 4.4. Finally, we provide optimal parameters for two exemplary use cases. The contributions of this chapter were published as parts of [2, 4, 12].

6.1 Security Level of XMSS*

In the following we show how to compute the security level of XMSS and XMSS^{MT} for given parameters. Security level here is used in the sense of [Len04][1]. This allows a comparison of the security of XMSS* with the security of a symmetric primitive like a block cipher for given security parameters. For example a security level of 128 bit is comparable to the security of AES with 128 bit keys. More specifically, we say that XMSS* has security level b if a successful attack on the scheme can be expected to require approximately 2^{b-1} evaluations of a hash function or block cipher on the average, following [Len04]. We detail the computation of the security level for the forward secure XMSS construction.

[1] The website `http://www.keylength.org` allows one to compute the security level for today's signature schemes. It also shows an estimation of how long a given security level is assumed to be secure according to [Len04].

6.1.1 Security Level of XMSS using W-OTS$^{\$}$

We now compute the security level of XMSS. Therefore we use the exact insecurity function given in equation 4.2. We can compute the security level, finding a lower bound for t such that $1/2 \leq \mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}; t, 1\right)$. According to the proof of Theorem 4.4, XMSS can only be attacked by attacking the second preimage resistance of $\mathcal{H}_n$ or the pseudorandomness of $\mathcal{F}_n$. Following the reasoning in [Len04], we only take into account generic attacks on $\mathcal{H}_n$ and $\mathcal{F}_n$.

For the insecurity of $\mathcal{H}(n)$ under generic attacks we assume $\mathrm{InSec}^{\mathrm{SPR}}\left(\mathcal{H}(n); t\right) = \frac{t}{2^n}$ which corresponds to a brute force search for second preimages. For the insecurity of $\mathcal{F}_n$ under generic attacks we assume that the best attack is a brute force key retrieval attack. Lemma 3.13 shows that if we assume $\mathcal{F}_n$ to be a PRF with security level n we get $\mathrm{InSec}^{\mathrm{PRF}}\left(\mathcal{F}_n; t, q\right) = \frac{t}{2^{n-log\kappa}} \cdot \left(\frac{1}{\kappa} - \frac{1}{2^n}\right)$ for the insecurity function and from Lemma 3.12 it follows that the number of key collisions $\kappa \leq 2$. Now, let $t' = t + t'' + \max\{\ell, w+1, 2\}$ where $t'' = 2^h \cdot t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + t_{\mathsf{Kg}}$. We compute the lower bound on t. The following bound holds for $t'' < \min\{2^{n-2h-5}, 2^{n-h-w-\log \ell w-4}\}) - \max\{\ell, w+1, 2\}$. We comment on the reasonableness of this bound after presenting the bit security.

$$\begin{aligned}
\frac{1}{2} &\leq \mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}; t, q=1\right)\\
&\leq 2^{2h+1}\frac{t'}{2^{n-\log\kappa}}\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)\\
&+ 2\cdot\max\left\{\begin{array}{l}\left(2^{h+\log\ell}-1\right)\cdot\frac{t'}{2^n},\\ 2^h\left(\frac{t'}{2^{n-\log\kappa}}\left(\frac{1}{\kappa}-\frac{1}{2^n}\right) + (\ell w\kappa^{w-1}\frac{1}{\left(\frac{1}{\kappa}-\frac{1}{2^n}\right)})\cdot\frac{t'}{2^{n-\log\kappa}}\left(\frac{1}{\kappa}-\frac{1}{2^n}\right)\right)\end{array}\right\}\\
&< 2^{2h+1}\frac{t'}{2^{n-1}}\left(\frac{1}{2}\right) + 2^{h+1}\left(\frac{t'}{2^{n-1}}\left(\frac{1}{2}\right) + \left(\ell w 2^{w-1}\right)\cdot\frac{t'}{2^{n-1}}\right)\\
&= \frac{t'}{2^{n-2h-1}} + \frac{t'}{2^{n-h-1}} + \frac{t'}{2^{n-h-w-\log\ell w-1}}\\
t' &> \frac{2^{n-h-2}}{2^h + 1 + 2^{w+\log\ell w}}\\
t' &> \frac{2^{n-h-2}}{2\max\{2^{h+1}, 2^{w+\log\ell w}\}}\\
t' &> \min\{2^{n-2h-4}, 2^{n-h-w-\log\ell w-3}\}
\end{aligned}$$

Now, using $t' = t + t'' + \max\{\ell, w+1, 2\}$ and the above condition on t'' we obtain

$$t > \min\{2^{n-2h-5}, 2^{n-h-w-\log\ell w-4}\}.$$

Therefore, the security level for XMSS is

$$b > n - h - 3 - \max\{h+1, w + \log \ell w\}\,.$$

The used bound on t'' is reasonable whenever a parameter set leads to a reasonable security level. This is the case as t'' is the time required to generate a key pair, use it to sign the maximum number of possible messages, and verify each of these signatures. Almost half of this time is required for key generation, a task that must be computable in reasonable time. Now, the bound says that this time must be smaller than the average runtime of a successful attack on the scheme, a task that should be impossible to do in reasonable time, minus $\max\{\ell, w+1, 2\}$. The result of the maximum must be much smaller than the gap between the runtimes as any possible outcome appears as a multiplicative factor in the theoretical formulas of all XMSS runtimes.

6.1.2 Security Level of XMSS using W-OTS$^+$

Now, we compute the security level of XMSS using W-OTS$^+$. The computation is almost the same as above. We have to replace the insecurity function for W-OTS by that for W-OTS$^+$ in the proofs to obtain the insecurity function for XMSS using W-OTS$^+$. Denote the function family used by W-OTS$^+$ by $\mathcal{G}_n$. Besides the generic attacks described above, we also need generic attacks against one-wayness and undetectability. We assume $\mathrm{InSec}^{\mathrm{OW}}(\mathcal{G}_n; t) = \frac{t}{2^n}$ which corresponds to a brute force search for preimages. For the insecurity regarding undetectability we assume $\mathrm{InSec}^{\mathrm{UD}}(\mathcal{G}_n; t) = \frac{t}{2^n}$ following [DSS05]. We again use an upper bound on t''. This time we have $t' = t + t'' + \max\{\ell, 2w, 2\}$ and we require $t'' < \min\{2^{n-2h-5}, 2^{n-h-\log \ell(w^2+w)-4}\}) - \max\{\ell, 2w, 2\}$, which holds for all reasonable parameters using the same argumentation as above. The resulting security level for XMSS is

$$b > n - h - 1 - \max\left\{h+3, \log(\ell 2w^2 + w)\right\}.$$

6.1.3 Security Level of XMSS^{MT}

Using the same reasoning as above, we can compute the security level of XMSS^{MT}. We present two lower bounds on the security level for XMSS^{MT}, one for the case of W-OTS$^\$$ and one for W-OTS$^+$. For similar bounds on the runtime of the algorithms as above, we can lower bound the security level b of XMSS^{MT}, using W-OTS$^\$$ by

$$b > \min_{0 \le \delta \le d-1} \left\{ n - h_\delta - \log d - (\sum_{j=\delta+1}^{d-1} h_j) - 3 - \max\{h_\delta - 1, w_\delta - \log(\ell_\delta w_\delta)\} \right\}.$$

