Hash Based Digital Signature Schemes

Anders Fog Bunzel, *Aarhus University*

Abstract—One-time signature schemes based on one-way hash functions offer two advantages compared to digital signature schemes based on trapdoor one-way functions such as RSA and ElGamal; signing and verification are very efficient and they are quantum immune. In this paper we discuss four different one-time signature schemes of Merkle, Winternitz, Bleichenbacher and Mauer. Common to all four one-time signature schemes are that they can be represented as trees or graphs, and we can therefore analyze them according to efficiency, number of hash operation needed to generate the trees or graphs and finally size of keys and signatures. We also prove that the four one-time signature schemes are secure against a chosen message attack.

1 Introduction

Security of digital signature schemes used in practice today are often based on the difficulty of factoring large integers and computing discrete logarithms such as RSA [1] and ElGamal [2]. These schemes have two main drawbacks; they are not quantum immune and they doesn't fit into small devices with limited computing power.

One-time signature schemes based on one-way hash functions deals with these two problems. A one-time signature scheme is a public key signature scheme with the property that it can only sign one message per key pair. They were first presented by Lamport [3] and later improved by Merkle and Winternitz [4] where Winternitz's one-time signature scheme is a generalization of Merkle's one-time signature scheme. Later Bleichenbacher and Maurer [5], [6] presented a generalization of the one-time signature schemes based on directed acyclic graphs.

The main problem of one-time signature schemes is key management, i.e. they can only sign one message per key pair. Merkle [4] presented a solution to this problem with his Merkle tree which authenticate multiple keys, but compared to e.g. RSA, this solution is not sufficiently efficient.

The purpose of this paper is to analyze the four one-time signature schemes we define in Section 3; the FMT-seq signature scheme, the Winternitz signature scheme, the Bleichenbacher-Mauer-Tree signature scheme and the Bleichenbacher-Mauer-Graph signature scheme, according to efficiency as defined in [5], the number of hash operations used to generate the trees and the sizes of signatures and keys.

The outline of the paper is as follows. Section 2 provides the notations and definitions used in the rest of the paper, Section 3 describe the four one-time signature schemes, and in Section 4 we analyze them. In Section 5 we make a comparison of the four one-time signature schemes, and finally in the appendices are given a part of the proof of Theorem 2 and a full description of the four one-time signature schemes.

2 Notations and definitions

In this section we present some security notations and definitions used in the rest of the paper.

2.1 Pseudo-random number generators (PRNG)

Randomness is essential in many aspects of cryptography; from generation of keys to sampling randomness in various protocols. The algorithm used for generating randomness is called a pseudo-random number generator (PRNG).

A PRNG collects randomness from low-entropy input streams such as key stroke and mouse movement (the seed) that should be unpredictable from an adversary, and tries to generate outputs that are indistinguishable from truly random bit strings. A PRNG is secure if the advantage of the adversary $\mathcal{A}$ in Game 1 is negligible in the length of the bit string r.

Game 1 (PRNG-security). Let $\mathcal{A}$ be a probabilistic polynomial time adversary and O an oracle. O sends a bit string r to $\mathcal{A}$ which is either a truly random bit string or generated by a PRNG. $\mathcal{A}$ then outputs 1 if he think r is generated by the PRNG and otherwise he outputs 0. If $\mathcal{A}$ guess correctly he wins the game.

2.2 Hash functions

A hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ maps an arbitrary size input $x \in \{0,1\}^*$ to a fixed sized output $y \in \{0,1\}^k$, also called the fingerprint or message digest of the input, where the output size is defined by the security parameter k.

Let $X = \{0,1\}^*$ and $Y = \{0,1\}^k$, i.e. $\mathsf{H} : X \to Y$. We say that the hash function H is secure if the following three problems are hard to solve:

1) **Preimage:** Given a hash function H and an element $y \in Y$ it should be hard to compute its preimage $x \in X$ such that $y = \mathsf{H}(x)$. If preimage is hard to solve, H is said to be one-way or preimage resistant.
2) **Second preimage:** Given a hash function H and an element $x \in X$, it should be hard to find another element $x' \in X$ such that $\mathsf{H}(x) = \mathsf{H}(x')$. If second preimage is hard to solve, H is said to be second preimage resistant.
3) **Collision:** Given a hash function H, it should be hard to find two elements $x, x' \in X$ such that $\mathsf{H}(x) = \mathsf{H}(x')$. If collision is hard to solve, H is said to be collision resistant.

Clearly, if we can do a second preimage attack, we can also do a collision attack. Therefore, the best security is obtained if the hash function is collision resistant.

By the "birthday paradox" it's possible to find a collision in every $2^{k/2}$ evaluations of the hash function as described in [7]. Therefore, with current state of the art, $k = 160$ is preferable.

2.3 Digital signature schemes

A digital signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ is a triple with a key generation algorithm KGen, a signing algorithm Sig and a verification algorithm Vf.

The key generation algorithm KGen is probalistic and given the security parameter k as input, KGen returns the key pair $(\mathsf{sk}, \mathsf{vk})$ where sk is the secret signing key and vk is the public verification key (pk is sometimes used to denote the public verification key instead of vk).

The signing algorithm Sig is either deterministic or probabilistic and given the secret signing key sk and the message m, Sig returns the signature σ of m. The verification algorithm Vf is deterministic and given the message m and the signature σ, Vf returns either true or false depending on whether σ is a valid signature of m. It should always be true that $\mathsf{Vf}_{\mathsf{vk}}(\mathsf{Sig}_{\mathsf{sk}}(m), m) \rightarrow \mathsf{true}$ for $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(k)$ and the message m.

The best achievable security for a digital signature scheme Σ is against a chosen message attack (CMA), and is defined by Game 2, where Σ is secure if the probability that the adversary $\mathcal{A}$ wins, i.e. he is able to forge a signature, is negligible in the security parameter k.

Game 2 (CMA-security). Let $\mathcal{A}$ be a probabilistic polynomial time adversary and O an oracle. Both are given the security parameter k as input. First O runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(k)$, where vk is given to $\mathcal{A}$. Then $\mathcal{A}$ submit as many messages m as he wants, and for each message m he receive its signature $\sigma = \mathsf{Sig}_{\mathsf{sk}}(m)$ from O. At some point $\mathcal{A}$ outputs a message m_0 and a signature σ_0, where m_0 is not one of the messages O was asked to sign. If $\mathsf{Vf}_{\mathsf{vk}}(\sigma_0, m_0) \rightarrow \mathsf{true}$ then $\mathcal{A}$ wins the game and has forged the signature σ_0 of the message m_0.

2.3.1 One-time signature schemes (OTS)

A one-time signature (OTS) scheme is a digital signature scheme that only can be used to sign one message per key pair.

Two main advantages of an OTS scheme is that they are based on one-way hash functions and the signing and verification algorithms are very fast compared to public key digital signature scheme such as RSA [1] and ElGamal [2]. On the other hand, there are certain drawbacks of a OTS scheme; the limited number of signatures that can be signed (using a Merkle signature scheme more than one message can be signed using the same public verification key, see section 2.5), the length of signatures and the size of keys.

The public verification keys in an OTS scheme can be seen as a commitment to the secret signing keys, where it's often the case that $\mathsf{vk}_i = \mathsf{H}(\mathsf{sk}_i)$ for some $i > 0$ and a one-way hash function H. The signer gives the committed values vk_i to the verifier in an authenticated way and during verification he open the committed values by sending the secret signing keys sk_i to the verifier who checks that $\mathsf{vk}_i = \mathsf{H}(\mathsf{sk}_i)$.

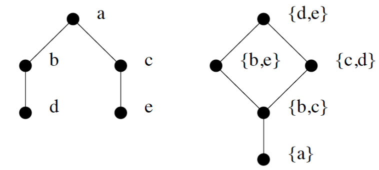

Fig. 1. The tree $T = [C_2C_2]$ and its associated poset $(T^*, \leq)$. The figure is copied from [6].

2.4 Tree-based one-time signature schemes

This section is a recap of the notations and definitions given in [6].

Let $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^k$ be a hash function with security parameter k and $T = (V, E)$ be a tree with vertex set V and edge set E, where the edges are directed from the leafs to the root. The tree T is a Merkle tree (hash tree), i.e. a vertex in the tree is the fingerprint of its children. We let the leafs in the tree be the secret signing keys sk and the root be the public verification key vk.

Let C_n denote the tree with a single path of n vertices, i.e. a chain of n vertices. For two trees T_1 and T_2 we denote $[T_1T_2]$ as the tree with a new root and T_1 and T_2 as subtrees. A subtree in this context is defined as a subtree whose leafs are also the leafs in the original tree. We also define a minimal cut set as the set of vertices which contains exactly one vertex from every path between the leafs and the root. We denote the set of minimal cut sets of T as T^*.

A poset (partially ordered set) $(T^*, \leq)$ is the set T^* with the order relation $\leq$ where $U \leq W$ for two minimal cut sets $U, W \in T^*$ if and only if every path from a vertex $w \in W$ to the root contains a vertex $u \in U$ (in words U is computable from W). We call $(T^*, \leq)$ the associated poset of the tree T and the maximal achievable size of an associated poset for a tree with n vertices is denoted $v(n)$. The associated poset of a tree can be computed recursively as defined in Theorem 4.1 in [6]:

Theorem 1. The associated poset of the chain C_n with n vertices is defined as

$$(C_n^*, \leq) \cong C_n \tag{1}$$

and for the tree $[T_1T_2]$ with the root x and the two subtrees T_1 and T_2 as

$$([T_1T_2]^*, \leq) \cong ((T_1^* \times T_2^*) \cup \{x\}, \leq_T) \tag{2}$$

where the order relation $\leq_T$ is defined by (i) $\{x\} \leq U$ for all $U \in (T_1^* \times T_2^*)$ and by (ii) $(U, W) \leq_T (U', W')$ if and only if both $U \leq U'$ in $(T_1^*, \leq)$ and $W \leq W'$ in $(T_2^*, \leq)$.

