XMSS – A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions

Johannes Buchmann and Andreas Hülsing[⋆]
{buchmann,huelsing}@cdc.informatik.tu-darmstadt.de

Cryptography and Computeralgebra
Department of Computer Science
TU Darmstadt

Abstract. We present XMSS, the first signature scheme that is provably forward secure and efficient when instantiated with two secure and efficient function families: one second-preimage resistant and the other pseudorandom. The security requirements for XMSS are minimal, because the existence of the two function families is known to be necessary for the existence of digital signature schemes. Also, XMSS appears to be quantum-computer resistant as there are many cryptographic hash functions and pseudorandom functions that are believed to resist quantum computer attacks. We present experimental results that show that the performance of XMSS is comparable to RSA.

Keywords digital signatures, forward secure signatures, provable security, hash functions

1 Introduction

Digital signatures are among the most widely used cryptographic primitives. The signature schemes currently used in practice are RSA, DSA, and ECDSA. Their security depends on the security of certain trapdoor one-way functions which, in turn, relies on the hardness of factoring integers and computing discrete logarithms, respectively. However, it is unclear whether those computational problems remain hard in the future. In fact, it has been shown by Shor [Sho94] that quantum computers can solve them in polynomial time. In view of the importance of digital signatures new schemes must be found that are practical and resist quantum computer attacks. Another desirable feature of such new signature schemes is forward security which means that after a key compromise all previously generated signatures remain valid. This property is important, because with the increasing use of mobile devices attacks may become more likely. In addition, forward secure signature schemes allow to remove the need for timestamps in long-term scenarios and, as shown in [KCL06], can be used to implement robust public key infrastructures.

In this paper we present XMSS (eXtended Merkle Signature Scheme), the first signature scheme that has all the properties described above. We show, that XMSS, when instantiated with two function families $\mathcal{H}$ and F, is

provably forward secure in the standard model, provided $\mathcal{H}$ is second preimage resistant and F is pseudorandom,

efficient, provided that $\mathcal{H}$ and F are efficient. This claim is supported by experimental results that show that the performance of XMSS and RSA are comparable.

[⋆] Supported by grant no. BU 630/19-1 of the German Research Foundation (www.dfg.de).

The first assertion shows that the security requirements for XMSS are minimal. This follows from [Rom90], [RS04], [HILL99] and [GGM86] where the existence of a secure signature scheme is proved to imply the existence of a second preimage resistant hash function family and a pseudorandom function family (see Section 3).

The second assertion shows that XMSS is practical and quantum-computer resistant as there are many ways to construct very efficient (hash) function families that are believed to be second preimage resistant or pseudorandom, respectively, even in the presence of quantum computers. For example, cryptographic hash functions and block ciphers can be used to construct such families. There are also constructions that are based on hard problems in algebra or coding theory. The huge number of instantiations of XMSS guarantees the long-term availability of secure and efficient signature schemes. We also show that one obtains a scheme with a smaller secret key and faster signature generation, if forward security is replaced by existential unforgeability under chosen message attacks (EU-CMA).

The idea of hash-based signatures was introduced by Merkle [Mer90a]. 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. XMSS incorporates many of those ideas. 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 [Gol09] and [Rom90] are not practical and the security proof of MSS-SPR [DOTV08] does not cover the pseudo-random 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 very important as the signature size is considered the main drawback of hash-based signatures.

The practical forward secure signature schemes are based on number theory [AMN01, AR00, BM99, CK06, IR01, KR03, Son01]. In addition, there are two generic constructions [Kra00] and [MMM02]. They allow building forward secure signature schemes from any secure digital signature scheme. However, they do not have minimal security requirements and do not appear to be practical, because of their key size. We note that some of the techniques used for XMSScan be applied to the construction from [MMM02], to minimize its drawbacks.

All security proofs in this work are exact and in the standard model. This means, that we do not make use of any idealized assumptions about the used primitives. The proofs yield the exact relation between the security level of the used function families and the security of XMSS. This allows to determine parameter sets for a given security level.

XMSS, as presented in this paper, signs fixed length messages. The scheme can easily be extended to sign messages of arbitrary length using TCR hash and sign as proposed in [DOTV08]. This requires a target collision resistant hash function family. As target collision resistant hash function families are known to exist if one-way functions exist [Rom90], this modification does not affect the minimality of the security requirements.

An extended abstract of this work appeared as [BDH11]. This is the full version. In contrast to [BDH11], this work includes the full security proofs, a detailed discussion of the security level for a given parameter set, and new experimental runtimes.

The paper is organized as follows. In Section 2 we describe the construction of XMSS. Its security and forward security is discussed in Sections 3 and 4. The efficiency of XMSS is shown in Section 5. Section 6 presents our implementation and performance results.

2 The eXtended Merkle Signature Scheme XMSS

In this section we describe XMSS. Like the Merkle signature scheme [Mer90a] it uses a one-time signature scheme (OTS) that can only sign one message with one key. To overcome the limitation to one message per key, a binary hash tree is used to reduce the authenticity of many OTS verification keys to one XMSS public key. To minimize storage requirements, pseudorandom generators (PRG) are used. They generate the OTS signature keys as needed.

The parameters of XMSS are the following:

- $n \in \mathbb{N}$, the security parameter,
- $w \in \mathbb{N}, w > 1$, the Winternitz parameter,
- $m \in \mathbb{N}$, the message length in bits,
- $F_n = \{f_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^n\}$ a function family,
- $H \in \mathbb{N}$, the tree height, XMSS allows to generate 2^H signatures per keypair,
- h, a hash function, chosen randomly with the uniform distribution from the family $\mathcal{H}_n = \{h_K : \{0,1\}^{2n} \to \{0,1\}^n | K \in \{0,1\}^n\}$,
- $x \in \{0,1\}^n$, chosen randomly with the uniform distribution. The string x is used to construct the one-time verification keys.

Those parameters are publicly known.

We keep the following description of XMSS and its components short by including references to more detailed descriptions. We write log for $\log_2$.

Winternitz OTS As OTS we use the Winternitz OTS (W-OTS) first mentioned in [Mer90a]. We use a slightly modified version proposed in [BDE+11]. For $K, x \in \{0,1\}^n$, $e \in \mathbb{N}$, and $f_K \in F_n$ we define $f_K^e(x)$ as follows. We set $f_K^0(x) = K$ and for $e > 0$ we define $K' = f_K^{e-1}(x)$ and $f_K^e(x) = f_{K'}(x)$. In contrast to previous versions of W-OTS this is a (random) walk through the function family instead of an iterated evaluation of a hash function. This modification allows to eliminate the need for a collision resistant hash function family.

Also, define

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

The secret signature key of W-OTS consists of ℓ n-bit strings sk_i, $1 \le i \le \ell$ chosen uniformly at random. The public verification key is computed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (f_{\mathsf{sk}_1}^{w-1}(x), \ldots, f_{\mathsf{sk}_\ell}^{w-1}(x)),$$

with f^{w-1} as defined above.