Similarly, for the case of W-OTS$^+$ we obtain

$$b > \min_{0 \leq \delta \leq d-1} \left\{ n - h_\delta - \log d - (\sum_{j=\delta+1}^{d-1} h_j) - 1 - \max\left\{ h_\delta + 3, \log(\ell_\delta 2 w_\delta^2 + w) \right\} \right\}.$$

6.2 Optimization

Given the theoretical formulas for runtimes and sizes from Section 5.3.2, we now show how to use them to model the parameter selection problem as linear optimization problem. There are parameters which control different trade-offs. The BDS parameters $k_\delta \in \mathbb{N}$ control a trade-off between signature time and secret key size. The Winternitz parameters $w_\delta \in \mathbb{N}$ control a trade-off between runtimes and signature size. Finally, the number of layers d determines a trade-off between key generation and signature time on the one hand and signature size on the other hand. Moreover, there are the different tree heights δ that do not define any obvious trade-off, but influence the security as well as the performance of the scheme. The goal of the optimization is to choose these parameters. The function families $\mathcal{F}_n$ and $\mathcal{H}_n$ can be instantiated, either using a cryptographic hash function or a block cipher. Hence, the security parameter n is restricted to the output size of such functions. We choose 128 and 256 bit corresponding to AES and SHA-2 for our optimization, respectively.

Optimization Model. To find good parameter choices, we use linear optimization. In the following we discuss how we model the problem of optimal parameter choices as a linear optimization problem. As objective function of our problem we chose a weighted sum of all runtimes and sizes that should be minimized. Using the weights, it is possible to control the importance of minimizing a certain parameter and thereby using the model for different scenarios. We further allow to apply absolute bounds on the runtimes and sizes. The formulas for runtimes and sizes are modeled as constraints as well as the parameter restrictions and the formula for bit security from above. The input to the model are the runtimes of $\mathcal{F}$ and $\mathcal{H}$ for $n = 128$ and $n = 256$, the overall height h, the message length m and a value b as lower bound on the bit security.

As many of our initial constraints are not linear, we have to linearize all functions and restrictions. This is done using the generalized lambda method [Mor07]. In addition, we split the problem into sub problems each having some decision variables fixed. The optimization problem contains the parameters of the scheme (d, n, the $h_\delta, k_\delta, w_\delta$ for all layers) as decision variables which are determined by solving the optimization problem. Furthermore, the message length m_δ on each layer has to be modelled as a decision variable. Since solving the optimization problem takes much time and memory, we split the problem into sub problems by fixing the decision variables n and d. Therefore, we receive one sub problem for each combination of possible values of n and d. The resulting sub problems are solved independently and the best of their solutions is chosen as global solution of the original optimization problem.

The next step is to linearize the remaining sub problems by using the generalized lambda method. Therefore, we introduce a grid point for each possible combination of the remaining variables h, k, w and m on each layer δ. For each grid point we have a binary variable $\lambda_{h,k,w,m,\delta}$ which takes value 1 if the combination of h, k, w, m is chosen on layer δ. Otherwise, it takes value 0. Since we need one choice of h, k, w, m for each $\delta \in [0, d-1]$, d λ's must be chosen.

We use those lambdas to calculate the functions describing the problem. Thus, before optimizing we determine the values of the functions for each possible values of their variables. To make this feasible, we have to introduce bounds on the decision variables. We bound the tree height per layer by 24. As $k \leq h$ this bound also applies to k. For w we chose 255. These bounds are reasonable for the scenarios of the next section. For different scenarios they might have to be changed. Then, to model the needed space of signatures $\sum_{\delta=0}^{d-1} (\ell_\delta + h_\delta) \cdot n$, we formulate the constraint

$$\text{SpaceSig} == \sum_{\delta=0}^{d-1} \sum_{h=1}^{24} \sum_{k=1}^{24} \sum_{w=2}^{512} \sum_{m \in \{128,256\}} \lambda_{h,k,w,m,i} \cdot \underbrace{f_{\text{SpaceSig}}(w,h,m)}_{\text{pre}-\text{calculated}}$$

in the optimization model, where $f_{\text{SpaceSig}}(w,h,m) = (\ell + h)n$. Thus, SpaceSig gives the exact value of the needed space of signatures for the choice of lambda's and can be used in constraints and objective function.

To linearize a condition containing the maximum of some terms, such as the public key size $n \cdot (\max_{0 \leq \delta \leq d-1}\{h_\delta + \lceil \log \ell_\delta \rceil\} + 2)$, we write the following constraint:

$$\begin{aligned} &\forall \delta \in \{1, ..., d\} \\ &\text{SpacePK} \geq \sum_{h=1}^{24} \sum_{k=1}^{24} \sum_{w=2}^{512} \sum_{m \in \{128,256\}} \lambda_{h,k,w,m,\delta} \cdot (\underbrace{f_{\lceil \log \ell \rceil}(w,m)}_{\text{pre}-\text{calculated}} + h + 2)n. \end{aligned}$$

Hence, SpacePK gives the public key size for the choice of lambda's. This constraint pushes the value of SpacePK up high and due to the objective function the value will be pushed down as low as possible, so that in the end it takes the exact value.

6.3 Results

In this section we present optimal parameters for two exemplary use cases. To solve the optimization problem, we used the IBM Cplex solver [IBM] that implements the Simplex algorithm [Dan63] with some improvements. The linearization described in the last section is exact. Thus, there is no loss of information or error. Therefore, it can be proven that the solution found by linear optimization based on the Simplex algorithm is the best possible solution. In the following we present the results and compare them with the results for parameter sets proposed in [BDH11] and [HBB13]. We choose a message length of 256 bits for all use cases assuming that the message is the output of a collision resistant hash function. Moreover, we use 80 bits as lower bound for the provable bit security. This seems reasonable, as the used bit security represents a provable lower bound on the security of the scheme and is not related to any known attacks. We used the instantiations for $\mathcal{F}$ and $\mathcal{H}$ proposed in [BDH11] with AES and SHA2 for $n = 128$ and $n = 256$, respectively and measured the resulting runtimes on a Laptop with Intel(R) Core(TM) i5-2520M CPU @2.5 GHz and 8 GB RAM. We got $t_{\mathrm{F}} = 0.000225$ms and $t_{\mathrm{H}} = 0.00045$ms for $n = 128$ as well as $t_{\mathrm{F}} = 0.00169$ms and $t_{\mathrm{H}} = 0.000845$ms for $n = 256$. As W-OTS, we used W-OTS$^{\$}$. Furthermore we used less tight bounds on the bit security from [HBB13] and [BDH11] to achieve comparability.