E.g. let $T = [C_2C_2]$ be the tree represented in Figure 1. To compute the associated poset we have that $T_1^* = \{b, d\}$, $T_2^* = \{c, e\}$ and $(T_1^* \times T_2^*) = \{\{b, c\}, \{b, e\}, \{c, d\}, \{d, e\}\}$. As illustrated in the same figure, then $\{a\} \leq U$ for all $U \in (T_1^* \times T_2^*)$ as defined by (i), and e.g. $\{b, c\} \leq_T \{b, e\}$ because $\{b\} \leq \{b\}$ in $(T_1^*, \leq)$ and $\{c\} \leq \{e\}$ in $(T_2^*, \leq)$ as defined by (ii).

Two minimal cut sets $U, W \in T^*$ are comparable if and only if $U \leq W$ or $W \leq U$ and they are incomparable otherwise. A subset $A \subseteq T^*$ is called an antichain if all

n	$\mu(n)$	$\nu(n)$	n	$\mu(n)$	$\nu(n)$	n	$\mu(n)$	$\nu(n)$	n	$\mu(n)$	$\nu(n)$
1	1	1	9	4	17	17	29	171	25	246	1718
2	1	2	10	5	22	18	39	222	26	326	2228
3	1	3	11	7	31	19	53	311	27	448	3132
4	1	4	12	8	41	20	67	411	28	576	4142
5	2	5	13	11	53	21	85	534	29	732	5372
6	2	7	14	14	71	22	114	711	30	977	7172
7	3	10	15	19	101	23	156	1011			
8	3	13	16	23	131	24	195	1314			

Fig. 2. The values of $v(n)$ and $\mu(n)$ for trees of size $n \leq 30$. The figure is copied from [6].

pair of minimal cut sets $(U, W) \in A$ are incomparable. A minimal cutset in the antichain A is also called a signature pattern. The maximal cardinality of an antichain is denoted $w((T^*, \leq))$, i.e. the size of the largest antichain in $(T^*, \leq)$. The maximal achievable size of an antichain in a tree with n vertices is denoted $\mu(n)$.

If there exists a collision resistant mapping G from the message space $\mathcal{M}$ to the antichain A, then we can use A as an OTS scheme with signatures as the signature patterns because A satisfy the following two requirements:

1) Signatures must be verifiable: The public verification key vk (the root of the tree) is computable from every signature pattern in A, i.e. every signatures are verifiable.
2) It should not be possible to forge signatures: Every signature pattern in A are incomparable, i.e. given a signature (pattern) of a message it's not possible to compute a signature (pattern) of a different message without inverting the used hash function.

Using the antichain A as the OTS scheme implies that we can maximal sign a $\log_2(w((T^*, \leq)))$-bit message. The one-time part is because if we sign multiple messages with the same tree we would reveal more and more vertices (from the signature patterns) and eventually we would have revealed all the secret signing keys (all the leafs of the tree).

As stated in [6] the value $v(n)$ can recursively be computed using Equation 3 where $v(n) = n$ for $n \leq 5$, but the value $\mu(n)$ can't.

$$v(n) = 1 + \max_{1 \leq i \leq n-2} \{v(i) \cdot v(n-i-1)\} \tag{3}$$

Fortunately we have the relation in Equation 4 to estimate the value of $\mu(n)$:

$$v(n) \geq \mu(n) \geq \frac{v(n)}{n} \tag{4}$$

See Figure 2 for values of $v(n)$ and $\mu(n)$ for trees of size $n \leq 30$.

2.4.1 An example

Let $T = [C_2C_2]$ be the tree illustrated in Figure 1 with its associated poset $(T^*, \leq)$ (also illustrated in Figure 1), where the secret signing keys are $\mathsf{sk} = (d, e)$ and the public verification key is $\mathsf{vk} = a$.

The antichain in T of maximal size is the set $A = \{\{b, e\}, \{c, d\}\}$ because the two signature patterns in A are incomparable. I.e. $\{b, e\} \not\leq \{c, d\}$ because the path from c to the root a doesn't contain the vertex e and $\{c, d\} \not\leq \{b, e\}$ because the path from b to the root a doesn't contain the vertex d.

Assume the message space is $\mathcal{M} = \{0, 1\}$ because $w((T^*, \leq)) = 2$. Now we could define the mapping $\mathsf{G} : \mathcal{M} \to A$ for this OTS scheme as $0 \mapsto \{b, e\}$ and $1 \mapsto \{c, d\}$, e.g. the signature of the message 1 is the signature pattern $\{c, d\}$.

Notice that the public verification key vk is computable from every signature pattern in A, i.e. the first requirement for A as defined previously is satisfied. The same is true for the second requirement, because given only the signature $\{c, d\}$ of the message 1, we have to invert the used hash function (i.e. invert $c = \mathsf{H}(e)$ to get e) to compute the signature $\{b, e\}$ of the message 0.

If we had used T to sign both 0 and 1 we would had revealed the two signature patterns $\{b, e\}$ and $\{c, d\}$ which contains the entire secret key sk.

2.5 Merkle signature schemes (MSS)

As described in section 2.3.1 it's only possible to sign one message per key pair with an OTS scheme which is inconvenient in most practical situation. One solution is to use a Merkle signature scheme (MSS) as described in [8], which is based on a Merkle tree and make it possible to use only one public verification key (the root of the Merkle tree) to verify multiple one-time signatures. Each leaf in the Merkle tree then corresponds to one OTS scheme, i.e. we can sign the same number of messages as leafs in the Merkle tree and verify them all with a single public verification key, the root (actually we use the public verification keys from the OTS scheme to verify the message, and then use the root to verify the public verification keys of the OTS scheme).

Assume H is a hash function and an OTS scheme such as Merkle's or Winternitz's OTS scheme has been chosen. The signer first selects $H \geq 2$ which is the height of the Merkle tree. Now the Merkle tree has 2^H leafs which is the number of messages that can be signed using this MSS. Therefore the signer generates the OTS key pairs sk^i_{OTS} and vk^i_{OTS} for $i = 1, 2, \ldots, 2^H$. Each leaf in the Merkle tree is then the fingerprint of all the public verification keys in vk^i_{OTS} concatenated together. The Merkle tree is then constructed, where each vertex is the fingerprint of its two children and the root is published as the signers public verification key vk_{MSS}.

To signing a message m, the signer first compute the fingerprint $\mathsf{H}(m)$ of the message, then he sign the fingerprint using the i'te OTS scheme located at the i'te leaf which returns the signature σ_{OTS}. Next he computes the authentication path A^i, where the h'te vertex in A^i is the sibling to the h'te vertex in the path from the i'te leaf to the root and $h = 0, 1, \ldots, H$ is the vertex's height in the Merkle tree. The MSS signature of the message m is then $\sigma_{MSS} = (i, \sigma_{OTS}, \mathsf{vk}^i_{OTS}, A^i)$.

Verification of the MSS signature σ_{MSS} consists of two steps; first is the OTS scheme with the public verification keys vk^i_{OTS} used to verify the signature σ_{OTS} of $\mathsf{H}(m)$. Then is the authentication path A^i used to verify the public verification keys vk^i_{OTS}, by computing the root of the Merkle tree and comparing it to the root previously received root vk_{MSS} from the signer.

3 EFFICIENT ONE-TIME SIGNATURE SCHEMES

In this section we describe four MSS; the FMTseq signature scheme, the Winternitz signature scheme, the Bleichenbacher-Mauer-Tree signature scheme and the Bleichenbacher-Mauer-Graph signature scheme, where the first three use OTS schemes that are based on trees (every vertices have in-degree two at most) and the last one use a OTS scheme based on a graph (vertices may have in-degree greater than two).

3.1 The FMTseq signature scheme

The FMTseq signature scheme described in [9] is using Merkle's OTS scheme, which is described in [4], and [10] to generate the authentication path A^i. For a full description of the FMTseq signature scheme see Appendix B.

The secret signing keys used in Merkle's OTS scheme are generated by a secure PRNG. The PRNG needs to be secure otherwise could an adversary break the PRNG and then compute the secret signing keys. And if the adversary somehow learn one secret signing key, a secure PRNG prevent him from learning the other secret signing keys, even though they are all computed from the same seed.

As stated in section 2.3 the best achievable security for a signature scheme is against a chosen message attack (CMA), which the FMTseq signature scheme is secure against:

Theorem 2. The FMTseq signature scheme is CMA-secure if the used PRNG R is secure, the used hash function H is collision resistant and Merkle's OTS scheme is CMA-secure.

Proof of Theorem 2: The prove given in [8], that Lamport-Diffie's OTS scheme is CMA-secure, can be used without loss of generality for Merkle's OTS scheme.