W-OTS signs messages of binary length m. They are processed in base w representation. They are of the form $M = (M_1 \ldots M_{\ell_1})$, $M_i \in \{0, \ldots, w-1\}$. The checksum $C = \sum_{i=1}^{\ell_1}(w - 1 - M_i)$ in base w representation is appended to M. It is of length ℓ_2. The result is a sequence of ℓ base w numbers, denoted by $(b_1, \ldots, b_\ell)$. The signature of M is

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

It is verified by constructing $(b_1 \ldots, b_\ell)$ using M and checking

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

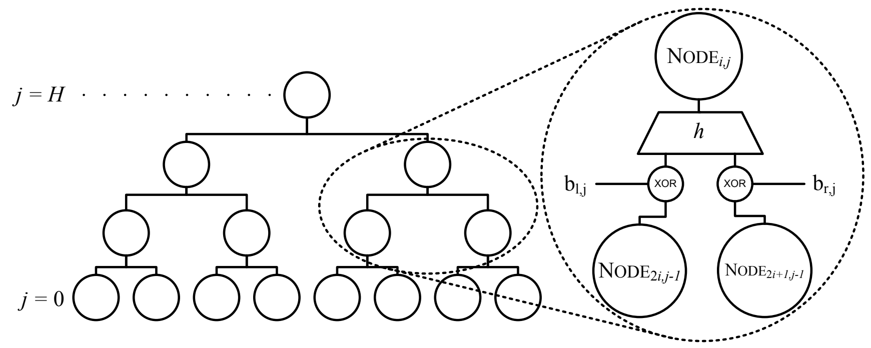
The sizes of signature, public, and secret key are ℓn. For more detailed information see [BDE+11].

XMSS Tree The XMSS tree is a modification of the Merkle Hash Tree proposed in [DOTV08]. It utilizes the hash function h. The XMSS tree is a binary tree. Denote its height by 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 $\textsc{Node}_{i,j}$, $0 \leq i < 2^{H-j}$. The construction of the leaves is explained below. Level j, $0 < j \leq H$, is constructed using a bitmask $(b_{l,j}||b_{r,j}) \in \{0,1\}^{2n}$ chosen uniformly at random. The nodes are computed as

$$\textsc{Node}_{i,j} = h((\textsc{Node}_{2i,j-1} \oplus b_{l,j})||(\textsc{Node}_{2i+1,j-1} \oplus b_{r,j}))$$

for $0 < j \leq H$. 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. Figure 1 shows the construction of the XMSS tree.

Fig. 1. The XMSS tree construction



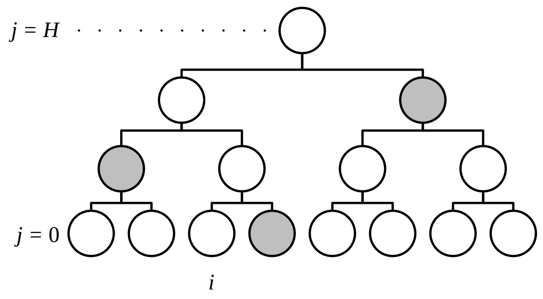
We explain the computation of the leaves of the XMSS tree. The XMSS tree is used to authenticate 2^H W-OTS verification keys, each of which is used to construct one leaf of the XMSS tree. The construction of the keys is explained at the end of this section. In the construction of a leaf another XMSS tree is used. It is called L-tree. The first ℓ leaves of an L-tree are the ℓ bit strings $(\mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$ from the corresponding verification key. As ℓ might not be a power of 2 there are not sufficiently many leaves. Therefore the construction is modified. A 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 as above but new bitmasks are used. The bitmasks are the same for each of those trees. As L-trees have height $\lceil \log \ell \rceil$, additional $\lceil \log \ell \rceil$ bitmasks are required. The XMSS public key PK contains all bitmasks and the root of the XMSS tree.

To sign the ith message, the ith W-OTS key pair is used. The signature $\mathsf{SIG} = (i, \sigma, \textsc{Auth})$ contains the index i, the W-OTS signature σ, and the authentication path for the leaf $\textsc{Node}_{0,i}$. It is the sequence $\textsc{Auth} = (\textsc{Auth}_0, \ldots, \textsc{Auth}_{H-1})$ of the siblings of all nodes on the path from $\textsc{Node}_{0,i}$ to the root. Figure 2 shows the authentication path for leaf i. To compute the authentication path we use the tree traversal algorithm from [BDS08] as it allows for optimal balanced runtimes using very little memory.

To verify the signature $\mathsf{SIG} = (i, \sigma, \textsc{Auth})$, the string $(b_0, \ldots, b_\ell)$ is computed as described in the W-OTS signature generation. Then the ith verification key is computed using the formula

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

Fig. 2. The authentication path for leaf i



The corresponding leaf $\textsc{Node}_{0,i}$ of the XMSS tree is constructed using the L-tree. This leaf and the authentication path are used to compute the path $(p_0, \dots, p_H)$ to the root of the XMSS tree, where $p_0 = \textsc{Node}_{0,i}$ and

$$p_j = \begin{cases} h((p_{j-1} \oplus b_{l,j}) || (\textsc{Auth}_{j-1} \oplus b_{r,j})), & \text{if } \lfloor i/2^j \rfloor \equiv 0 \mod 2 \\ h((\textsc{Auth}_{j-1} \oplus b_{l,j}) || (p_{j-1} \oplus b_{r,j})), & \text{if } \lfloor i/2^j \rfloor \equiv 1 \mod 2 \end{cases}$$

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

Signature Key Generation Now we describe the XMSS signature key generation for the EU-CMA-secure XMSS. In Section 4 we show how to change this construction to obtain the forward secure XMSS. The W-OTS secret signature keys are computed using a seed $\textsc{Seed} \in \{0,1\}^n$, the pseudorandom function family F_n, and the pseudorandom generator GEN which for $\lambda \in \mathbb{N}, \mu \in \{0,1\}^n$ yields

$$\mathsf{GEN}_\lambda(\mu) = f_\mu(1) || \dots || f_\mu(\lambda).$$

For $i \in \{1, \dots, 2^H\}$ the i-th W-OTS signature key is

$$\mathsf{sk}_i \leftarrow \mathsf{GEN}_\ell(f_{\textsc{Seed}}(i)).$$

The XMSS secret key contains $\textsc{Seed}$ and the index of the last signature i. The bit length of the XMSS public key is $(2(H + \lceil \log \ell \rceil) + 1)n$, an XMSS signature has length $(\ell + H)n$, and the length of the XMSS secret signature key is $< 2n$.

Tree Chaining So far, the key generation of this construction takes more time then that of other practical signature schemes. As key generation is an offline task, this might be acceptable. If it is inacceptable, it is possible to use the so called tree chaining technique, introduced in [BGD$^+$06] and improved in [BDK$^+$07]. This technique allows a trade off between signature size and the runtime of the key generation algorithm. The basic idea of this technique is to use many levels of hash based signature schemes. While the schemes on the lowest level are used to sign the messages, the schemes on higher levels are used to certify the trees on the level below. As it is straight forward to apply this technique to XMSS, we will not discuss it in this work to keep the analysis simple. For a detailed description we refer the reader to [BDS09].

3 Standard Security

In this section we show that XMSS is provably secure in the standard model and discuss the minimality of the assumptions we use. We first provide the needed preliminaries. We keep the notations of Section 2.