The first use case we look at meets the requirements of a document or code signature. We assume that the most important parameters are signature size and verification time. We try to minimize them, while keeping reasonable bounds on the remaining parameters. We used the bounds $\mathsf{t_{Sign}} < 1000$ms, $\mathsf{t_{Vf}} < 1000$ms, $\mathsf{t_{Kg}} < 60$s, $\mathsf{sk} < 25$kB, $\mathsf{pk} < 1.25$kB, $\sigma < 100$kB and the weights $\mathsf{t_{Sign}} = 0.00000001, \mathsf{t_{Vf}} = 0.00090000$, $\mathsf{t_{Kg}} = 0.00000001, \mathsf{sk} = 0.00000001, \sigma = 0.99909996, \mathsf{pk} = 0.00000001$ for $h = 20$. We chose different weights for $\mathsf{t_{Vf}}$ and σ, because the optimization internally counts in bits and milliseconds. We set the weights such that 1ms costs the same as 1000 bit. The remaining weights are not set to zero but to $1.0e - 8$, the smallest possible value that we allow. This is necessary to ensure that our implementation of inequalities in the model works. This also ensures that within the optimal solutions regarding $\mathsf{t_{Vf}}$ and σ, the best one regarding the remaining parameters is chosen. It turns out that the optimization can be solved for $d \geq 2$. For $d = 1$ the bound on the key generation time cannot be achieved for the required height. If we relax this bound to be $\mathsf{t_{Kg}} < 600$s, i.e. 10 minutes, the problem can be solved for $n = 128$ using AES. For $d \geq 2$ this relaxation does not change the results. The optimal parameters for this setting are $n = 128, d = 2$, $h_0 = 17, k_0 = 5, w_0 = 5$ and $h_1 = 3, k_1 = 3, w_1 = 22$. For comparison we used a parameter set from [HBB13]

	Runtimes (ms)			Sizes (bit)		
Use case	t_{Kg}	$\mathsf{t}_{\mathsf{Sign}}$	t_{Vf}	σ	PK	SK
$UC1$ optimal	27251	1.65	0.36	21376	6144	25472
$UC1$ from [HBB13]	326	1.00	0.28	28288	4608	25856
$UC2$ optimal	166min	25.55	9.13	83968	13824	209152
$UC2$ from [BDK$^+$07]	98min	14.53	5.01	119040	13824	233472

Table 6.1: Runtimes and sizes for optimized parameters and parameters proposed in previous works.

that matches the bound on the bit security ($n = 128, d = 2, h_0 = h_1 = 10, k_0 = k_1 = 4, w_0 = w_1 = 4$). The resulting runtimes and sizes are shown in Table 6.1. The results show that it is possible to reduce the signature size by almost one kilo byte, changing the other parameters within their bounds and increasing the second important parameter, the verification time, by 0.08 milliseconds.

As a second use case we take a total tree height of 80 and aim for a balanced performance over all parameters. This use case corresponds to the use in a communication protocol. Again, we choose the weights such that 1ms costs the same as 1000 bit but this time we use the same weights for all runtimes and for all sizes. For comparison we use parameters from [BDK$^+$07] ($d = 4$, $h_0 = h_1 = h_2 = h_3 = 20$, $w_0 = 5$, $w_1 = w_2 = w_3 = 8$, $k_0 = k_1 = k_2 = k_3 = 4$). As they do not use a BDS parameter, we choose $k = 4$ on all layers. To make a fair comparison, we limited our optimization also to four layers. The optimal parameters returned by the optimization are $h_0 = h_1 = h_2 = h_3 = 20$, $w_0 = 14$, $w_1 = w_2 = w_3 = 24$ and $k_0 = k_1 = k_2 = 4$, $k_3 = 2$. The results are shown in Table 6.1. It turns out that again, by trading some runtime, the signature size can be significantly reduced.

7 XMSS* in Practice

To finally backup our claim regarding practicality of XMSS and XMSS^{MT}, we implemented the schemes to provide practical evidence. In this chapter we present runtimes of a C implementation of XMSS on a standard CPU as well as runtimes of XMSS^{MT} and XMSS on an of-the-shelf smart card and compare the results with those of other signature schemes. We also use the runtimes to show the influence of the different parameters on the practical performance of the schemes. We first generally discuss different possible implementations of the used function families in practice. Afterwards we first present the C implementation and end with the smart card implementation. The contributions of this chapter were published as parts of [2, 4, 7].

7.1 Implementing the Function Families

The implementation of XMSS$^+$ is straightforward besides the implementation of the used function families $\mathcal{F}_n$ and $\mathcal{H}_n$ as well as $\mathcal{G}_n$ when W-OTS$^+$ is used. We propose constructions based on hash functions and block ciphers for all three function families and argue why they are secure.

Instantiations using Hash Functions. First we discuss the hash function based constructions. We assume a hash function **Hash** that takes inputs of arbitrary length[2] and outputs n bit hash values. If we assume **Hash** is a secure cryptographic hash function, we can use **Hash** as the randomly chosen function from $\mathcal{H}_n$ and $\mathcal{G}_n$. To assume that **Hash** is a randomly chosen element of a hash function family, is common practice. This is the case because in theory we assume hash function families but in practice cryptographic hash functions are constructed as a single

[2] In practice the input length is mostly bounded by 2^{64} bits. This can be assumed to be virtually unlimited.

function [RS04]. Furthermore, cryptographically secure hash functions are assumed to be second-preimage resistant, one-way and undetectable.