To prove that the FMTseq signature scheme is CMA-secure, when the secure PRNG R is used to generated the secret signing keys, we use a black-box reduction: Assume we have two adversaries $\mathcal{A}'$ and $\mathcal{A}$. $\mathcal{A}'$ is trying to break R, i.e. tell whether the secret signing keys are truly random bit strings or output from R, and $\mathcal{A}$ is trying to forge a signature of the FMTseq signature scheme, i.e. a chosen message attack (CMA). Our goal is to construct $\mathcal{A}'$ such that he succeed by letting him use $\mathcal{A}$ where we doesn't care about how the CMA is carried out, i.e. we treat $\mathcal{A}$ as a black box (hence the name black-box reduction). The construction of $\mathcal{A}'$ is illustrated in Figure 3 where the secret signing keys used in the FMTseq signature scheme are generated by the oracle. $\mathcal{A}$ outputs 1 if the CMA succeed, i.e. he has forge a signature of the FMTseq signature scheme, and otherwise he outputs 0. Likewise, $\mathcal{A}'$ outputs "random" if he think the secret signing keys are truly random bit strings, else if he think they were generated by R he outputs "PRNG".

In Appendix A we have proved that the FMTseq signature scheme is CMA-secure when the secret signing keys are truly random bit strings. Therefore, when $\mathcal{A}$ outputs 0, $\mathcal{A}'$ know it's because the secret signing keys are truly random bit strings and he outputs "random". Likewise, when $\mathcal{A}$ outputs 1, $\mathcal{A}'$ know it's because the secret signing keys are generated by R and he outputs "PRNG". The above implies that $\mathcal{A}'$ can break R which is a contradiction because we assumed it was secure.

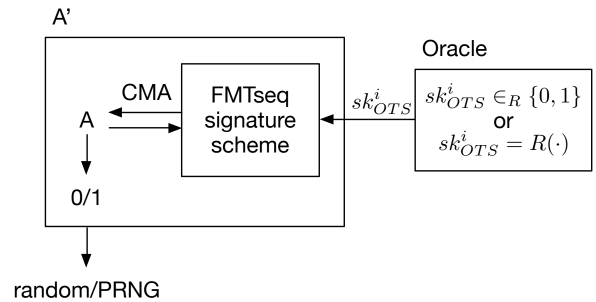

Fig. 3. The black-box reduction used in the proof of Theorem 2.

So, by using a black-box reduction we have proved that a secure PRNG implies CMA-security of the FMTseq signature scheme when using the same PRNG to generate the secret signing keys. □

3.2 The Winternitz signature scheme

The Winternitz signature scheme is using Winternitz's OTS scheme which is described in [8]. Winternitz's OTS scheme use the parameter $w \geq 2$ which defines the number of bits to be signed simultaneously where [11] states that Winternitz's OTS scheme is most efficient when $w = 2$ is chosen. In the rest of the paper we may write w in the equations but we use $w = 2$ in all computations. See Appendix C for a full description of the Winternitz signature scheme.

Because Winternitz's OTS scheme is a generalization of Merkle's OTS scheme (where Merkle's OTS scheme is using $w = 1$), the proof of CMA-security for the Winternitz signature scheme is almost identical to Theorem 2:

Theorem 3. The Winternitz signature scheme is CMA-secure if the used PRNG R is secure, the used hash function H is collision resistant and Winternitz's OTS scheme is CMA-secure.

Proof of Theorem 3: Winternitz's OTS scheme is a generalization of Merkle's OTS scheme, i.e. the proof of Theorem 2 can be used without loos of generality. □

3.3 The Bleichenbacher-Mauer-Tree signature scheme

The Bleichenbacher-Mauer-Tree signature scheme is using the trees described in section 2.4 as the OTS scheme.

Unlike the FMTseq and Winternitz signature scheme where the tree representation of the used OTS easily can be computed (i.e. we can easily compute the number of secret signing keys needed and the number of hash operations used to generate the tree), it's not the case with this signature scheme. The reason is because $w((T^*, \leq))$ for a tree T defines the number of bits that can be signed, and given a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ there doesn't exist a recursive method to compute a tree with $\log_2(w((T^*, \leq))) = k$ (remember we only sign the fingerprint of the message), i.e. we can't recursively generate a tree that can sign a k-bit message. Fortunately we can do better than an exhaustive search over all trees as stated in corollary 4.7 in [6]:

size	subtrees	size	subtrees
2	$\mathcal{C}_2$	7	$[\mathcal{C}_3\mathcal{C}_3]$
3	$\mathcal{C}_3$	8	$[\mathcal{C}_3\mathcal{C}_4]$, $[\mathcal{C}_2[\mathcal{C}_2\mathcal{C}_2]]$
4	$\mathcal{C}_4$	9	$[\mathcal{C}_4\mathcal{C}_4]$, $[\mathcal{C}_3[\mathcal{C}_2\mathcal{C}_2]]$
5	$[\mathcal{C}_2\mathcal{C}_2]$	10	$[\mathcal{C}_3[\mathcal{C}_2\mathcal{C}_3]]$, $[\mathcal{C}_2[\mathcal{C}_3\mathcal{C}_3]]$
6	$[\mathcal{C}_2\mathcal{C}_3]$	11	$[\mathcal{C}_3[\mathcal{C}_3\mathcal{C}_3]]$

Fig. 4. The table described in Corollary 1. The table is copied from [6].

TABLE 1
The estimated size of the trees described in section 2.4 for signing a k-bit message.

k	Lower bound	Upper bound
128	310	330
160	388	409
256	621	643
512	1242	1267

Corollary 1. Let T be any tree with n vertices. If every subtree of size $s \leq 11$ in T is contained in the table in Figure 4 then $w((T^*, \leq)) = \mu(n)$.

If we are not interested in the shape of the trees, it's also possible just to estimate the size of the tree for signing a k-bit message: $v(n)$ can recursively be computed using Equation 3, and using Equation 4 we have an upper and lower bound on $\mu(n)$. E.g. for $k = 160$ we have that $\log_2\left(\frac{v(409)}{409}\right) = 160$ and $\log_2(v(388)) = 160$, i.e. a tree for signing a 160-bit message has between 388 and 409 vertices. See Table 1 for estimated size of trees for signing various values of k.

So, how does we sign and verify a message with this OTS scheme? For signature generation the signer first compute the antichain A of maximal size and then use a collision resistant mapping G from the message space $\mathcal{M}$ to the antichain A. The signature is then the signature pattern mapped to by G. During verification the verifier first reconstruct the root of the tree with the revealed signature pattern and then compare it with the previously received root. If they are equal he know that the signature is valid. See Appendix D for a full description of the Bleichenbacher-Mauer-Tree signature scheme.

[12] proved that the OTS scheme using the trees described in section 2.4 is CMA-secure, but with a different notation; they have edges denoting the fingerprint and vertices denoting the hash functions where our trees are constructed in the opposite way. But fortunately as stated in [12], these two constructions are equivalent and we can therefore use their proof without loss of generality:

Theorem 4. The Bleichenbacher-Mauer-Tree signature scheme is CMA-secure if the used PRNG R is secure, the used hash function H is collision resistant and the OTS scheme using the tree described in section 2.4 is CMA-secure.

Proof of Theorem 4: The OTS scheme using the tree described in section 2.4 is CMA-secure as stated in [12]. It then follows from the proof of Theorem 2 with the tree described in section 2.4 as the used OTS scheme instead of Merkle's OTS scheme, that the Bleichenbacher-Mauer-Tree signature scheme is CMA-secure. □

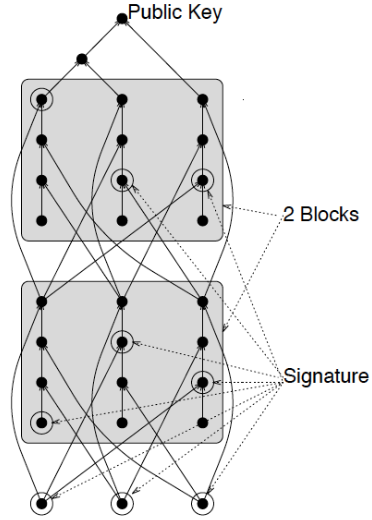

Fig. 5. The graph with $w = 3$, $B = 2$ blocks and a signature pattern of size 9 indicated. The figure is copied from [5].

3.4 The Bleichenbacher-Mauer-Graph signature scheme

The Bleichenbacher-Mauer-Graph signature scheme is using the graph described in section 6 in [5] as the OTS scheme.

Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$ be the used hash function, i.e. we want to sign a k-bit message. The graph then consist of $B = \lceil \frac{k}{\log_2(p)} \rceil + \lceil \log_p \left(\lceil \frac{k}{\log_2(p)} \rceil \right) \rceil$ blocks as defined in [11] where each block consists of $w \cdot (w+1)$ vertices. As [5] we use the value $w = 3$, which [11] also states is the most efficient value for the scheme. In the rest of the paper we may write w in the equations but we use $w = 3$ in all computations. The value p in the definition of B depends on w and $p = 51$ for $w = 3$ as given in [11].

The first row of vertices in each block together with the first w vertices are the secret signing keys, i.e. we have $w \cdot (B + 1)$ secret signing keys. The root of the graph is the public verification key.

As illustrated in Figure 5 with $B = 2$ blocks and $w = 3$, the blocks are connected in a specific way and the last row of vertices in the last block are hashed together to generate the public verification key.

For signature generation the signer first compute the antichain A of maximal size in the graph and then he use a collision resistant mapping G from the message space $\mathcal{M}$ to the antichain A (just as described in section 2.4 for trees). The signature is then equal the signature pattern mapped to by G. In Figure 5 is a signature pattern of size 9 indicated. Now verification is carried out in the obvious way, i.e. the verifier compute the root using the revealed signature pattern and compare it with the previous received root. If they are equal he know that the signature is valid. See Appendix E for a full description of the Bleichenbacher-Mauer-Graph signature scheme.