3.1 Preliminaries I

We write $m = \text{poly}(n)$ to denote that m is a function, polynomial in n. We call a function $\epsilon(n) : \mathbb{N} \rightarrow [0,1]$ negligible and write $\epsilon(n) = \text{negl}(n)$ 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 $n > n_c$. We write $x \xleftarrow{\$} X$ if x is chosen from X uniformly at random. In our proofs we measure algorithmic runtimes as the number of evaluations of functions from F_n and $\mathcal{H}_n$.

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 T. While the public key pk is fixed, the scheme operates on T different secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{T-1}$, 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. This is the case for XMSS, where a period ends after signing one message and the key update algorithm is automatically called after each signature creation. 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 T 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 1 (Key Evolving Signature Scheme). *A key evolving signature scheme is a quadruple of algorithms* $\text{Kes} = (\mathsf{Kg}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vf})$. *It is parameterized by a security parameter $n \in \mathbb{N}$ and the number of time periods $T \in \mathbb{N}, T = \text{poly}(n)$ and operates on the following finite sets with description length polynomial in n: The secret key space $\mathcal{KS} = \mathcal{KS}_0 \times \ldots \times \mathcal{KS}_{T-1}$ consisting of T 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:*

$\mathsf{Kg}(1^n, T)$*: The key generation algorithm is a probabilistic algorithm that on input of the security parameter $n \in \mathbb{N}$ in unary, and the number of time periods T, 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 is a possibly probabilistic algorithm that 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 $i < T - 1$ and $\mathsf{sk} \in \mathcal{KS}_i$. If $i \geq T - 1$ it outputs the empty string. In all other cases it returns* `fail`.

$\mathsf{Sign}(\mathsf{sk}, M, i)$*: The signature algorithm is a possibly probabilistic algorithm that 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 is a deterministic algorithm that 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* $M \in \mathcal{M}$*,* $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{Kg}(1^n, T)$*, and* $i < T$*:* $\mathsf{Vf}(M, (\mathsf{Sign}(M, \mathsf{KUpd}(\mathsf{sk}_0)^i), i), \mathsf{pk}) = 1$.

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 $T = 2^H$ 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 usual security model for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) introduced in [GMR88]. We translate it to the setting of key evolving signature schemes, using the following experiment. $\textsc{Kes}(1^n, T)$ denotes a key evolving signature scheme with security parameter n and number of periods T. 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 T times, once for each time period. The adversary algorithm A is given oracle access to an instance of a signature oracle Sign initialized with secret key sk_i and index i, denoted by $\mathsf{A}^{\mathsf{Sign}(\mathsf{sk}_i, \cdot, i)}$. 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 $\textsc{Out}$ variable allows the adversary to switch from the `cma` to the `forge` phase.

Experiment $\mathsf{Exp}^{\text{EU-CMA}}_{\textsc{Kes}(1^n,T)}(\mathsf{A})$

```
    $i \leftarrow 0$, state $\leftarrow$ null, out $\leftarrow$ null, $(\mathsf{sk}_0, \mathsf{pk}) \leftarrow \mathsf{Kg}(1^n, T)$
    While $i < T$ And out $\neq$ halt
        $(\texttt{out}, \texttt{state}) \leftarrow \mathsf{A}^{\mathsf{Sign}(\mathsf{sk}_i, \cdot, i)}(1^n, \texttt{cma}, \mathsf{pk}, \texttt{state})$
        i++; $\mathsf{sk}_i \leftarrow \mathsf{KUpd}(\mathsf{sk}_{i-1}, i)$
    $(M^\star, \sigma^\star, i^\star) \leftarrow \mathsf{A}(1^n, \texttt{forge}, \texttt{state})$
    If $\mathsf{Vf}(\mathsf{pk}, M^\star, (\sigma^\star, i^\star)) = 1$ And Sign was not queried for a signature on $M^\star$ Return 1
    Return 0
```

For the success probability of an adversary A in the above experiment we write

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

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

Definition 2 (EU-CMA). *Let* $n, q \in \mathbb{N}$*,* $t = \mathrm{poly}(n)$*,* $\textsc{Kes}$ *a key evolving signature scheme. Fix* $T \in \mathbb{N}$*. We call* $\textsc{Kes}$ EU-CMA*-secure, if* $\mathrm{InSec}^{\text{EU-CMA}}(\textsc{Kes}(1^n, T); t, q)$*, the maximum success probability of all possibly probabilistic adversaries* A*, running in time* $\leq t$*, making at most* q *queries to each instance of* Sign *in the above experiment, is negligible in* n*:*

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

For a Dss this translates to the initial notion. An OTS is a Dss that is EU-CMA secure for $q = 1$.

Function Families In the following let $n \in \mathbb{N}$, $m, k = \text{poly}(n)$, $\mathcal{H}_n = \{h_K : \{0,1\}^m \to \{0,1\}^n | K \in \{0,1\}^k\}$ a function family. We first define the success probability of an adversary A against second preimage resistance (SPR). Informally the adversary receives a second preimage challenge, consisting of a random preimage and a random function key, and has to come up with a collision for this preimage under the function identified by the key. More formally:

$$\begin{aligned}\text{Succ}^{\text{SPR}}(\mathcal{H}_n; \mathsf{A}) = &\mathsf{Pr}[\, K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m, M' \leftarrow \mathsf{A}(K, M) : \\ &(M \neq M') \wedge (h_K(M) = h_K(M'))] \qquad (1)\end{aligned}$$

Using this we define second preimage resistance of a function family.

Definition 3 (spr). *Let $t = \text{poly}(n)$, $\mathcal{H}_n$ as above. We call $\mathcal{H}_n$ second preimage resistant, if $\text{InSec}^{\text{SPR}}(\mathcal{H}_n; t)$, the maximum success probability of all possibly probabilistic adversaries A, running in time $\leq t$, is negligible in n:*

$$\text{InSec}^{\text{SPR}}(\mathcal{H}_n; t) \overset{def}{=} \max_{\mathsf{A}}\{\text{Succ}^{\text{SPR}}(\mathcal{H}_n; \mathsf{A})\} = negl(n)\,.$$

The second 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 $\mathcal{G}(m, n)$ of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$. The goal of the adversary is to distinguish both cases:

$$\begin{aligned}\text{Succ}^{\text{PRF}}(\mathcal{H}_n; \mathsf{A}) = \Big| &\mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{H}_n : \mathsf{A}^{\mathsf{Box}(\cdot)} = 1] \\ &- \mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{G}(m, n) : \mathsf{A}^{\mathsf{Box}(\cdot)} = 1]\Big|\,. \qquad (2)\end{aligned}$$

Now we can define pseudorandomness for a function family.

Definition 4 (prf). *Let $q \in \mathbb{N}, t = \text{poly}(n)$, $\mathcal{H}_n$ as above. We call $\mathcal{H}_n$ a pseudorandom function family, if $\text{InSec}^{\text{PRF}}(\mathcal{H}_n; t, q)$, the maximum success probability of all possibly probabilistic adversaries A, running in time $\leq t$, making at most q queries to Box, is negligible in n:*

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

Key Collisions In [BDE$^+$11] the authors define a key collision of a function family F_n as a pair of distinct keys (K, K') such that $f_K(M) = f_{K'}(M)$ holds for one or more messages $M \in \{0,1\}^m$. They denote the upper bound on the maximum number of keys that collide on one input value by κ, i.e. $\kappa = 1$ says that there exist no key collisions for F_n. For more information and a formal definition we refer the reader to [BDE$^+$11].