The more complicated part is to implement $\mathcal{F}_n$. Some hash functions like SHA-3 come with a special PRF mode. If this is not the case, we propose a construction for any secure hash function that uses the Merkle-Darmgard (M-D) construction [Mer90b, Dam90]. The family $\mathcal{F}_n$ is constructed as follows. Given a hash function **Hash** with block length b and output size n that uses the M-D construction, we construct the function family $\mathcal{F}_n$ as

$$f_K(M) = \mathbf{Hash}(\mathbf{Pad}(K)||\mathbf{Pad}(M)),$$

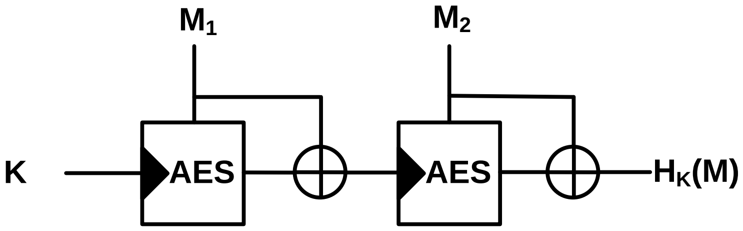
for key $K \in \{0,1\}^n$, message $M \in \{0,1\}^n$ and $\mathbf{Pad}(X) = (X||10^{b-|X|-1})$ for $|X| < b$.

We argue that it is reasonable to assume that this is a secure PRF if **Hash** is a secure cryptographic hash function. The assumptions we use are essentially those used for the security of HMAC using a practical hash function. In [BCK96a] it is assumed that the compression function of a good M-D hash function is a pseudorandom function family if it is keyed using the input. In [BCK96b], it is assumed, that the compression function of a good M-D hash function is a pseudorandom function family if keyed on the chaining input. Furthermore it is shown, that a fixed input length M-D hash function, keyed using the initialization vector (IV), is a pseudorandom function family for fixed length inputs. In our construction the internal compression function of **Hash** is evaluated twice: First on the IV and the padded key, second on the resulting chaining value and the padded message. Due to the pseudorandomness of the compression function when keyed on the message input, the first evaluation works as a pseudorandom key generation. As we have a fixed message length, the second iteration is a pseudorandom function family keyed using the IV input.

Instantiations using Block Ciphers. Now we present constructions using a block cipher $\mathrm{E}(K, M)$ with block and key length n bit. This is of special interest in case of AES, because many smart card crypto co-processors and also most of today's Intel processors provide hardware acceleration for AES. To implement $\mathcal{F}_n$ we use E without modification as the theoretical standard model for a block cipher is a pseudorandom permutation family. Therefore, we assume that for a secure block cipher this is indeed the case. To implement $\mathcal{H}$ we build a compression function using the Matyas-Meyer-Oseas (MMO) construction [MMO85] and iterate it using the M-D construction. This is shown in Figure 7.1. More specifically we compute $\mathrm{H}_K(M) = C_2$ for $M = M_1||M_2$, with

$$C_i = \mathrm{E}_{C_{i-1}}(M_i) \oplus M_i, \quad C_0 = K, \quad 0 < i \leq 2.$$

Figure 7.1: Construction of $\mathcal{H}$ using AES with the Matyas-Meyer-Oseas construction in M-D Mode.



In [BRS02] the authors give a black box proof for the security of the above compression function construction. There is no need to use M-D strengthening, as our domain has fixed size.

To construct $\mathcal{G}_n$ we also use the MMO construction. As $\mathcal{G}_n$ maps bit strings of length n to bit strings of the same length, we do not need a domain extension method like M-D. Hence, we only use one round of MMO and compute

$$G_K(M) = \mathrm{E}_K(M) \oplus M.$$

Remark 7.1. *Please note that we left the area of provable security in this section. All the proposed constructions are heuristic as the statements about the used hash functions and block ciphers are. While we cannot prove that the implementations are secure we argued why they are reasonable. One might look at this as our hardness assumptions being something like the used hash function is second-preimage resistant. For a hash function like SHA-2 this might be analyzed in the same detail as the hardness of some algebraic or number theoretic problem for given parameters.*

7.2 C Implementation

We now discuss our implementation of XMSS for general purpose CPUs. The implementation was done in C and it uses the OpenSSL library[3] for the implementation of hash functions and block ciphers. The implementation supports any hash function or block cipher supported by OpenSSL and any combination of these. For our measurements we used either AES or SHA2 to implement all function families. Table 7.1 shows our experimental results for XMSS with W-OTS$^{\$}$ on a computer with an Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz, 8GB RAM, and Intel AES-NI[4]. The displayed results are for the forward secure construction. The construction for

[3] http://www.openssl.org/

[4] http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni

Table 7.1: XMSS performance for $m = 256$ on a computer with an Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz and 8GB RAM. b denotes the bit security. For the chosen parameters, the bit security is the same for forward security and EU-CMA security. AES-NI and AES are used with 128 bit keys. We used standard SHA2 with 256 bit digests.

					Timings (ms)			Sizes (byte)		
Function	h	k	w	Keygen	Sign	Verify	Secret key	Public key	Signature	b
SHA2	10	4	4	868	3.47	0.43	1,604	1,188	4,580	229
SHA2	10	4	16	1,522	6.38	0.75	1,604	1,124	2,468	216
SHA2	10	4	64	3,925	16.67	1.97	1,604	1,060	1,764	167
SHA2	10	4	108	5,839	24.85	2.94	1,604	1,060	1,604	122
SHA2	16	4	4	54,180	5.96	0.44	2,660	1,572	4,772	220
SHA2	16	4	16	95,876	10.70	0.75	2,660	1,508	2,660	210
SHA2	16	4	64	247,494	27.83	1.95	2,660	1,444	1,956	161
SHA2	16	4	108	369,741	41.58	2.91	2,660	1,444	1,796	116
SHA2	20	8	4	879,010	6.09	0.45	10,404	1,828	4,900	212
SHA2	20	8	16	1,531,497	10.90	0.76	10,404	1,764	2,788	206
SHA2	20	8	64	3,991,598	28.54	1.98	10,404	1,700	2,084	157
SHA2	20	8	108	5,982,298	42.43	2.93	10,404	1,700	1,924	112
SHA2	20	4	4	868,647	7.62	0.44	3,364	1,828	4,900	212
SHA2	20	4	16	1,534,748	13.71	0.76	3,364	1,764	2,788	206
SHA2	20	4	64	4,012,157	35.60	1.97	3,364	1,700	2,084	157
SHA2	20	4	108	5,941,291	53.15	2.93	3,364	1,700	1,924	112
AES-NI	10	4	4	55	0.24	0.07	804	596	2,292	101
AES-NI	10	4	16	77	0.33	0.06	804	564	1,236	88
AES-NI	16	4	4	3,505	0.41	0.07	1,332	788	2,388	92
AES-NI	16	4	16	4,915	0.56	0.06	1,332	756	1,332	82
AES-NI	20	8	4	56,526	0.42	0.07	5,204	916	2,452	84
AES-NI	20	8	16	78,728	0.57	0.06	5,204	884	1,396	78
AES-NI	20	4	4	56,066	0.52	0.07	1,684	916	2,452	84
AES-NI	20	4	16	79,196	0.71	0.06	1,684	884	1,396	78
AES	10	4	4	129	0.49	0.11	804	596	2,292	101
AES	10	4	16	168	0.72	0.11	804	564	1,236	88
AES	16	4	4	7,500	0.84	0.11	1,332	788	2,388	92
AES	16	4	16	10,832	1.21	0.11	1,332	756	1,332	82
AES	20	8	4	120,433	0.85	0.11	5,204	916	2,452	84
AES	20	8	16	171,674	1.22	0.11	5,204	884	1,396	78
AES	20	4	4	119,736	1.06	0.11	1,684	916	2,452	84
AES	20	4	16	172,851	1.53	0.11	1,684	884	1,396	78
RSA 2048				-	3.08	0.09	≤ 512	≤ 512	≤ 256	95
DSA 2048				-	0.89	1.06	≤ 512	≤ 512	≤ 256	95
MSS-SPR (n=128, h=20)							2^{32}	960	8,512	98