Theorem 5. The Bleichenbacher-Mauer-Graph signature scheme is CMA-secure if the used PRNG R is secure, the used hash function H is collision resistant and the OTS scheme using the graph described above is CMA-secure.

Proof of Theorem 5: The Bleichenbacher-Mauer-Graph signature scheme is a special case of the Bleichenbacher-Mauer-Tree signature scheme where the increased in-degree of some vertex has no influence on the security, i.e. the proof of Theorem 4 can be used without loos of generality. □

4 Analysis of the One-Time Signature Schemes

In this section we analyze the four OTS schemes used in the four MSS schemes described in section 3 according to $|(T^*, \leq)|$ and $w((T^*, \leq))$ as all four OTS schemes can be represented as trees (or three of the OTS schemes can be represented as trees and the last one, the OTS scheme in the Bleichenbacher-Mauer-Graph signature scheme, as a graph. But as described in [5] our notations for trees in section 2.4 can also be used for graphs). We only analyze the used OTS schemes because the Merkle tree used in each MSS scheme are the same, i.e. we can ignore this part of the MSS scheme.

We also analyze the number of hash operations used, the signature and key size and the efficiency of each OTS scheme. Finally we describe the Java implementation we used to get an idea about the shape of the trees described in section 2.4 and to estimate the number of leafs in the trees, which we need later in the computation of the number of hash operations used to generate the trees and the size of signatures and keys.

We use the following efficiency measure as defined in [5]:

$$\eta(\Sigma) = \frac{k}{n+1} \tag{5}$$

where k is the size of the message we want to sign and n is the number of vertices in the tree representation of the OTS scheme Σ. In [5] is an upper bound on the efficiency for a tree T given as:

$$\eta(T) \leq \gamma_T \approx 0.41614263726 \tag{6}$$

where γ_T is called the tree efficiency constant and they conjecture for a graph G that the graph efficiency constant γ_G is:

$$\eta(G) \leq \gamma_G = \frac{1}{2} \tag{7}$$

4.1 The Java implementation

We implemented in Java an algorithm for computing the associated poset $(T^*, \leq)$ (using the recursive method described in Theorem 1) and the largest antichain A of the tree T. Before we describe the method used to compute the largest antichain we first observe that the antichain is equal the maximum independent set of the graph representing the associated poset, because two vertices in the associated poset graph are only connected if one is computable from the other. We also remember that the complement of a minimum vertex cover is equal a maximum independent set and both problems are NP-complete. We are interested in the minimum vertex cover of the graph representing the associated poset because the Java library we used only had a greedy method for computing the minimum vertex cover.

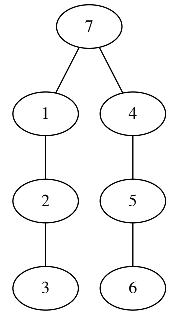

Fig. 6. The tree $T = [C_3C_3]$.

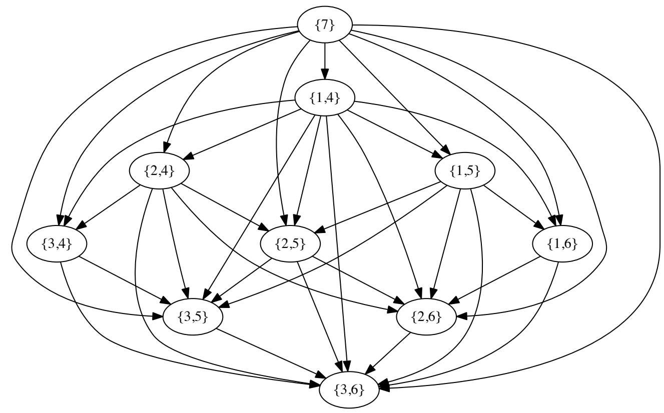

Fig. 7. The associated poset $(T^*, \leq)$ of the tree $T = [C_3C_3]$ illustrated in Figure 6. An arrow from e.g. the vertex $\{7\}$ to the vertex $\{1,4\}$ means that $\{7\}$ is computable from $\{1,4\}$ (i.e. $\{7\} \leq \{1,4\}$). Our implementation returned the antichain $A = \{\{3,5\},\{2,6\}\}$ of size 2, but the largest antichain is $A' = \{\{3,4\},\{2,5\},\{1,6\}\}$ of size 3 (i.e. $w((T^*, \leq)) = 3$).

Because we used a greedy algorithm to compute the vertex cover, we have found examples where our implementation doesn't return the largest antichain in the poset. E.g. the poset in Figure 7 we generated for the tree $T = [C_3C_3]$ in Figure 6 contains an antichain of size 3 (i.e. $w((T^*, \leq)) = 3$), but the greedy algorithm returned an antichain of size 2. Therefore, we believe that when our implementation find a tree with n vertices and $|(T^*, \leq)| = v(n)$, the tree also has $w((T^*, \leq)) = \mu(n)$.

In Table 2 is the result of $|(T^*, \leq)|$ and $w((T^*, \leq))$ using our implementation for trees of size $n \leq 27$. We where unable to compute the values for trees of size $n > 27$ because our implementation had a very high time complexity.

4.2 The FMTseq signature scheme

Let T_{FMTseq} be the tree representing the FMTseq signature scheme with height H. T_{FMTseq} has $t = 2^H$ leafs which is also the number of k-bit messages that we can sign using T_{FMTseq}. Now each leafs in T_{FMTseq} are the root of the tree T_{Mer} representing Merkle's OTS scheme with $l = k + \lceil \log_2(k) \rceil$ leafs.

TABLE 2
The result of $|(T^*, \leq)|$ and $w((T^*, \leq))$ using our Java implementation where T is the shape of the tree, n is the number of vertices and l is the number of leafs in T. Green entries indicate that $|(T^*, \leq)| = v(n)$ and $w((T^*, \leq)) = \mu(n)$ according to Figure 2.

T	n	l	$\|(T^*, \leq)\|$	$w((T^*, \leq))$
C_1	1	1	1	1
C_2	2	1	2	1
C_3	3	1	3	1
C_4	4	1	4	1
$[C_2C_2]$	5	2	5	2
$[C_2C_3]$	6	2	7	2
$[C_3C_3]$	7	2	10	2
$[C_3C_4]$	8	2	13	3
$[C_4C_4]$	9	2	17	3
$[C_3[C_2C_3]]$	10	3	22	5
$[C_3[C_3C_3]]$	11	3	31	7
$[C_4[C_3C_3]]$	12	3	41	8
$[C_4[C_4C_3]]$	13	3	53	9
$[[C_3C_3][C_2C_3]]$	14	4	71	12
$[[C_3C_3][C_3C_3]]$	15	4	101	19
$[[C_3C_3][C_4C_3]]$	16	4	131	21
$[[C_3C_3][C_4C_4]]$	17	4	171	24
$[[C_3C_4][C_4C_4]]$	18	4	222	33
$[[C_3C_3][[C_3C_3]C_3]$	19	5	311	51
$[[C_3C_3][[C_3C_3]C_4]$	20	5	411	61
$[[C_3C_3][[C_3C_4]C_4]$	21	5	531	75
$[[C_3C_3][[C_3C_3][C_2C_3]]]$	22	6	711	91
$[[C_3C_3][[C_3C_3][C_3C_3]]]$	23	6	1011	141
$[[C_3C_3][[C_3C_4][C_3C_3]]]$	24	6	1311	163
$[[C_3C_3][[C_4C_4][C_3C_3]]]$	25	6	1711	222
$[[C_3C_3][[C_4C_4][C_3C_4]]]$	26	6	2221	251
$[[C_3[[C_3C_3][[C_3C_3][C_3C_3]]]]]$	27	7	3034	396

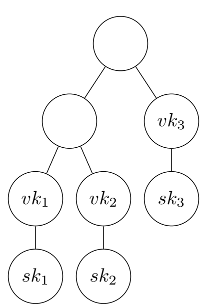

Fig. 8. The tree $T_{Mer} = [[C_2C_2]C_2]$ representing Merkle's OTS scheme for signing a $k = 2$-bit message where $|(T^*_{Mer}, \leq)| = 11$ and $w((T^*_{Mer}, \leq)) = 3$.

Each T_{Mer} consist of $n_{Mer} = 3 \cdot l - 1$ vertices when all public verification keys used in the OTS are hashed into a single public verification key (the root). Figure 8 illustrate the tree for $k = 2$. The size of T_{FMTseq} is then:

$$\begin{aligned} n_{FMTseq} &= t - 1 + t \cdot n_{Mer} \\ &= t - 1 + t \cdot (3 \cdot l - 1) \\ &= 3 \cdot l \cdot t - 1 \end{aligned} \tag{8}$$

TABLE 3
The values of $|(T^*_{Mer}, \leq)|$ and $w((T^*_{Mer}, \leq))$ using our Java implementation where k is the size of the signed message, n is the number of vertices and l is the number of leafs in T_{Mer}. Numbers inside a parenthesis are respectively $v(n)$ and $\mu(n)$ according to Figure 2.

k	n	l	$\|(T^*_{Mer}, \leq)\|$	$w((T^*_{Mer}, \leq))$
1	2	1	2 (2)	1 (1)
2	8	3	11 (13)	3 (3)
3	14	5	47 (71)	10 (14)
4	17	6	95 (171)	20 (29)
5	23	8	383 (1011)	70 (156)
6	26	9	767 (2228)	111 (326)
7	29	10	1535 (5372)	252 (732)

TABLE 4
Efficiency measure of Merkle's OTS scheme.

k	$\eta(T_{Mer})$
128	0.316049383
160	0.317460317
256	0.323232323
512	0.327575176

4.2.1 *Analysis of* $|(T^*_{Mer}, \leq)|$, $w((T^*_{Mer}, \leq))$ *and* $\eta(T_{Mer})$

First we analyze the values of $|(T^*_{Mer}, \leq)|$ and $w((T^*_{Mer}, \leq))$ for the tree T_{Mer}. Because [6] only had computed the values of $|(T^*, \leq)|$ and $w((T^*, \leq))$ in their appendix for trees of size $n \leq 30$ and we where unable to compute the values for trees of size $n > 27$, we are limited to trees of size $n \leq 30$ as given in Table 3. It's clear from the table that Merkle's OTS scheme is far away from being optimal according to $v(n)$ and $\mu(n)$ as defined in Figure 2. What the table also show, is that e.g. the tree representing Merkle's OTS scheme for signing a 5-bit message has the potential to sign a $\log_2(w((T^*_{Mer}, \leq))) = \log_2(70) \approx$ 6-bit message.