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

Experiment $\mathsf{Exp}_{G_n}^{\text{PRG}-1}(\mathsf{A})$	**Experiment** $\mathsf{Exp}_{G_n}^{\text{PRG}-0}(\mathsf{A})$
$x \xleftarrow{\$} \{0,1\}^n; c \leftarrow G_n(x)$	$c \xleftarrow{\$} \{0,1\}^b$
$g \leftarrow \mathsf{A}(c)$	$g \leftarrow \mathsf{A}(c)$
`Return` g	`Return` g

The success probability of an adversary A against the security of PRG G is defined as the ability of the adversary to distinguish both experiments:

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

Now we define secure pseudorandom generators.

Definition 5 (prg). *Let $n \in \mathbb{N}$, $t = \text{poly}(n)$, G_n as above. We call G_n a secure pseudorandom generator, if $\text{InSec}^{\text{PRG}}(G_n; t)$, the maximum success probability of all possibly probabilistic adversaries* A*, running in time $\leq t$, is negligible in n:*

$$\text{InSec}^{\text{PRG}}(G_n; t) \stackrel{def}{=} \max_{\mathsf{A}}\{\text{Succ}^{\text{PRG}}(G_n; \mathsf{A})\} = negl(n).$$

3.2 XMSS is Existentially Unforgeable under Chosen Message Attacks

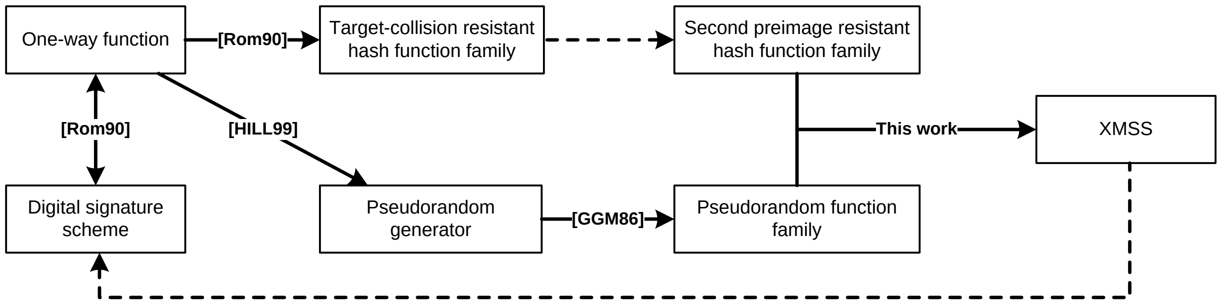
Now, we prove XMSS secure in the standard model and discuss some implications of this result. We prove the following Theorem:

Theorem 1. *If $\mathcal{H}_n$ is a second preimage resistant hash function family and F_n a pseudorandom function family, then XMSS is existentially unforgeable under chosen message attacks.*

Before we give the proof of Theorem 1, 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. Also 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 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 3.

Now we give the proof of Theorem 1. 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.

Fig. 3. 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 primitiv that fulfills A also fulfills B.



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

Proof (of Theorem 1). First we look at the key generation algorithm Kg in more detail. Kg uses the PRG $\mathsf{GEN}_\lambda(\mu) = f_\mu(0)||\dots||f_\mu(\lambda-1)$ 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 $\mathsf{GEN}_{2^H}(\textsc{Seed})$ and $\textsc{Seed}$ is the XMSS secret key. We show that GEN_λ is a secure PRG if the used function family is pseudorandom.

Claim 2. *Let $n, \lambda \in \mathbb{N}, \mu \in \{0,1\}^n$, $F_n = \{f_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^n\}$ be a pseudorandom function family with insecurity function $\mathrm{InSec}^{\mathrm{PRF}}(F_n; t, q)$. Then $\mathsf{GEN}_\lambda : \{0,1\}^n \to \{0,1\}^{\lambda n}$,*

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

is a PRG with insecurity function

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

Proof (of Claim 2). For the sake of contradiction assume there exists an adversary A distinguishing the output of GEN_λ from a uniformly random λn bit string. Then we can build an oracle machine M^{A} that given access to A distinguishes F_n from $\mathcal{G}(n,n)$. M^{A} queries Box for the outputs for $0,\dots,\lambda-1$ and hands the concatenation of the results to A. Then M^{A} simply forwards A's output. M^{A} succeeds with the same probability as 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 $2^H n$ bits for the 2^H W-OTS key pairs of XMSS.

Claim 3. *Let $n, n', q, t, T \in \mathbb{N}$, $G_n : \{0,1\}^n \to \{0,1\}^{\lambda n}$ be a PRG that stretches n-bit random input to λn-bit pseudo-random output with insecurity function $\mathrm{InSec}^{\mathrm{PRG}}(G_n; t)$ and let $\textsc{Kes} = (\mathsf{Kg}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vf})$ be a key evolving signature scheme with insecurity function $\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'}, T); t, q\right)$ that needs λn bits of random input for key generation. 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*

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

where $t' = t + \mathsf{t}_{\mathsf{Kg}^\star} + Tt_{\mathsf{KUpd}^\star} + q\mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$.

The proof for the above claim is based on the idea, that we can use any adversary against the scheme with pseudorandom key generation to attack the original scheme, as the format of the inputs is the same. In case of the original scheme we obtain an upper bound on the success probability of an adversary using the upper bound from the assumption for the success probability of any adversary against the original scheme. Hence, if an adversary has a higher success probability against the scheme with pseudorandom key generation than against the original scheme, we can use such an adversary to distinguish between a bit string produced by the PRG and a random bit string. We use the bit string to generate a key pair for the signature scheme and run the adversary on this key pair. If the adversary succeeds, it is more likely that the bit string was produced by the PRG, than that it was chosen at random.

Proof (of Claim 3). We want to find a bound on the success probability of any adversary 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 $\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'}, T); t, q\right)$. Given such an adversary, we can build a oracle machine M^{A} distinguishing the output of G_n from random λn-bit strings as described in algorithm 1.