EU-CMA security has slightly faster runtimes and the secret keys are $2(h-k) \cdot n$ bits smaller. The key pairs can be used to sign about 1,000 ($h = 10$), 65,000 ($h = 16$) or one million messages ($h = 20$). To show the effect of this limitation in practice, we give an example. If a key pair is used for one year, it can be used to sign about 3, 179, or 2872 messages a day, for $h = 10$, $h = 16$, and $h = 20$, respectively. The last column of the table shows the bit security of the configuration. Following the heuristic of Lenstra and Verheul [LV01] with the updated equations [Len04] the configurations with bit security 84 are secure until 2024. The configurations with a bit security of 100 and more are at least secure until 2048. Please note that these numbers are based on the provable security and not on the runtimes of possible attacks, which is the common practice and for example used for the security level of RSA and DSA. This would result in better values. For this reason we included also settings where the bit security is smaller than 80 bits. For RSA and DSA [Len04] provides an optimistic and a conservative estimate. The optimistic estimate says RSA and DSA with 2048 bit keys have 95 bits of security and are assumed to be secure until 2040. The conservative estimation says 90 bits of security and secure until 2033. The timings for RSA and DSA were taken using the OpenSSL `speed` command. As this does not provide timings for key generation, we had to leave this field blank.

The results show that XMSS is comparable to existing signature schemes. Only the key generation takes more time. This problem is solved by XMSS^{MT}. But even without tree chaining, key generation takes less then 100 minutes for $h = 20$ and $w = 108$. As key generation is an offline task that needs no user interaction, this might not be a problem in many cases. Moreover, the results show the different trade-offs. Using a larger w, the signature size shrinks while the runtimes increase. Unfortunately, also the bit security decreases for bigger w. This can be avoided using W-OTS$^+$. Using a larger k, the runtimes of signature generation and verification decrease while the secret key size increases. The choice of k has no influence on the bit security.

The results also show the influence of n by comparing AES ($n = 128$) and SHA2 ($n = 256$). AES is obviously much faster than SHA2 and keys as well as signatures using SHA2 are about twice the size of those for AES. The only drawback using AES is the significant decrease in the bit security. This can be solved using W-OTS$^+$ instead of W-OTS$^\$$. Last but not least, the use of AES is interesting, because many new CPUs come with hardware acceleration for AES. Our results show that using AES-NI results in a speed up of approximately 50 %.

The results for AES also show that most of the time is used for W-OTS. Looking at the signature verification times, there is no recognizable change, even if the height is doubled. Hence, the runtimes remain the same in case of AES or get even faster in case of SHA2 replacing W-OTS$^\$$ by W-OTS$^+$ as the instantiations of $\gg_n$ need at most the same number of evaluations of the underlying primitive. The additional xor operations will not be recognizable measuring milliseconds. Something else that might seem confusing is that in case of AES-NI verification for $w = 16$ is faster than for $w = 4$. For $w = 16$, ℓ is reduced from 133 to 67 while w is quadrupeled. On average, twice the number of evaluations of AES is needed to compute the W-OTS verification key. On the other hand, the number of nodes in the L-tree is halved and as hashing requires two evaluations of AES this reduces the final runtime.

The last row of table 7.1 shows the signature and key size for MSS-SPR [DOTV08]. To make the results from [DOTV08] comparable, we computed the signature and public key size for message length $m = 256$ bit, using their formulas. [DOTV08] does not provide runtimes, therefore we had to leave these fields blank. Comparing XMSS using SHA-256 and $w = 108$ with MSS-SPR shows that even for a slightly higher bit security we achieve a signature size of less than 25 % of the signature size of MSS-SPR. Moreover, the secret key of MSS-SPR is bigger. Although the authors of [DOTV08] mention the possibility to generate the secret key using a pseudorandom generator, this is not covered by their security proof. For the provided values a secret key of size $2^h \cdot mn$ is assumed. Anyhow, a secret key size compareable to that of XMSS is possible using the pseudorandom key generation described in this work.

7.3 Smart Card Implementation

In this section we present a smart card implementation of XMSS and XMSS^{MT} with W-OTS$^\$$. First we give a description of our implementation. Then we present our results and compare the performance of XMSS, XMSS^{MT}, RSA and ECDSA. At the end of the section we discuss an issue regarding the non-volatile memory (NVM).

Implementation Details. For the implementation we use as smart card an Infineon SLE78 CFLX4000PM offering 8 KB RAM and 404 KB NVM. Its core consists of a 16-bit CPU running at 33 MHz. Besides other peripherals, it provides a True Random Number Generator (TRNG), a symmetric and an asymmetric crypto co-processor. We use the hardware accelerated AES implementation of the card to implement the function families $\mathcal{F}$ and $\mathcal{H}$ using the block cipher based constructions from above. All random inputs of the schemes are generated using the TRNG. We implemented XMSS and XMSS^{MT} with two layers ($d = 2$). We also restricted w to powers of two, which speeds up and simplifies the implementation.

Table 7.2: Results for XMSS and XMSS^{MT} ($d = 2$) for message length $m = 256$ on an Infineon SLE78. We use the same k and w for both trees. b denotes the security level in bits. The value in parentheses denotes the security level using W-OTS$^+$. The signature times are worst case times.