Finally we compute the efficiency of T_{Mer} using Equation 5:

$$\begin{aligned} \eta(T_{Mer}) &= \frac{k}{n_{Mer} + 1} \\ &= \frac{k}{(3 \cdot l - 1) + 1} \\ &= \frac{k}{3 \cdot l} \end{aligned} \tag{9}$$

In Table 4 is the efficiency of Merkle's OTS scheme computed using Equation 9 with the definition of l and various values of k. If we compute the limit of $\eta(T_{Mer})$ as k goes to infinity using the definition of l, we get the upper bound on the efficiency for Merkle's OTS scheme:

$$\lim_{k \to \infty} \eta(T_{Mer}) = \lim_{k \to \infty} \frac{k}{3 \cdot l} = \frac{1}{3} \tag{10}$$

Again, Merkle's OTS scheme is far away from being efficient according to the tree efficiency constant given in Equation 6.

4.2.2 *Analysis of the number of hash operations used and the size of signatures and keys*

To generate the tree T_{Mer} with n vertices and l leafs we have to apply the hash function $|\mathsf{H}_{T_{Mer}}| = n - l$ times. In

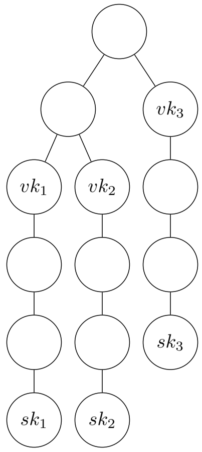

Fig. 9. The tree $T_{Win} = [[C_4C_4]C_4]$ representing Winternitz's OTS scheme with $w = 2$ for signing a $k = 2$-bit message where $|(T^*_{Win}, \leq)| = 69$ and $w((T^*_{Win}, \leq)) = 12$.

Merkle's OTS scheme we have l secret signing keys and l public verification keys where the public verification keys are hashed into a single public verification key, i.e. the keys are $|\mathsf{sk}| = k \cdot l$-bit and $|\mathsf{vk}| = k$-bit because each key is k-bit.

As illustrated in Figure 8 we have to compute $|\mathsf{H}_{\mathsf{Sig}}| = l$ and $|\mathsf{H}_{\mathsf{Vf}}| = l$ hash operations to sign and verify the message respectively. But because we represent Merkle's OTS scheme as a tree we also have to include the number of hash operations needed to generate the root, i.e. we set $|\mathsf{H}_{\mathsf{Vf}}| = l + (l - 1)$. Finally the signature consists of l vertices where each vertex correspond to k-bit, i.e. the signature size is $|\sigma| = k \cdot l$-bit.

The described values for Merkle's OTS scheme with various values of k are given in Table 5.

4.3 The Winternitz signature scheme

Let T_{WIN} be the tree representing the Winternitz signature scheme with height H and $t = 2^H$ leafs, which is also the number of k-bit messages that can be signed. Each leaf in T_{WIN} is the root of the tree T_{Win} representing Winternitz's OTS scheme. T_{Win} has $l = l_1 + l_2$ leafs where $l_1 = \lceil \frac{k}{w} \rceil$ and $l_2 = \lceil \frac{\lfloor \log_2(l_1) \rfloor + 1 + w}{w} \rceil$ as defined in [8]. As defined previously w is the number of bits to be signed simultaneously.

Each T_{Win} consists of $n_{Win} = l \cdot 2^w + l - 1$ vertices when all public verification keys are hashed into a single public verification key. Figure 9 illustrate the tree for $k = 2$. The size of T_{WIN} is then:

$$\begin{aligned} n_{WIN} &= t - 1 + t \cdot n_{Win} \\ &= t - 1 + t \cdot (l \cdot 2^w + l - 1) \end{aligned} \tag{11}$$

4.3.1 *Analysis of $|(T^*_{Win}, \leq)|$, $w((T^*_{Win}, \leq))$ and $\eta(T_{Win})$*

In Table 6 are the values of $|(T^*_{Win}, \leq)|$ and $w((T^*_{Win}, \leq))$ for trees of size $n \leq 30$ computed. It's clear that Winternitz's OTS is far away from being optimal according to $v(n)$ and $\mu(n)$ as defined in Figure 2. And just like Merkle's OTS scheme, the tree used to sign a 6-bit message has the potential to sign a $\log_2(w((T^*_{Win}, \leq))) = \log_2(155) \approx 7$-bit message.

TABLE 6
The values of $|(T^*_{Win}, \leq)|$ and $w((T^*_{Win}, \leq))$ using our Java implementation where k is the size of the signed message, n is the number of vertices and l is the number of leafs in T_{Win}. Numbers inside a parenthesis are respectively $v(n)$ and $\mu(n)$ according to Figure 2.

k	n	l	$\|(T^*_{Win}, \leq)\|$	$w((T^*_{Win}, \leq))$
1, 2	14	3	69 (71)	12 (14)
3, 4	19	4	277 (311)	39 (53)
5, 6	24	5	1109 (1314)	155 (195)

TABLE 7
Efficiency measure of Winternitz's OTS scheme with $w = 2$.

k	$\eta(T_{Win})$
128	0.371014493
160	0.376470588
256	0.384962406
512	0.390839695

Finally we use Equation 5 to compute the efficiency of T_{Win}:

$$\begin{aligned} \eta(T_{Win}) &= \frac{k}{n_{Win} + 1} \\ &= \frac{k}{(l \cdot 2^w + l - 1) + 1} \\ &= \frac{k}{l \cdot 2^w + l} \end{aligned} \tag{12}$$

In Table 7 is the efficiency of Winternitz's OTS scheme computed using Equation 12 with the definition of l and $w = 2$ for various values of k. The upper bound of $\eta(T_{Win})$ when using the definition of l and $w = 2$ is:

$$\lim_{k \to \infty} \eta(T_{Win}) = \lim_{k \to \infty} \frac{k}{l \cdot 2^w + l} = \frac{2}{5} \tag{13}$$

The upper bound of Winternitz's OTS scheme is a close approximation of the tree efficiency constant as defined in Equation 6, but the preferable sizes $k = \{128, 160, 256, 512\}$ in Table 7 are not.

4.3.2 *Analysis of the number of hash operations used and the size of signatures and keys*

To generate the tree T_{Win} with n vertices and l leafs we have to apply the hash function $|\mathsf{H}_{T_{Win}}| = n - l$ times. Winternitz's OTS scheme has l secret signing keys and l public verification keys where each key is k-bit and the public verification keys are hashed into a single public verification key, i.e. the size of the keys are $|\mathsf{sk}| = k \cdot l$-bit and $|\mathsf{vk}| = k$-bit. In worst case we need $|\mathsf{H}_{\mathsf{Sig}}| = (2^w - 1) \cdot l$ and $|\mathsf{H}_{\mathsf{Vf}}| = (2^w - 1) \cdot l$ hash operations to sign and verify the message respectively. But just like Merkle's OTS scheme we also add the number of hash operations needed to generate the root from the public verification keys, i.e. we set $|\mathsf{H}_{\mathsf{Vf}}| = (2^w - 1) \cdot l + (l - 1)$. Finally the signature consists

TABLE 5
The number of hash operations used to generate the tree T_{Mer} ($|\mathsf{H}_{T_{Mer}}| = n - l$), to sign ($|\mathsf{H}_{\mathsf{Sig}}| = l$) and verify ($|\mathsf{H}_{\mathsf{Vf}}| = l + (l-1)$) a k-bit message, the size of the secret signing keys ($|\mathsf{sk}| = k \cdot l$), the public verification keys ($|\mathsf{vk}| = k$) and the signature ($|\sigma| = k \cdot l$) in bits. n is the number of vertices and l is the number of leafs in T_{Mer}.

k	n	l	$\lvert\mathsf{H}_{T_{Mer}}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{H}_{\mathsf{Sig}}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{H}_{\mathsf{Vf}}\rvert$
128	404	135	239	17280	128	135	17280	269
160	503	168	335	26880	160	168	26880	335
256	791	264	527	67584	256	264	67584	527
512	1562	521	1041	266752	512	521	266752	1041

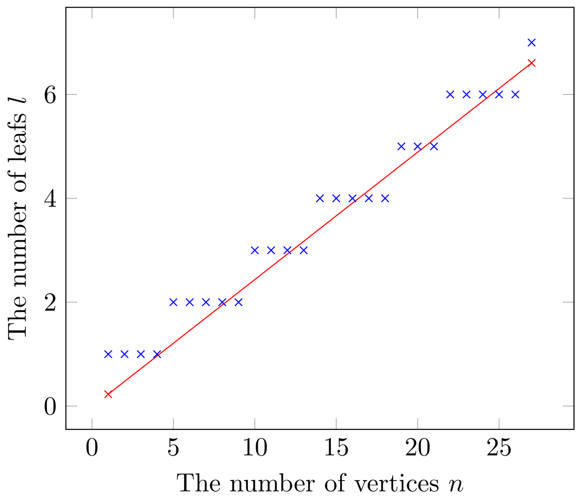

Fig. 10. The coordinate system used to estimate the number of leafs l in T_{BMtree} with n vertices using data from Table 2. The function of the trendline is given in Equation 14.

of l vertices where each vertex correspond to k-bit, i.e. the signature size is $|\sigma| = k \cdot l$-bit.