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

Algorithm 1 M^{A}

```
Input: Security parameter $n$ and challenge $c \in \{0,1\}^{\lambda n}$
Output: $g \in \{0,1\}$

1. compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^{n'}, T)$ using $c$ as the randomness of $\mathsf{Kg}^{\star}$
2. out ← null, state ← null, $i \leftarrow 0$;
3. While $i < T$ And out ≠ halt
   (a) run (out, state) ← $\mathsf{A}^{\mathsf{Sign}=M}(1^n, \texttt{cma}, \mathsf{pk}, \texttt{state})$
   (b) If A queries Sign in time period $i$ Then answer up to $q$ queries using $sk_i$
4. If $(M^{\star}, \sigma^{\star}, i^{\star}) \leftarrow \mathsf{A}(1^n, \texttt{forge}, \texttt{state})$ is a valid forgery Then Return $g = 1$
5. Else Return $g = 0$
```

Now we calculate the success probability of M^{A}. If M^{A} is in $\mathsf{Exp}_{G_n}^{prg-1}$, c is a pseudorandom output of G_n. Hence, A succeeds with probability $\mathrm{Succ}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'}, T); \mathsf{A}\right)$ by definition and we get

$$\mathsf{Pr}\left[\mathsf{Exp}_{G_n}^{prg-1}(M^{\mathsf{A}}) = 1\right] = \mathrm{Succ}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'}, T); \mathsf{A}\right)$$

If M^{A} is in $\mathsf{Exp}_{G_n}^{prg-0}$, c is chosen uniformly at random. In this case A succeeds with probability $\leq \mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'}, T); t, q\right)$. Otherwise A would be a forger for $\textsc{Kes}$ that running in time t succeeds with probability greater than $\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'}, T); t, q\right)$, which would contradict the assumption. So we get

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

Altogether this leads to

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

and therefore

$$\begin{aligned}&\mathrm{Succ}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'}, T); \mathsf{A}\right)\\ &\qquad \leq \mathrm{InSec}^{\text{PRG}}\left(G_n; t + \mathsf{t}_{\mathsf{Kg}} + T\mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}\right) + \mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'}, T); t, q\right).\end{aligned}$$

As this holds for any adversary A running in time $\leq t$, making at most q queries to each instance of Sign we get

$$\begin{aligned}\mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}^{\star}(1^{n'}, T); t, q\right) &\leq \mathrm{InSec}^{\text{PRG}}\left(G_n; t + \mathsf{t}_{\mathsf{Kg}} + T\mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}\right)\\ &\quad + \mathrm{InSec}^{\text{EU-CMA}}\left(\textsc{Kes}(1^{n'}, T); t, q\right)\end{aligned}$$

□

In [BDE+11] it is shown that the insecurity function for the EU-CMA-security of W-OTS is

$$\mathrm{InSec}^{\text{EU-CMA}}\left(\text{W-OTS}(1^n, T = 1); t, q = 1\right) \leq (\ell^2 w^2 \kappa^{w-1} \frac{1}{\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)}) \cdot \mathrm{InSec}^{\text{PRF}}\left(F_n; t', q = 2\right)$$

where $t' = t + t_{\mathsf{Sign}} + t_{\mathsf{Kg}} + t_{\mathsf{Vf}}$ and κ denotes the upper bound on the number of key collisions in F_n.

In [DOTV08] the authors give an exact security proof for the XMSS-Tree construction, combined with any OTS. Using W-OTS as OTS, we obtain the following insecurity function for the EU-CMA-security of the XMSS-Tree construction:

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

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 3 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 3 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

$$\begin{aligned}
&\mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}(1^n, T = 2^H); t, q = 1\right) \\
&\qquad \leq \mathrm{InSec}^{\mathrm{PRF}}\left(F_n; (t' + 2^H), q = 2^H\right) \\
&\qquad + 2 \cdot \max \left\{ \begin{array}{l} (2^{H+\log \ell} - 1) \cdot \mathrm{InSec}^{\mathrm{SPR}}\left(\mathcal{H}_n; t'\right), \\ 2^H \Big(\mathrm{InSec}^{\mathrm{PRF}}\left(F_n; (t' + \ell), q = \ell\right) \\ \qquad + (\ell^2 w^2 \kappa^{w-1} \frac{1}{\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)}) \cdot \mathrm{InSec}^{\mathrm{PRF}}\left(F_n; (t'), q = 2\right) \Big) \end{array} \right\}
\end{aligned} \tag{3}$$

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

3.3 Security Level of XMSS

Given equation 3, which is an exact version of Theorem 1, we can compute the security level in the sense of [Len04]. This allows a comparison of the security of XMSS with the security of a symmetric primitive like a block cipher for given security parameters. Following [Len04], 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 functions from F_n and $\mathcal{H}_n$. We can compute the security level, finding a lower bound for t such that $1/2 \leq \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}\ ; t, q = 1\right)$. According to the proof of Theorem 1, XMSS can only be attacked by attacking the second preimage resistance of $\mathcal{H}_n$ or the pseudorandomness of F_n. Following the reasoning in [Len04], we only take into account generic attacks on $\mathcal{H}_n$ and 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 F_n under generic attacks we assume that the best attack is a brute force key retrieval attack. In [BDE+11] the authors show, that if we assume F_n to be a PRF with security level n we get $\mathrm{InSec}^{\mathrm{PRF}}\left(F_n; t, q\right) = \frac{t}{2^{n-log\kappa}} \cdot \left(\frac{1}{\kappa} - \frac{1}{2^n}\right)$ for the insecurity function and $\kappa \leq 2$ for the number of key collisions. Now, let $t'' = t' - t$. We compute the lower bound on t. The following bound holds under the condition that $t'' 2^{w+2\log \ell w} < 2^{n-H-4} - 2^\ell$. This condition holds for most reasonable choices of parameters, including those proposed in Section 6. Otherwise, given a specific choice of parameters it is easy to compute the security level from scratch.

$$\begin{aligned}
\frac{1}{2} &\leq \mathrm{InSec}^{\text{EU-CMA}}\left(\mathrm{XMSS}\ ; t, q = 1\right) \\
&\leq \frac{t + t'' + 2^H}{2^{n-\log \kappa}} \left(\frac{1}{\kappa} - \frac{1}{2^n}\right) \\
&+ 2 \cdot \max \left\{ \begin{array}{l} (2^{H+\log \ell} - 1) \cdot \frac{t+t''}{2^n}, \\ 2^H \left(\frac{t+t''+\ell}{2^{n-\log \kappa}} \left(\frac{1}{\kappa} - \frac{1}{2^n}\right) + (\ell^2 w^2 \kappa^{w-1} \frac{1}{\left(\frac{1}{\kappa} - \frac{1}{2^n}\right)}) \cdot \frac{t+t''}{2^{n-\log \kappa}} \left(\frac{1}{\kappa} - \frac{1}{2^n}\right) \right) \end{array} \right\} \\
&\leq \frac{t + t'' + 2^H}{2^n} + \frac{t + t'' + \ell}{2^{n-H-1}} + \frac{t + t''}{2^{n-H-w-2\log \ell w-1}} \\
&\leq \frac{t + t''}{2^{(n-H-w-2\log \ell w-2)}} + 2^{-(n-H)} + 2^{-(n-H-1-\ell)}
\end{aligned}$$

Now, using the above condition we obtain

$$t \leq 2^{n-H-4-w-2\log(\ell w)}$$

Therefore, the security level for XMSS is $b \geq n - H - 3 - w - 2\log(\ell w)$.

4 Forward Security

Given the above result we can go even further. In [And97] Anderson introduced the idea of forward security for signature schemes (FSSIG) which was later formalized in [BM99]. It says that even after a key compromise all signatures created before remain valid. Obviously, this notion is only meaningful for key evolving signature schemes that change their secret key over time. From an attack based point of view this translates to: If an attacker learns the current secret key sk_i, she is still unable to forge a signature under a secret key sk_j, $j < i$. This is a desirable property, especially in the context of long term secure signatures, as it allows to remove the need for timestamps and an online trusted third party.