				Timings (ms)			Sizes (byte)			
Scheme	h	k	w	KeyGen	Sign	Verify	Secret key	Public key	Signature	b
XMSS$^+$	16	2	4	5,600	106	25	3,760	544	3,476	94 (97)
XMSS$^+$	16	2	8	5,800	105	21	3,376	512	2,436	90 (96)
XMSS$^+$	16	2	16	6,700	118	22	3,200	512	1,892	81 (94)
XMSS$^+$	16	2	32	10,500	173	28	3,056	480	1,588	65 (93)
XMSS$^+$	20	4	4	22,200	106	25	4,303	608	3,540	90 (93)
XMSS$^+$	20	4	8	22,800	105	21	3,920	576	2,500	86 (92)
XMSS$^+$	20	4	16	28,300	124	22	3,744	576	1,956	77 (90)
XMSS$^+$	20	4	32	41,500	176	28	3,600	544	1,652	61 (89)
XMSS	10	4	4	14,600	86	22	1,680	608	2,292	101
XMSS	10	4	16	18,800	100	17	1,648	576	1,236	88
XMSS	16	4	4	925,400	134	23	2,448	800	2,388	92
XMSS	16	4	16	1,199,100	159	18	2,416	768	1,332	82

Results. Tables 7.2 and 7.3 show the runtimes of our implementation with different parameter sets. We use the same k and w for both trees. The last column shows the security level for the given parameter sets. Following the updated heuristic of Lenstra and Verheul [Len04] the configurations with a security level of 81 (85, 86) bits are secure until the year 2019 (2025, 2026). Again, please note that these numbers represent a lower bound on the provable security level. A successful attack would still require an adversary to either find a second-preimage in a 128 bit hash function or to launch a successful key retrieval attack on AES 128. This would result in 128 bit security for all parameter sets. In Table 7.2, the signature time is the worst case time over all signatures of one key pair. The secret key size in the table differs from the values we would obtain using the theoretical formulas from chapter 5. This is because it includes all data that has to be stored on the card to generate signatures, including the bitmasks and X.

We used parameter sets with two heights. A key pair with $h = 16$ allows to generate more than $65,000$, one with $h = 20$ to generate more than one million signatures. Assuming a validity period of one year, this corresponds to seven signatures per day and two signatures per minute, respectively. The runtimes show, that XMSS^{MT} key generation can be done on the smart card in practical time. For all but one used parameter set, the key generation time is below 30 seconds. The times for signature generation and verification are all below 200 ms and 30 ms, respectively. The size

Table 7.3: Results for XMSS$^+$ for message length $m = 256$ on an Infineon SLE78 for different values of k. We use the same k and w for both trees. The table shows the worst case signing times, as well as the average case times

				Timings (ms)			Size (byte)
Scheme	h	k	w	KeyGen	Sign (w.c.)	Sign (avg.c.)	Secret key
XMSS$^+$	16	0	16	6,700	133	96	3,312
XMSS$^+$	16	2	16	6,700	118	96	3,200
XMSS$^+$	16	4	16	6,700	97	83	3,232
XMSS$^+$	16	6	16	7,000	95	67	4,352
XMSS$^+$	16	8	16	8,000	94	53	10,112

of the secret key is around four kilo bytes and signatures are around two kilo bytes, while the public keys are around 500 bytes. Increasing the tree height for XMSS almost doubles key generation time. For XMSS^{MT} the key generation time is almost doubled if one increases the height by two, as this means that the height of each internal tree is increased by one.

The results show that we can reduce the signature size by increasing the Winternitz parameter w. The behavior of the implementation reflects the theory. The factor for the reduction of the W-OTS signature size is only logarithmic in w. The increase of the runtime is negligible for small w. This can be explained by the following. While the length of the single function chains increases, the number of chains decreases. For $w > 16$ the increase of the runtime becomes almost linear. So from this point of view, $w = 16$ seems to be a good choice. On the other hand, the provable security level also decreases almost linearly in w. This problem can be solved using W-OTS$^+$ instead of W-OTS$^\$$. In Table 7.2 we show the security level for W-OTS$^+$ in parentheses. The values show that using W-OTS$^+$ we can achieve a security level of more then 90 bits for all presented parameter sets but one.

Table 7.3 shows two things. On the one hand, it is possible to decrease the average signing time spending more storage for the secret key state, by increasing k. This is what one assumes given the theory. On the other hand, the worst case signing time can only be reduced up to a certain limit. For the given parameters this limit is 94ms, the worst case signing time, when both trees are completely stored. These 94ms are mainly caused by the write operations, when one key pair on the lower layer is finished. While all the computations are done in previous rounds, the data structures for the next lower layer key pair have to be copied to the data structure for the current lower layer key pair. Furthermore the new data structures for the next lower layer key pair must be initialized. Choosing $k = 4$ seems to be the most reasonable choice for $h = 16$.

Comparison. The last rows of Table 7.2 show the results for XMSS. The results show that XMSS key generation can be done on the smart card but is impractical as it already takes more than 15 minutes for $h = 16$. Increasing the height by one almost doubles the runtime of key generation. Generating a key with XMSS^{MT} is already for $h = 16$ almost 200 times faster than with XMSS. While XMSS^{MT} signature generation is slightly faster for comparable parameters, verification is faster for XMSS. The faster key generation is paid for by slightly bigger secret keys and signatures, while the XMSS^{MT} public keys are smaller, because of the reused bitmasks.

XMSS^{MT} seems to be the better choice for a smart card implementation and thus we compare it with RSA 2048 and ECDSA 256 on the same smart card. The key generation performance of XMSS^{MT} is similar to RSA 2048, which needs on average 11 seconds but slower than ECDSA 256 (95ms). Signature generation is comparable to RSA 2048 (190ms) and ECDSA 256 (100ms). Only verification takes slightly longer than with RSA 2048 (7ms) but it is faster than with ECDSA 256 (58ms). The security level of RSA 2048 and ECDSA 256 is 95 and 128 bits, respectively. In contrast to the security level shown in Table 7.2, these numbers are not based on a security proof but on the best known attacks. As mentioned above, the security level of XMSS^{MT} is 128 bit, when we only assume the best known attacks.

NVM. The changing key presents a challenge for the implementation of XMSS^{MT} and XMSS on smart cards. NVM is organized in sectors and pages. Due to physical limitations only complete pages can be written (erased and reprogrammed) at once. Furthermore, they wear out and cannot be programmed anymore after a certain number of write cycles, depending on the technology (about $500,000$ in our case). However, as write operations are distributed over all 33 physical pages of a sector, the complete available cycles are around 16.5 million per sector.

Generating a key takes only a few hundred write cycles but its state has to be updated after each signature step. Overall, one million available signatures require one million write cycles for the modification of the state. Using careful memory management, layout and optimization, we managed to keep the number of write cycles below five million for a key pair with $h = 20$, which is far below the 16.5 million available per sector. This includes key generation and all 2^{20} signatures. It should be noted that this affects only one NVM sector of the card. To use multiple keys, they can be placed in different sectors in order to preserve NVM quality.