The described values for Winternitz's OTS scheme with various values of k are given in Table 8.

4.4 The Bleichenbacher-Mauer-Tree signature scheme

Let T_{BMTREE} be the tree representing the Bleichenbacher-Mauer-Tree signature scheme as described in section 3.3 with height H and $t = 2^H$ leafs, where each leaf is the root of the tree T_{BMtree} representing the OTS scheme described in section 2.4.

Later in the analyze we need the number of leafs in T_{BMtree} for $k = \{128, 160, 256, 512\}$, but because we where unable to generate these trees we try to estimate the number of leafs using data from Table 2. Plotting n and l from Table 2 in a coordinate system as illustrated in Figure 10 and computing the trendline (the red linear regression line in Figure 10) we get the following equation for computing the number of leafs l in T_{BMtree} with n vertices:

$$l = \frac{125}{546} \cdot n + \frac{149}{351} \tag{14}$$

It's important to remember that Equation 14 is only a roughly estimation because we have no ideas about the behavior of the graph in Figure 10 after $n = 27$! But it's the best we can do, and later we will see that the data computed using Equation 14 seems reliable.

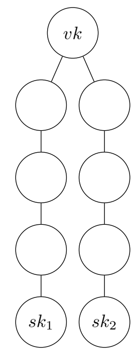

Fig. 11. The tree $T_{BMtree} = [C_4C_4]$ as described in section 2.4 for signing a $k = 2$-bit message where $|(T^*_{BMtree}, \leq)| = 17$ and $w((T^*_{BMtree}, \leq)) = 4$.

TABLE 9
Efficiency measure of the trees described in section 2.4 using the estimated values of n from Table 1.

k	n	$\eta(T_{BMtree})$
128	310	0.411575563
	330	0.386706949
160	388	0.411311054
	409	0.390243902
256	621	0.411575563
	643	0.397515528
512	1242	0.411906677
	1267	0.403785489

The size n_{BMtree} of T_{BMtree} for $k = \{128, 160, 256, 512\}$ is estimated in Table 1 as described previously. Using the result of our Java implementation in Table 2, the tree T_{BMtree} for $k = 2$ is illustrated in Figure 11. The size of T_{BMTREE} is then:

$$n_{BMTREE} = t - 1 + t \cdot n_{BMtree} \tag{15}$$

4.4.1 Analysis of $|(T^*_{BMtree}, \leq)|$, $w((T^*_{BMtree}, \leq))$ and $\eta(T_{BMtree})$

As described previous the optimal values of $|(T^*_{BMtree}, \leq)|$ and $w((T^*_{BMtree}, \leq))$ for trees of size $n \leq 30$ are given in Figure 2, and in Table 2 have we computed the shapes of the trees. The efficiency of T_{BMtree} is defined by:

$$\eta(T_{BMtree}) = \frac{k}{n_{BMtree} + 1} \tag{16}$$

In Table 9 is the efficiency of T_{BMtree} computed using Equation 16 with various values of k and the estimated val-

TABLE 8
The number of hash operations used to generate the tree T_{Win} ($|\mathsf{H}_{T_{Win}}| = n - l$), to sign ($|\mathsf{H}_{\mathsf{Sig}}| = (2^w - 1) \cdot l$) and verify ($|\mathsf{H}_{\mathsf{Vf}}| = (2^w - 1) \cdot l + (l - 1)$) a k-bit message in worst case, the size of the secret signing keys ($|\mathsf{sk}| = k \cdot l$), the public verification keys ($|\mathsf{vk}| = k$) and the signature ($|\sigma| = k \cdot l$) in bits. $w = 2$ defines the number of bits to be signed simultaneously, n is the number of vertices and l is the number of leafs in T_{Win}.

k	n	l	$\lvert\mathsf{H}_{T_{Win}}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{H}_{\mathsf{Sig}}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{H}_{\mathsf{Vf}}\rvert$
128	344	69	275	8832	128	207	8832	275
160	424	85	339	13600	160	255	13600	339
256	664	133	531	34048	256	399	34048	531
512	1309	262	1047	134144	512	786	134144	1047

ues of n from Table 1. The values are a close approximation of the tree efficiency constant as defined in Equation 6.

4.4.2 Analysis of the number of hash operations used and the size of signatures and keys

To generate the tree T_{BMtree} with n vertices and l leafs we have to apply the hash function $|\mathsf{H}_{T_{BMtree}}| = n - l$ times. The trees described in section 2.4 representing the OTS scheme have l secret signing keys and one public verification key where each key is k-bit, i.e. $|\mathsf{sk}| = k \cdot l$-bit and $|\mathsf{vk}| = k$-bit.

As described in section 2.4 the only hash operations needed during message signing are in the mapping G from the message space to the antichain, i.e. $|\mathsf{H}_{\mathsf{Sig}}| = |\mathsf{H}_{\mathsf{G}}|$. To verify the message in worst case we need the number of hash operation used to generate the tree minus one vertex, otherwise would all the secret signing keys be revealed to the verifier, i.e. $|\mathsf{H}_{\mathsf{Vf}}| = n - l - 1$. Finally, the signature size in worst case is the size of the largest signature pattern in the antichain where each vertex in the signature pattern is k-bit. The size of a signature pattern in the largest antichain of a tree is equal the number of leafs l, i.e. $|\sigma| = k \cdot l$-bit.

The described values for T_{BMtree} with various values of k and the estimated values of n are given in Table 10.

4.5 The Bleichenbacher-Mauer-Graph signature scheme

Let $T_{BMGRAPH}$ be the tree representation of the Bleichenbacher-Mauer-Graph signature scheme as described in section 3.4 with height H and $t = 2^H$ leafs, where each leaf is the root of the graph $G_{BMgraph}$ representing the OTS scheme described in same section. Each $G_{BMgraph}$ consist of $n_{BMgraph} = (w \cdot (w+1)) \cdot B + 5$ vertices where B is the number of blocks in the graph. The size of $T_{BMGRAPH}$ is then:

$$
\begin{aligned}
n_{BMGRAPH} &= t - 1 + t \cdot n_{BMgraph} \\
&= t - 1 + t \cdot ((w \cdot (w+1)) \cdot B + 5)
\end{aligned}
\tag{17}
$$

4.5.1 Analysis of $|(G^*_{BMgraph}, \leq)|$, $w((G^*_{BMgraph}, \leq))$ and $\eta(G_{BMgraph})$

In Table 11 are the values of $|(G^*_{BMgraph}, \leq)|$ and $w((G^*_{BMgraph}, \leq))$ given for k between 1 and 5 using $w = 3$. We where unfortunately unable to compute the values for $k > 5$. Notice that $w((G^*_{BMgraph}, \leq)) = 58$ compared to $\mu(17) = 29$ (as given in Figure 2) because $G_{BMgraph}$ is a graph and not a tree.

TABLE 11
The values of $|(G^*_{BMgraph}, \leq)|$ and $w((G^*_{BMgraph}, \leq))$ using our Java implementation with $w = 3$ where k is the size of the signed message, n is the number of vertices and B is the number of blocks in $G_{BMgraph}$.

k	n	B	$\lvert(G^*_{BMgraph}, \leq)\rvert$	$w((G^*_{BMgraph}, \leq))$
1-5	17	1	69	58

TABLE 12
Efficiency measure of the graph described in section 3.4 with $w = 3$.

k	$\eta(G_{BMgraph})$
128	0.435374150
160	0.437158470
256	0.449122807
512	0.456327986

Using Equation 5 we can compute the efficiency of $G_{BMgraph}$:

$$
\begin{aligned}
\eta(G_{BMgraph}) &= \frac{k}{n_{BMgraph} + 1} \\
&= \frac{k}{((w \cdot (w+1)) \cdot B + 5) + 1} \\
&= \frac{k}{(w \cdot (w+1)) \cdot B + 6}
\end{aligned}
\tag{18}
$$

In Table 12 is the efficiency computed with various values of k and $w = 3$. The upper bound of $\eta(G_{BMgraph})$ when using the definition of B and $w = 3$ is:

$$
\begin{aligned}
\lim_{k \to \infty} \eta(G_{BMgraph}) &= \lim_{k \to \infty} \frac{k}{(w \cdot (w+1)) \cdot B + 6} \\
&\approx 0.472702
\end{aligned}
\tag{19}
$$

The upper bound is almost a close approximation of the graph efficiency constant as defined in Equation 7, but the preferable sizes $k = \{128, 160, 256, 512\}$ are not.