In this section we show that XMSS is forward secure if we slightly modify the key generation process based on an idea from [Kra00]. Before we describe the modification and state our second Theorem, we provide the used definitions.

4.1 Preliminaries II

We stick to the notions and definitions from the last sections. In the following we define stateful pseudorandom generators and the notion of forward security for these generators, but before we 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 is allowed to break in. This means that during the `cma` phase, the adversary is allowed to indicate to the experiment that she 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 $\mathsf{Exp}^{\mathrm{FSSIG}}_{\mathrm{Kes}(1^n,T)}(\mathsf{A})$
    $i \leftarrow 0$, state $\leftarrow$ null, $(\mathsf{sk}_0, \mathsf{pk}) \leftarrow \mathsf{Kg}(1^n, T)$
    While $i < T$ And out $\neq$ breakin
        (out, state) $\leftarrow \mathsf{A}^{\mathsf{Sign}(\mathsf{sk}_i,\cdot,i)}(1^n, \mathtt{cma}, \mathsf{pk}, \mathtt{state})$
        i++, $\mathsf{sk}_i \leftarrow \mathsf{KUpd}(\mathsf{sk}_{i-1}, i)$
    $(M^\star, \sigma^\star, i^\star) \leftarrow \mathsf{A}(1^n, \mathtt{forge}, \mathtt{state}, \mathsf{sk}_{i-1})$
    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 A in the above experiment we write

$$\mathrm{Succ}^{\mathrm{FSSIG}}\left(\mathrm{Kes}(1^n,T);\mathsf{A}\right) = \Pr\left[\mathsf{Exp}^{\mathrm{FSSIG}}_{\mathrm{Kes}(1^n,T)}(\mathsf{A}) = 1\right]$$

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

Definition 6 (FSSIG). *Let $n, q \in \mathbb{N}$, $t = \mathrm{poly}(n)$, $\textsc{Kes}$ a key evolving signature scheme. Fix $T \in \mathbb{N}$. We call $\textsc{Kes}(1^n, T)$ FSSIG-secure, if $\mathrm{InSec}^{\mathrm{FSSIG}}(\textsc{Kes}(1^n,T); t, q)$, the maximum success probability of all possibly probabilistic adversaries A, running in time $\leq t$, making at most q queries to each instance of Sign in the above experiment, is negligible in n:*

$$\mathrm{InSec}^{\mathrm{FSSIG}}(\textsc{Kes}(1^n,T); t, q) \overset{def}{=} \max_{\mathsf{A}}\{\mathrm{Succ}^{\mathrm{FSSIG}}(\textsc{Kes}(1^n,T); \mathsf{A})\} = negl(n)\,.$$

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, she is unable to distinguish the former outputs from random bit strings. More formally, a stateful PRG is a function $G_n : \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^b$, for $n, b \in \mathbb{N}, b = poly(n)$, that on input of a state $\textsc{State}_i$ of length n outputs a new state $\textsc{State}_{i+1}$ and b output bits. Forward security for a stateful PRG that is used to produce no more than $\widetilde{n}$ outputs is defined using the two following experiments $\mathsf{Exp}_{G_n}^{fsprg-1}(\mathsf{A})$ and $\mathsf{Exp}_{G_n}^{fsprg-0}(\mathsf{A})$ which are based on the ones from [BY03]. In both experiments the adversary 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 at random. The adversary can keep a history using the variable h. The adversary can switch to the `guess` phase setting $d = \mathtt{guess}$. In the `guess` phase, the adversary gets the current state of G_n and has to output one bit, indicating if the bit strings were random or generated by G_n:

```
Experiment $\mathsf{Exp}_{G_n}^{fsprg-1}(\mathsf{A})$
  $\textsc{State}_0 \xleftarrow{\$} \{0,1\}^n$
  $i \leftarrow 0; h, d \leftarrow \mathtt{null}$
  Repeat
    $i \leftarrow i + 1$
    $(\textsc{Out}_i, \textsc{State}_i) \leftarrow G_n(\textsc{State}_{i-1})$
    $(d, h) \xleftarrow{\$} \mathsf{A}(1^n, \mathtt{find}, \textsc{Out}_i, h)$
  Until $(d = \mathtt{guess})$ Or $(i = \widetilde{n})$
  $g \xleftarrow{\$} \mathsf{A}(1^n, \mathtt{guess}, \textsc{State}_i, h)$
  Return $g$
```

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

The success probability of an adversary A is denoted by

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

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

Definition 7 (FSSIG). *Let $n, \widetilde{n} \in \mathbb{N}$, $t = \text{poly}(n)$, G_n a stateful PRG as defined above. We call G_n FSPRG-secure, if $\text{InSec}^{\text{FSPRG}}(G_n; t)$, the maximum success probability of all possibly probabilistic adversaries A, running in time $\leq t$, in the experiment above, is negligible in n:*

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

4.2 XMSS is Forward Secure

In the following we describe the modifications needed to make XMSS forward secure. Then we state our second 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 $\text{SEED} = \text{STATE}_0$ of length n, FsGen uses F_n and the previous state STATE_{i-1} to generate n bits of pseudorandom output OUT_i and a new state STATE_i of length n:

$$\mathsf{FsGen}(\text{STATE}_{i-1}) = (\text{STATE}_i || \text{OUT}_i) = (f_{\text{STATE}_{i-1}}(0) || f_{\text{STATE}_{i-1}}(1))$$

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 2, 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. This problem is already addressed in [BDS08]. We use their solution that requires to store $2H$ states of FsGen. This results in a secret signature key size of $2Hn$.

For the modified XMSS from above we proof the following security theorem.

Theorem 4. *If $\mathcal{H}_n$ is a second preimage resistant hash function family and 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 3. But this time it is more complicated to upper bound the success probability in the case of random bit strings.

Proof (of Theorem 4). First we revisit a result from [BY03] about the security of FsGen. There the authors show that if F_n is a pseudorandom function family with insecurity function $\text{InSec}^{\text{PRF}}(F_n; t, q)$, then FsGen is a forward secure PRG with insecurity function

$$\text{InSec}^{\text{FSPRG}}(\mathsf{FsGen}; t) = 2\widetilde{n} \cdot \text{InSec}^{\text{PRF}}(F_n; (t + 2\widetilde{n}), 2)\,.$$

The proof makes use of a hybrid argument and can be found in [BY03].

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

Claim 5. *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 $\text{InSec}^{\text{EU-CMA}}\left(XMSS'(1^{n'}, 2^H); t, q = 1\right)$. Further, let $XMSS^\star$ 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*

$$\mathrm{InSec}^{\mathrm{FSSIG}}\left(XMSS^\star(1^{n'},2^H);t,q=1\right) \leq 2^H \cdot \mathrm{InSec}^{\mathrm{FSPRG}}\left(\mathsf{FsGen};t'\right) \\ +\mathrm{InSec}^{\mathrm{EU\text{-}CMA}}\left(XMSS'(1^{n'},2^H);t,q=1\right)$$

$t' = t + \mathsf{t}_{\mathsf{Kg}^\star} + Tt_{\mathsf{KUpd}^\star} + q\mathsf{t}_{\mathsf{Sign}} + \mathsf{t}_{\mathsf{Vf}}$.