8 Conclusion

In this work, we introduced XMSS and XMSS^{MT}, two forward secure signature schemes with minimal security assumptions. Towards this end we introduced two new OTS: W-OTS^{+} and W-OTS$^{\$}$. One of them has provably minimal security assumptions, while the other comes with a tighter security reduction and security assumptions that are conjectured to be minimal. We showed how to select optimal parameters for given use cases reaching a provable security level and how to implement the used function families. Moreover, we presented experimental data, which show that our schemes have a performance comparable to that of today's signature schemes. This is especially interesting as our schemes fulfill the stronger security notion of forward security. With this work we showed that it is possible to construct practical (forward secure) signature schemes using minimal security assumptions that can even be implemented on smartcards.

This work shows that our schemes are ready to be used in practice. They are favorable in many use cases, especially when it comes to mid- or long-term security, i.e. if the validity of a signature has to be verifiable for more than some seconds. This includes important use cases like certification authority certificate signing, document signatures, software updates and more. The reason is that in these cases the forward security guarantees that signatures remain valid after a key compromise without any additional measures. And key compromises happen, even in case of certification authorities [Com13, ho13a, ho13b].

While we were concerned with constructing the schemes and analyzing them, there are still open questions about how to use forward secure signature schemes in practice. It turns out that it is not straightforward to combine FSS with today's public key infrastructures. In a separate paper we make a first attempt in this direction [BHW^{+}13] but this is out of the scope of this work.

Another issue using forward secure schemes in practice is the use of a state. Forward secure signature schemes are necessarily stateful and so are XMSS and XMSS^{MT}. From a technical point of view, this should not pose a problem as we showed that even the non volatile memory on a smartcard that wears out over time can handle XMSS^{MT} key pairs. However, using a stateful signature scheme, the common bad habit of generating backup copies of signing keys must be prevented. But creating backups of signing keys should be prevented anyway, as it increases the probability of a key compromise. There are other strategies to handle the case of data loss, like keeping a different key pair with a valid certificate as backup. Anyway, users have to be aware of this requirement when using forward secure signature schemes.

Besides our main result, some of our smaller contributions are interesting on their own. W-OTS$^+$ is currently the most efficient hash-based OTS that guarantees strong unforgeability using standard assumptions. Our approach of choosing optimal parameters using linear optimization might also be interesting for other cryptographic schemes with a variety of parameters, e.g. lattice-based schemes. Our XMSS^{MT} smartcard implementation is the first implementation of a FSS on smartcards and shows the feasibility and practical usability of forward secure signature schemes.

Future Work There still exist some challenges that are related to this work. The first one, we already discussed above, is to use FSS in real application. Besides, there are also more theoretical questions. So far, our schemes are the only FSS that resist quantum computer aided attacks. All non-generic constructions are based on problems from number theory. It is an interesting challenge to construct non-generic FSS using security assumptions that are conjectured to resist quantum computers like assumptions from the field of lattice- or code-based cryptography.

Moreover, we showed that our schemes can be constructed from any one-way function, but this is only an existential relation. Regarding efficiency, we only showed that our schemes are efficient if the used second-preimage resistant hash function family and the pseudorandom function family are efficient. To show that our schemes are efficient if we start from any efficient one-way function, we would need efficient constructions of second-preimage resistant hash function families and pseudorandom function families from any one-way function. While great improvements were made in this field during the last decade [HHR$^+$10, HRV10], the constructions are still far from being efficient. It would be a great achievement to further improve the existing constructions.

Bibliography

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer Berlin / Heidelberg, 2000. Cited on page 10.

[AMN01] Michel Abdalla, Sara Miner, and Chanathip Namprempre. Forward-secure threshold signature schemes. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 441–456. Springer Berlin / Heidelberg, 2001. Cited on page 4.

[AR00] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129. Springer Berlin / Heidelberg, 2000. Cited on page 4.

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, July 2004. Cited on page 2.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 227–242. Springer Berlin / Heidelberg, 2004. Cited on page 10.

[BCK96a] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 1996. Cited on page 82.

[BCK96b] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of 37th Annual Symposium on Foundations of Computer Science*, pages 514–523. IEEE, 1996. Cited on page 82.

[BDE+11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In A. Nitaj and D. Pointcheval, editors, *Africacrypt 2011*, volume 6737 of *Lecture Notes in Computer Science*, pages 363–378. Springer Berlin / Heidelberg, 2011. Cited on page 11.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, volume 7071 of *Lecture Notes in Computer Science*, pages 117–129. Springer Berlin / Heidelberg, 2011. Cited on page 78.

[BDK+07] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 31–45. Springer Berlin / Heidelberg, 2007. Cited on pages 5, 11, 63, and 79.

[BDS08] Johannes Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, volume 5299 of *Lecture Notes in Computer Science*, pages 63–78. Springer Berlin / Heidelberg, 2008. Cited on pages 11, 14, 16, 56, and 70.

[BDS09] Johannes Buchmann, Erik Dahmen, and Michael Szydlo. Hash-based digital signature schemes. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 35–93. Springer Berlin Heidelberg, 2009. Cited on pages 11, 12, 13, 14, and 61.

[BGD+06] Johannes Buchmann, L. C. Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS - an improved Merkle signature scheme. In *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 349–363. Springer, 2006. Cited on pages 5, 11, and 63.

[BHW$^+$13] Johannes Braun, Andreas Hülsing, Alex Wiesmaier, Martin A.G. Vigil, and Johannes Buchmann. How to avoid the breakdown of public key infrastructures. In Sabrina Capitani di Vimercati and Chris Mitchell, editors, *Public Key Infrastructures, Services and Applications*, volume 7868 of *Lecture Notes in Computer Science*, pages 53–68. Springer Berlin Heidelberg, 2013. Cited on pages 2 and 91.

[BKN07] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for Merkle tree traversal. In Joaquim Filipe, Helder Coelhas, and Monica Saramago, editors, *E-business and Telecommunication Networks*, volume 3 of *Communications in Computer and Information Science*, pages 150–162. Springer Berlin Heidelberg, 2007. Cited on page 14.

[BM96] Daniel Bleichenbacher and Ueli M. Maurer. Optimal tree-based one-time digital signature schemes. In *STACS '96: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, pages 363–374, London, UK, 1996. Springer-Verlag. Cited on page 11.

[BM99] Mihir Bellare and Sara Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 786–786. Springer Berlin / Heidelberg, 1999. Cited on pages 4, 45, and 53.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burton Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 470–484. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0052256. Cited on page 42.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 103–118. Springer Berlin / Heidelberg, 2002. Cited on page 83.

[BY03] Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2003. Cited on pages 47, 55, and 56.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222. Springer Berlin / Heidelberg, 2004. Cited on page 10.