4.5.2 Analysis of the number of hash operations used and the size of signatures and keys

To generate the graph $G_{BMgraph}$ with n vertices and B blocks we have to apply the hash function $|\mathsf{H}_{G_{BMgraph}}| = n - (B+1) \cdot w$ times where $(B+1) \cdot w$ is the number of "leafs" in the graph (the first row of vertices in each block

TABLE 10
The number of hash operations used to generate the tree T_{BMtree} ($|\mathsf{H}_{T_{BMtree}}| = n - l$), to sign ($|\mathsf{H}_{\mathsf{Sig}}| = |\mathsf{H}_{\mathsf{G}}|$) and verify ($|\mathsf{H}_{\mathsf{Vf}}| = n - l - 1$) a k-bit message in worst case, the size of the secret signing keys ($|\mathsf{sk}| = k \cdot l$), the public verification keys ($|\mathsf{vk}| = k$) and the signature ($|\sigma| = k \cdot l$) in bits. $|\mathsf{H}_{\mathsf{G}}|$ is the number of hash operation used in the mapping G from the message space to the antichain, n is the estimated number of vertices from Table 1 and l is the number of leafs in T_{BMtree} where l is estimated using Equation 14.

k	n	l	$\lvert\mathsf{H}_{T_{BMtree}}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{H}_{\mathsf{Sig}}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{H}_{\mathsf{Vf}}\rvert$
128	310	78	232	9984	128	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	9984	231
	330	83	247	10624	128	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	10624	246
160	388	97	291	15520	160	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	15520	290
	409	102	307	16320	160	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	16320	306
256	621	154	467	39424	256	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	39424	466
	643	160	483	40960	256	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	40960	482
512	1242	308	934	157696	512	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	157696	933
	1267	314	953	160768	512	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	160768	952

plus the w first vertices, see Figure 5 for an illustration with $w = 3$). These "leafs" are also the number of secret signing keys and the root of the graph is the public verification key, i.e. $|\mathsf{sk}| = k \cdot (B + 1) \cdot w$-bit and $|\mathsf{vk}| = k$-bit because each key is k-bit.

Like T_{BMtree}, the number of hash operations needed to sign a message is the number of hash operations used in the mapping G from the message space to the antichain, i.e. $|\mathsf{H}_{\mathsf{Sig}}| = |\mathsf{H}_{\mathsf{G}}|$. Likewise is the number of hash operation used to verify the message in worst case equal the number of hash operations needed to generate the graph, i.e. $|\mathsf{H}_{\mathsf{Vf}}| = B * w^2 + 2 \cdot (w - 1)$ where $B * w^2$ is the number needed to generate the B blocks and $2 \cdot (w - 1)$ is the number needed to generate the root. The size of a signature is likewise in worst case the size of the largest signature pattern in the antichain where each vertex in the signature pattern correspond to k-bit. Because $G_{BMgraph}$ is a graph the number of vertices in a signature pattern doesn't depend on the number of "leafs" as T_{BMtree} did. I.e. $|\sigma| = w(\mathsf{SP}) \cdot k$-bit where $w(\mathsf{SP})$ is the size of the largest signature pattern. E.g. for $G_{BMgraph}$ given in Table 11 we have that $w(\mathsf{SP}) = 10$.

The described values for $T_{BMgraph}$ with various values of k are given in Table 13.

5 Comparison of the One-Time Signature Schemes

In this section we compare the results of the four OTS schemes from the previous section. Where possible we only compare the result for $k = 160$-bit because it's the preferable security parameter for a hash function with current state of the art.

5.1 Comparison of $|(T^*, \leq)|$, $w((T^*, \leq))$ and $\eta(T)$

First we investigate the values of $|(T^*, \leq)|$ and $w((T^*, \leq))$ for Merkle's and Winternitz's OTS scheme and compare it with the OTS scheme using the trees described in section 2.4.

As previously described we were unable to generate trees of size $n > 27$ and [6] has only computed the values for trees of size $n \leq 30$. Therefore, in Table 14 we only have trees representing OTS schemes for signing a $k = 5$-bit message. As given in the table, the tree T_{Mer} representing Merkle's OTS scheme for signing a $k = 5$-bit message has

TABLE 14
Comparison of $|(T^*, \leq)|$ and $w((T^*, \leq))$ for trees with n vertices and l leafs for signing a $k = 5$-bit message.

OTS	n	l	$\lvert(T^*, \leq)\rvert$	$w((T^*, \leq))$
T_{Mer}	503	8	383	70
T_{Win}	424	5	1109	155
T_{BMtree}	18	4	222	39

TABLE 15
Comparison of the efficiency measure $\eta(T)$ of the trees with n vertices representing the OTS schemes for signing a $k = 160$-bit message.

OTS	n	$\eta(T)$
T_{Mer}	503	0.317460317
T_{Win}	424	0.376470588
$(T_{BMtree})_{n=\perp}$	388	0.411311054
$(T_{BMtree})_{n=\top}$	409	0.390243902
$G_{BMgraph}$	365	0.437158470

the potential to sign a $\log_2(w((T^*_{Mer}, \leq))) = \log_2(70) \approx 6$-bit message. Likewise has the tree T_{Win} representing Winternitz's OTS scheme the potential to sign a $\log_2(w((T^*_{Mer}, \leq))) = \log_2(155) \approx 7$-bit message. I.e. trees representing Winternitz's OTS scheme have the biggest waist of bits that could be signed. The tree T_{BMtree} is only given in the table for comparison with the optimal tree for signing a 5-bit message.

Finally we compare the efficiency measure of the trees representing the four OTS schemes in Table 15. Notice that for T_{BMtree} we have the tree representing the lower ($(T_{BMtree})_{n=\perp}$) and upper ($(T_{BMtree})_{n=\top}$) bound on the number of vertices as estimated in Table 1. We also remember from Equation 6 and Equation 7 that the tree and graph efficiency constant are $\gamma_T \approx 0.41614263726$ and $\gamma_G = \frac{1}{2}$ respectively. As given in the table all the trees are far away from the constant except from $(T_{BMtree})_{n=\perp}$ which is almost equal the constant. But because the number of vertices for this tree is the estimated lower bound we can't be sure that it's the correct tree for signing a $k = 160$-bit message.

TABLE 13
The number of hash operations used to generate the graph $G_{BMgraph}$ ($|\mathsf{H}_{G_{BMgraph}}| = n - (B+1) \cdot w$), to sign ($|\mathsf{H}_{\mathsf{Sig}}| = |\mathsf{H}_{\mathsf{G}}|$) and verify ($|\mathsf{H}_{\mathsf{Vf}}| = (B * w^2 + 2 \cdot (w-1))$) a k-bit message in worst case, the size of the secret signing keys ($|\mathsf{sk}| = k \cdot (B+1) \cdot w$), the public verification keys ($|\mathsf{vk}| = k$) and the signature ($|\sigma| = w(\mathsf{SP}) \cdot k$) in bits. $|\mathsf{H}_{\mathsf{G}}|$ is the number of hash operation used in the mapping G from the message space to the antichain, $w = 3$ defines the size of each block, n is the number of vertices, $l = (B+1) \cdot w$ is the number of "leafs", B is the number of blocks and $w(\mathsf{SP})$ is the size of the largest signature pattern in $G_{BMgraph}$.

k	n	l	$\lvert\mathsf{H}_{G_{BMgraph}}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{H}_{\mathsf{Sig}}\rvert$	$\lvert\sigma\rvert$	$\lvert\mathsf{H}_{\mathsf{Vf}}\rvert$
128	293	75	218	9600	128	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	$w(\mathsf{SP}) \cdot 128$	220
160	365	93	272	14880	160	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	$w(\mathsf{SP}) \cdot 160$	274
256	596	144	425	36864	256	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	$w(\mathsf{SP}) \cdot 256$	427
512	1121	282	839	144384	512	$\lvert\mathsf{H}_{\mathsf{G}}\rvert$	$w(\mathsf{SP}) \cdot 512$	841

5.1.1 Comparison of the number of hash operations used and the size of signatures and keys

In Table 16 we compare the trees representing the four OTS schemes according to the number of hash operations used and the size of signatures and keys. For T_{BMtree} we have three versions, two as described previously and the last one $(T_{BMtree})_{k=8}^{20*}$, as the tree $(T_{BMtree})_{k=8} = [[C_3C_3][[C_4C_4][C_3C_4]]]$ from Table 2 for signing a 8-bit message generated 20 times because $8 \cdot 20 = 160$-bit. I.e. the 160-bit message is divided into 8-bit blocks and each block is signed using one of the 20 trees.

As given in Table 16 $(T_{BMtree})_{k=8}^{20*}$ has way shorter signature and secret signing keys compared to the other but it come with the cost of the high number of vertices in the tree which implies that the number of hash operations needed to generate the tree and verify messages are high. We also observe in the table that T_{Mer} has almost twice the size of signature and secret signing keys compared to the others. That $(T_{BMtree})_{n=\perp}$ and $(T_{BMtree})_{n=\top}$ use fewest vertices of the trees in the table and thereby use fewest number of hash operations to generate the tree and to verify a message is anticipated because we have previously described that these type of trees are the most efficient one according to the number of bits per vertex that can be signed (as defined in Equation 5).

Finally we can conclude from the table that if the size of signatures and keys matters we should use a tree for signing few bits multiple times such as $(T_{BMtree})_{k=8}^{20*}$. Otherwise if we want a small tree, i.e. we want few hash operations to generate the tree and verify messages we should use $G_{BMgraph}$ but only if $|\mathsf{H}_{\mathsf{G}}|$ is cheap. Else we have to use T_{Win} which also have short signatures and keys because of its few leafs that results from the value w which defines the number of bits to be signed simultaneously. [11] also states that the Winternitz signature scheme is more efficient than the Bleichenbacher-Mauer-Graph signature scheme because $|\mathsf{H}_{\mathsf{G}}|$ isn't cheap.