Proof (of claim). We want to limit the success probability of any adversary A that tries to break the forward security of XMSS*. More specifically, we want to find an upper bound for the insecurity function $\mathrm{InSec}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}^\star(1^{n'},2^H);t,q=1\right)$. Therefore we assume 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 M^{A} distinguishing the output of FsGen from truly random outputs, given black box access to A.

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

Now we calculate the success probability of M^{A} in distinguishing the output of FsGen from uniformly random outputs. The probability that 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 M^{A} is run in $\mathsf{Exp}^{fsprg-1}_{\mathsf{FsGen}}(M^{\mathsf{A}})$, the OUT_i, $1 \leq i \leq 2^H$ are pseudorandom outputs of FsGen. Hence A succeeds with probability $\mathrm{Succ}^{\mathrm{FSSIG}}\left(\mathrm{XMSS}^\star(1^{n'},2^H);\mathsf{A}\right)$ per definition. As M^{A} returns 1 if A is successful we get

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

If M^{A} is in $\mathsf{Exp}^{fsprg-0}_{\mathsf{FsGen}}(M^{\mathsf{A}})$, 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 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 M^{A} returns 1 if A succeeds. We will need an upper bound for the probability that M^{A} returns 1, so we have to limit A's success probability for the case that A breaks in in time period $i=\alpha-1$. We will show that in this case, A succeeds with probability $\leq \mathrm{InSec}^{\mathrm{EU\text{-}CMA}}\left(\mathrm{XMSS}'(1^{n'},2^H);t,q=1\right)$. For the moment assume this is true. Then we get

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

Putting all of this together, we get

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

and therefore

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

As this holds for all A running in time $\leq t$, making at most $q = 1$ queries to each instance of Sign we get

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

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

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

We do this, showing how to build an oracle machine $\hat{M}^{\mathsf{A}}$, that behaves exactly like M^{A}, from A's point of view. In contrast to M^{A}, $\hat{M}^{\mathsf{A}}$ uses 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 $\mathcal{H}_n$. We describe $\hat{M}^{\mathsf{A}}$.

$\hat{M}^{\mathsf{A}}$ 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 $\hat{M}^{\mathsf{A}}$ gets access to the corresponding signing oracle for pk_c. Like M^{A}, $\hat{M}^{\mathsf{A}}$ chooses $\alpha \xleftarrow{\$} \{1, \ldots, 2^{H}\}$ uniformly at random. Additionally, $\hat{M}^{\mathsf{A}}$ chooses $\beta \xleftarrow{\$} \{0, \alpha - 1\}$ uniformly at random. Next $\hat{M}^{\mathsf{A}}$ generates 2^{H} W-OTS* key pairs. This is done in a way simulating the $\mathsf{Exp}_{\mathsf{FsGen}}^{fsprg-0}(M^{\mathsf{A}})$ case: For the first α key pairs $\hat{M}^{\mathsf{A}}$ uses a random seed. Then $\hat{M}^{\mathsf{A}}$ uses FsGen to compute $\textsc{State}_{\alpha}$ using a random seed and uses FsGen starting from $\textsc{State}_{\alpha}$ to generate the seeds for the remaining key pairs. Afterwards $\hat{M}^{\mathsf{A}}$ 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 M^{A}.

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

If A returns a valid forgery $(M', (j, \sigma', \textsc{Auth}'))$ for time period $j < \alpha - 1$, $\hat{M}^{\mathsf{A}}$ 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 $j = \beta$ $\hat{M}^{\mathsf{A}}$ returns (M, σ'), otherwise $\hat{M}^{\mathsf{A}}$ 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, $\hat{M}^{\mathsf{A}}$ found a collision. If one of the inputs is x_c, $\hat{M}^{\mathsf{A}}$ returns the second preimage. Otherwise $\hat{M}^{\mathsf{A}}$ aborts. $\hat{M}^{\mathsf{A}}$ runs in time $t' = t + 2^H \cdot t_{\mathsf{Sign}} + t_{\mathsf{Vf}} + t_{\mathsf{Kg}}$.

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

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

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 1 we see that the right part of the equation for $\alpha = 2^H$ is exactly $\mathrm{InSec}^{\text{EU-CMA}}\left(\text{XMSS}'(1^{n'}, 2^H); t, q = 1\right)$. This concludes the claim. □

Combining this with the above result for FsGen leads 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* , $\mathrm{InSec}^{\text{FSSIG}}\left(\text{XMSS}^\star; t, q = 1\right)$, is bounded by

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

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

Note that, following the calculation in Section 3 the security level of the forward secure XMSS is

$$b \geq \min\left\{n - 2H - 2, n - H - 2 - w - 2\log(\ell w)\right\} - 1$$

as long as $t'' 2^{w + 2\log \ell w} < 2^{n-H-4} - 2^{\ell - 1} - 2^{-H}$.

5 Efficiency

In this Section we discuss the efficiency of XMSS. We will show that XMSS and its forward secure variant are efficient if $\mathcal{H}_n$ is an efficient second preimage resistant hash function family and F_n an efficient pseudorandom function family. Efficient here refers to the runtimes and space requirements for sufficiently secure parameters. We show this, showing that for fixed height, message length, and Winternitz parameter, the runtimes of the algorithms of XMSS are within a small constant factor of the runtimes needed to evaluate elements of the function families. Similarly, we show for the space requirements, that for fixed height, message length, and Winternitz parameter, they are within a small constant factor of the output length of the function families. As the forward secure variant of XMSS is slightly less performant and requires more space, we do the analysis for the forward secure variant. In Section 6 we will propose parameters that are secure according to [LV01] and present experimental results that support the efficiency of XMSS.

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. Therefore this is an important choice. We propose the BDS algorithm [BDS08] as it allows for optimal balanced runtimes using very little memory. The BDS algorithm allows for a time-memory trade-off controlled by the variable K. For the BDS algorithm it has been shown in [BDS09] that for $H, K \geq 2$, $H - K$ is even, it requires $\left(5H + \lfloor \frac{H}{2} \rfloor - 5K - 2 + 2^K\right) \cdot n$ bits for the state, including the secret key. Further it requires at most $(H - K)/2 + 1$ 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 a functions from F_n.

The runtime of all three algorithms of XMSS is dominated by the number $\#call_F$ of calls to F_n and the number $\#call_{\mathcal{H}}$ of calls to $\mathcal{H}_n$. We ignore the negligible computational overhead for adding the bitmasks, control flow and computing the base w representation of the message. Using a simple counting argument we obtain the following result:

For one call to the XMSS signature algorithm, the number of calls to $\mathcal{H}_n$ and F_n is bounded by

$$\#call_{\mathcal{H}} \leq \frac{\ell + 3}{2} \cdot (H - K) + \ell - \frac{1}{2}, \quad \#call_F \leq \frac{\ell w + 4}{2} \cdot (H - K) + \ell w + 2.$$

For one call to the XMSS signature verification algorithm, the number of calls to $\mathcal{H}_n$ and F_n is bounded by

$$\#call_{\mathcal{H}} \leq H + \ell, \quad \#call_F \leq \ell w.$$

For one call to the XMSS key generation algorithm, the number of calls to $\mathcal{H}_n$ and F_n is bounded by

$$\#call_{\mathcal{H}} \leq 2^H(\ell + 1), \quad \#call_F \leq 2^H(2 + \ell(w + 1)).$$

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 $\left(5H + \lfloor \frac{H}{2} \rfloor - 5K - 2 + 2^K + 1\right) \cdot n + 32$ bits, assuming the index is stored using a 32bit integer variable. $2(H - K)n + n$ of these bits have to be kept secret. For the remaining bits there is no secrecy requirement.