[CJMM03] Eric Cronin, Sugih Jamin, Tal Malkin, and Patrick McDaniel. On the performance, feasibility, and use of forward-secure signatures. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 131–144, New York, NY, USA, 2003. ACM. Cited on page 5.

[CK06] Jan Camenisch and Maciej Koprowski. Fine-grained forward-secure signature schemes without random oracles. *Discrete Applied Mathematics*, 154(2):175 – 188, 2006. Coding and Cryptography. Cited on page 4.

[Com13] Comodo. The Recent RA Compromise. `http://blogs.comodo.com/it-security/data-security/the-recent-ra-compromise/`, visited Sep 2013. Cited on page 91.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer Berlin / Heidelberg, 1990. Cited on page 82.

[Dan63] George B. Dantzig. *Linear Programming And Extensions.* Princeton University Press, 1963. Cited on page 78.

[DOTV08] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, volume 5299 of *Lecture Notes in Computer Science*, pages 109–123. Springer Berlin / Heidelberg, 2008. Cited on pages 4, 11, 42, 52, 60, and 86.

[DSS05] Chris Dods, Nigel Smart, and Martijn Stam. Hash based digital signature schemes. In Nigel Smart, editor, *Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 96–115. Springer Berlin / Heidelberg, 2005. Cited on pages 4, 11, 19, 27, and 75.

[EGM96] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996. Cited on page 19.

[ETS10] ETSI. XML advanced electronic signatures (XAdES). Standard TS 101 903, European Telecommunications Standards Institute, December 2010. Cited on page 2.

[ETS12] ETSI. CMS advanced electronic signatures (CAdES). Standard TS 101 733, European Telecommunications Standards Institute, March 2012. Cited on page 2.

[Gar05] L. C. Coronado García. On the security and the efficiency of the Merkle signature scheme. Technical Report Report 2005/192, Cryptology ePrint Archive - Report 2005/192, 2005. Available at http://eprint.iacr.org/2005/192/. Cited on page 11.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. Cited on pages 3, 9, 34, and 48.

[GKL88] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. In *SFCS '88: Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 12–24, Washington, DC, USA, 1988. IEEE Computer Society. Cited on page 19.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. Cited on page 10.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, New York, NY, USA, 2009. Cited on page 4.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual Symposium on the Theory of Computing*, pages 212–219, New York, 1996. ACM Press. Cited on page 2.

[HBB13] Andreas Hülsing, Christoph Busold, and Johannes Buchmann. Forward secure signatures on smart cards. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin Heidelberg, 2013. Cited on pages 78 and 79.

[HHR$^+$10] Iftach Haitner, Thomas Holenstein, Omer Reingold, Salil Vadhan, and Hoeteck Wee. Universal one-way hash functions via inaccessible entropy. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 616–637. Springer Berlin / Heidelberg, 2010. Cited on page 92.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28:1364–1396, March 1999. Cited on pages 3, 9, 28, 34, and 48.

[HM02] Alejandro Hevia and Daniele Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 191–196. Springer Berlin / Heidelberg, 2002. Cited on pages 4 and 11.

[ho13a] h online. Attack on Israeli Certificate Authority. `http://h-online.com/-1264008`, visited May 2013. Cited on page 91.

[ho13b] h online. Fake Google certificate is the result of a hack. `http://h-online.com/-1333728`, visited May 2013. Cited on page 91.

[HRV10] Iftach Haitner, Omer Reingold, and Salil Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 437–446, New York, NY, USA, 2010. ACM. Cited on page 92.

[IBM] IBM. IBM ILOG CPLEX Optimizer. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`. Accessed Januray 2013. Cited on page 78.

[IR01] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354. Springer Berlin / Heidelberg, 2001. Cited on page 4.

[JLMS03] Markus Jakobsson, Tom Leighton, Silvio Micali, and Michael Szydlo. Fractal Merkle tree representation and traversal. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 314–326. Springer Berlin Heidelberg, 2003. Cited on pages 11 and 14.

[KR03] Anton Kozlov and Leonid Reyzin. Forward-secure signatures with fast key update. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 241–256. Springer Berlin / Heidelberg, 2003. Cited on page 4.

[Kra00] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 108–115, New York, NY, USA, 2000. ACM. Cited on pages 4 and 53.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979. Cited on page 12.

[Len04] Arjen K. Lenstra. Key lengths. Contribution to The Handbook of Information Security, 2004. Cited on pages 5, 73, 74, 85, and 87.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 2001. Cited on page 85.

[Mer90a] Ralph Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer Berlin / Heidelberg, 1990. Cited on pages 4, 11, 12, 15, and 17.

[Mer90b] Ralph Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer Berlin / Heidelberg, 1990. Cited on page 82.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Lars Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 400–417. Springer Berlin / Heidelberg, 2002. Cited on pages 4 and 67.

[MMO85] Stephen Matyas, Carl Meyer, and Jonathan Oseas. Generating strong one-way functions with cryptographic algorithms. In *IBM Technical Disclosure Bulletin 27*, pages 5658–5659. IBM, 1985. Cited on page 82.

[Mor07] Susanne Moritz. *A Mixed Integer Approach for the Transient Case of Gas Network Optimization*. PhD thesis, TU Darmstadt, Februar 2007. Cited on page 76.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, page 43. ACM, 1989. Cited on page 2.

[RED$^+$08] Sebastian Rohde, Thomas Eisenbarth, Erik Dahmen, Johannes Buchmann, and Christof Paar. Fast hash-based signatures on constrained devices. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications*, volume 5189 of *Lecture Notes in Computer Science*, pages 104–117. Springer Berlin / Heidelberg, 2008. Cited on page 5.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394, New York, NY, USA, 1990. ACM Press. Cited on pages 2, 3, 4, 9, 28, and 48.

[RS04] Phillip Rogaway and Thomas 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, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer Berlin / Heidelberg, 2004. Cited on pages 7, 9, 48, and 82.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1994)*, pages 124–134. IEEE Computer Society Press, 1994. Cited on page 1.

[Son01] Dawn Xiaodong Song. Practical forward secure group signature schemes. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 225–234, New York, NY, USA, 2001. ACM. Cited on page 4.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EURO-*

CRYPT 2004, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554. Springer Berlin / Heidelberg, 2004. Cited on pages 11 and 14.

Wissenschaftlicher Werdegang

Juli 2010 - heute Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Professor Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt im Forschungsprojekt BU 630/19-1: "Beweisbar sichere, effiziente und langfristig sichere Variante des Merkle Signatur Verfahrens" der Deutschen Forschungsgemeinschaft (DFG).

Dezember 2007 - Juni 2010 Wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Sichere Informationstechnologie (SIT), Darmstadt.

Oktober 2001 - März 2008 Studium der Informatik (Diplom) an der Technischen Universität Darmstadt.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, August 2013