6 Implementation

We implemented XMSS to evaluate its practical performance. The implementation was done in C, using the AES and SHA2 implementation of OpenSSL[1]. The implementation is straightforward, except for the construction of $\mathcal{H}_n$ and F_n. We implemented constructions based on hash functions and block ciphers for both.

First we discuss the hash function based constructions. Our implementation supports the use of any hash function from the OpenSSL library that uses the Merkle-Darmgard (M-D) construction [Mer90b][2]. The family 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 F_n as

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

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

We show that this is a pseudorandom function family if `Hash` is a good cryptographic 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. Further 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.

For $\mathcal{H}_n$ we use `Hash` without modifications, as we only need a randomly chosen element of $\mathcal{H}_n$ and not the whole family. Here, we follow the standard assumption for the security of keyless hash functions. It assumes that a keyless hash function is an element of a family of hash functions, chosen uniformly at random.

Next we present the 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 smartcard crypto co-processors and also most of todays Intel processors provide hardware acceleration for AES. For F_n we use E without modification as a standard assumption states that a good block cipher can be modeled as pseudorandom permutation. $\mathcal{H}_n$ is constructed as $h_K(M) = C_2$ for $M = M_1||M_2$, with

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

in M-D mode. In [BRS02] the authors give a black box proof for the security of this construction. We do not use M-D strengthening, as our domain has fixed size. Table 1 shows our experimental results for XMSS on a computer with an Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz, 8GB RAM, and Infineon AES-NI[3]. The displayed results are for the forward secure construction. The construction with standard 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

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

[2] To obtain a secure instantiation, one must only use hash functions that are still assumed to be secure.

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

Table 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	220
SHA2	10	4	16	1,522	6.38	0.75	1,604	1,124	2,468	206
SHA2	10	4	64	3,925	16.67	1.97	1,604	1,060	1,764	156
SHA2	10	4	108	5,839	24.85	2.94	1,604	1,060	1,604	110
SHA2	16	4	4	54,180	5.96	0.44	2,660	1,572	4,772	214
SHA2	16	4	16	95,876	10.70	0.75	2,660	1,508	2,660	200
SHA2	16	4	64	247,494	27.83	1.95	2,660	1,444	1,956	150
SHA2	16	4	108	369,741	41.58	2.91	2,660	1,444	1,796	104
SHA2	20	8	4	879,010	6.09	0.45	10,404	1,828	4,900	210
SHA2	20	8	16	1,531,497	10.90	0.76	10,404	1,764	2,788	196
SHA2	20	8	64	3,991,598	28.54	1.98	10,404	1,700	2,084	146
SHA2	20	8	108	5,982,298	42.43	2.93	10,404	1,700	1,924	100
SHA2	20	4	4	868,647	7.62	0.44	3,364	1,828	4,900	210
SHA2	20	4	16	1,534,748	13.71	0.76	3,364	1,764	2,788	196
SHA2	20	4	64	4,012,157	35.60	1.97	3,364	1,700	2,084	146
SHA2	20	4	108	5,941,291	53.15	2.93	3,364	1,700	1,924	100
AES-NI	10	4	4	55	0.24	0.07	804	596	2,292	92
AES-NI	10	4	16	77	0.33	0.06	804	564	1,236	78
AES-NI	16	4	4	3505	0.41	0.07	1,332	788	2,388	86
AES-NI	16	4	16	4915	0.56	0.06	1,332	756	1,332	72
AES-NI	20	8	4	56526	0.42	0.07	5,204	916	2,452	82
AES-NI	20	8	16	78728	0.57	0.06	5,204	884	1,396	68
AES-NI	20	4	4	56066	0.52	0.07	1,684	916	2,452	82
AES-NI	20	4	16	79196	0.71	0.06	1,684	884	1,396	68
AES	10	4	4	129	0.49	0.11	804	596	2,292	92
AES	10	4	16	168	0.72	0.11	804	564	1,236	78
AES	16	4	4	7500	0.84	0.11	1,332	788	2,388	86
AES	16	4	16	10832	1.21	0.11	1,332	756	1,332	72
AES	20	8	4	120433	0.85	0.11	5,204	916	2,452	82
AES	20	8	16	171674	1.22	0.11	5,204	884	1,396	68
AES	20	4	4	119736	1.06	0.11	1,684	916	2,452	82
AES	20	4	16	172851	1.53	0.11	1,684	884	1,396	68
RSA 2048				-	3.08	0.09	≤ 512	≤ 512	≤ 256	87
DSA 2048				-	0.89	1.06	≤ 512	≤ 512	≤ 256	87
MSS-SPR (n=128, H=20)							2^{32}	960	8,512	98

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] the configurations with bit security 82 are secure until 2015. The configurations with bit security 100 and more are at least secure until 2039. Please note that these numbers are based on the provable security and not on the runtimes of possible attacks, which is the common practice. This would result in better values. For this reason we included also settings where the bit security is smaller than 80 bits. According to [LV01], RSA as well as DSA are assumed to be secure until 2022, using a 2048-bit key. The timings for RSA and DSA where 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. As mentioned in Section 2 it is possible to mitigate this using the tree chaining technique. 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 most 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. Using a larger K, the runtimes of signature generation and verification decrease, while the secret key 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 ($n = 256$). On the one hand, AES is obviously much faster than SHA2. On the other hand, keys and signatures using SHA2 are about twice the size of those for AES. The only drawback of using AES is the significant decrease in the bit security. 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 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. 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. So 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 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.

References

[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.

[And97] Ross Anderson. Two remarks on public key cryptology. In *Manuscript. Relevant material presented by the author in an invited lecture at the 4th ACM Conference on Computer and Communications Security, CCS*, pages 1–4. Citeseer, 1997.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[DSS05] Chris Dods, Nigel Smart, and Martijn Stam. Hash based digital signature schemes. In *Cryptography and Coding*, pages 96–115. Springer Verlag LNCS 3796, November 2005.

[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/.

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

[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.

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

[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.

[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.

[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.

[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.

[KCL06] Bo Kim, Kyu Choi, and Dong Lee. Disaster coverable pki model utilizing the existing pki structure. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4277 of *Lecture Notes in Computer Science*, pages 537–545. Springer Berlin / Heidelberg, 2006.

[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.

[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.

[Len04] Arjen K. Lenstra. Key lengths. Contribution to The Handbook of Information Security, 2004.

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

[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.

[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.

[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.

[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.

[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 K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[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.

[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.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554. Springer Berlin / Heidelberg, 2004.