6 Conclusion

In this paper we described and analyzed the four one-time signature schemes: the FMTseq signature scheme, the Winternitz signature scheme, the Bleichenbacher-Mauer-Tree signature scheme and the Bleichenbacher-Mauer-Graph signature scheme. Additionally we also proved that the four one-time signature schemes are CMA-secure.

The analysis shows that the trees and graphs proposed by Bleichenbacher and Mauer are the most efficient ones and with the fewest number of hash operations needed to verify messages and to generate the trees and graphs, but only if there exists an efficient mapping from the message space to the antichain. Otherwise is the Winternitz signature scheme the best candidate as other papers also consider as the best choice in practice.

Appendix A
Part of the proof of Theorem 2

Assume for the sake of contradiction that an adversary $\mathcal{F}$ with advantage $\epsilon_{\mathcal{F}}$ can forge a FMTseq signature. We can then use $\mathcal{F}$ to prove that the FMTseq signature scheme is CMA-secure when the secret signing keys are truly random bit strings, by proving that an adversary $\mathcal{A}$ using $\mathcal{F}$ can either forge a signature σ_{OTS} of Merkle's OTS scheme or find a collision for the hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$. But because we have assumed that Merkle's OTS scheme is CMA-secure and the hash function is collision resistant, we have a contradiction.

Section 8.3 in [8] was used as template for the following construction of the proof.

The algorithm:

1) Let O be an oracle that given a message returns a signature σ_{OTS} using Merkle's OTS scheme. O is given the security parameter k as input:
    a) Generates a key pair by running the probabilistic algorithm KGen: $(\mathsf{sk}_{Mer}, \mathsf{vk}_{Mer}) \leftarrow \mathsf{KGen}(k)$. O sends vk_{Mer} to $\mathcal{A}$.
2) The adversary $\mathcal{A}$ is given the height H of the Merkle tree and the security parameter k as input:
    a) Selects a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^k$.
    b) Selects an index $c \in_R \{1, 2, \ldots, 2^H\}$.
    c) Generates 2^H key pairs by running the probabilistic algorithm KGen: $(\mathsf{sk}_{OTS}^i, \mathsf{vk}_{OTS}^i) \leftarrow \mathsf{KGen}(k)$, where $i = 1, 2, \ldots, 2^H$ is the signature number and each key pair is used to sign and verify one message.
    d) Update the c'te public verification key to the one received from O: $\mathsf{vk}_{MSS}^c = \mathsf{vk}_{Mer}$.
    e) Sends the set of public keys vk_{OTS}^i, the height H and the hash function H to the forger $\mathcal{F}$.
3) The following exchange of messages between $\mathcal{A}$ and $\mathcal{F}$ occurs at most 2^H times:

TABLE 16
Comparison of the trees with n vertices and l leafs representing the OTS schemes for signing a $k = 160$-bit message. Green and red entries represent the lowest and highest value respectively in that column.

OTS	n	l	$\|\mathsf{H}_T\|$	$\|\mathsf{sk}\|$	$\|\mathsf{vk}\|$	$\|\mathsf{H}_{\mathsf{Sig}}\|$	$\|\sigma\|$	$\|\mathsf{H}_{\mathsf{Vf}}\|$
T_{Mer}	503	135	335	26880	160	168	26880	335
T_{Win}	424	85	339	13600	160	255	13600	339
$(T_{BMtree})_{n=\perp}$	388	97	291	15520	160	$\|\mathsf{H}_\mathsf{G}\|$	15520	290
$(T_{BMtree})_{n=\top}$	409	102	307	16320	160	$\|\mathsf{H}_\mathsf{G}\|$	16320	306
$(T_{BMtree})^{20*}_{k=8}$	520	120	400	960	160	$\|\mathsf{H}_\mathsf{G}\|$	960	380
$G_{BMgraph}$	365	93	272	14880	160	$\|\mathsf{H}_\mathsf{G}\|$	$w(\mathsf{SP}) \cdot 160$	274

a) $\mathcal{F}$ sends a message M and a signature number q to $\mathcal{A}$ where $1 \leq q \leq 2^H$.
b) If $q = c$ then $\mathcal{A}$ ask the oracle O for a signature of M. O computes the signature by $\sigma_{OTS} = \mathsf{Sig}_{\mathsf{sk}_{Mer}}(M)$ and sends it to $\mathcal{A}$. $\mathcal{A}$ then forward σ_{OTS} to $\mathcal{F}$.
c) Else if $q \neq c$ then $\mathcal{A}$ computes the signature of M using Merkle's OTS scheme by $\sigma_{OTS} = \mathsf{Sig}_{\mathsf{sk}^q_{MSS}}(m)$ and sends it to $\mathcal{F}$.
d) With $\epsilon_\mathcal{F}$ the forger $\mathcal{F}$ returns a forged signature of the FMTseq signature scheme, i.e. he returns to $\mathcal{A}$ a message $\overline{M} \neq M$ and a signature $\overline{\sigma_{MSS}} = (s, \overline{\sigma_{OTS}}, \overline{\mathsf{vk}^s_{MSS}}, \overline{A^s})$ where s is the signature number and $\mathsf{Vf}_{\mathsf{vk}_{MSS}}(\overline{M}, \overline{\sigma_{MSS}}) \rightarrow \mathsf{true}$ for the the root vk_{MSS} of the Merkle tree.

4) The adversary $\mathcal{A}$:

a) Compares the received signature $\overline{\sigma_{MSS}}$ of $\overline{M}$ with the signature $\sigma_{MSS} = (s, \sigma_{OTS}, \mathsf{vk}^s_{MSS}, A^s)$ of M.
b) If $(\overline{\mathsf{vk}^s_{MSS}}, \overline{A^s}) \neq (\mathsf{vk}^s_{MSS}, A^s)$ then $\mathcal{A}$ returns a collision for the hash function H.
c) Else if $(\overline{\mathsf{vk}^s_{MSS}}, \overline{A^s}) = (\mathsf{vk}^s_{MSS}, A^s)$ and $s = c$ then $\mathcal{A}$ returns a forged signature of Merkle's OTS scheme.

A.1 Explanation of 4.b (collision)

$\mathcal{A}$ construct the path $B^s = (B^s_0, B^s_1, \ldots, B^s_H)$ where $B^s_0 = \mathsf{H}(\mathsf{vk}^s_{MSS})$ (the s'te leaf), $B^s_H = \mathsf{vk}_{MSS}$ (the root) and $B^s_{i+1} = \mathsf{H}(B^s_i \| A^s_i)$ for $i = 0, 1, \ldots, H$. Likewise is the path $\overline{B^s}$ constructed by using $\overline{A^s}$.

Assume $B^s \neq \overline{B^s}$: This is true when $\mathsf{vk}^s_{MSS} \neq \overline{\mathsf{vk}^s_{MSS}}$. Because $B^s_H = \overline{B^s_H} = \mathsf{vk}_{MSS}$ there exists an index $0 \leq i < H$ such that $B^s_{i+1} = \overline{B^s_{i+1}}$ and $B^s_i \neq \overline{B^s_i}$ which is a collision.

Assume $B^s = \overline{B^s}$: Then $B^s_0 = \overline{B^s_0}$. If $\underline{\mathsf{vk}^s_{MSS}} \neq \overline{\mathsf{vk}^s_{MSS}}$ then a collision is found. Else if $\mathsf{vk}^s_{MSS} = \overline{\mathsf{vk}^s_{MSS}}$ then $A^s \neq \overline{A^s}$ and $A^s_i \neq \overline{A^s_i}$ for some index $0 \leq i < H$, which is a collision for $B^s_{i+1} = \mathsf{H}(B^s_i \| A^s_i) = \mathsf{H}(\overline{B^s_i} \| \overline{A^s_i}) = \overline{B^s_{i+1}}$.

A.2 Explanation of 4.c (forgery)

When $s = c$ then $\overline{\mathsf{vk}^s_{MSS}} = \mathsf{vk}^s_{MSS} = \mathsf{vk}^c_{MSS} = \mathsf{vk}_{Mer}$ and $\overline{\sigma_{OTS}}$ is a valid Merkle one-time signature for $\overline{M}$, i.e. $\mathsf{Vf}_{\mathsf{vk}_{Mer}}(\overline{M}, \overline{\sigma_{OTS}}) \rightarrow \mathsf{true}$.

A.3 $\mathcal{A}$'s advantage

Either $\mathcal{A}$ find a collision for H or he forge a signature for Merkle's OTS scheme. $\mathcal{A}$'s probability ϵ_{CR} for returning a collision is at least $\epsilon_\mathcal{F}$. Likewise is $\mathcal{A}$'s probability ϵ_{OTS} for returning a forged signature at least $\frac{1}{2^H} \cdot \epsilon_\mathcal{F}$ because it depends on whether $s = c$ or not where $Pr[s = c] = \frac{1}{2^H}$.

Appendix B
The FMTseq signature scheme

The FMTseq signature scheme is a MMS using Merkle's OTS scheme. The following is a recap of the FMTseq signature scheme from [9] with a slightly different notation:

Key generation:

1) Choose a secure PRNG R, a secret key $SK \in \{0,1\}^k$ (the seed of R) and a collision resistant hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^l$.
2) Construct a Merkle tree of height H:

a) Generate $t = l + \log_2(l)$ secret signing keys for each OTS scheme by using R, such that $\mathsf{sk}^i_j = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
b) Use the hash function H to compute the t public verification keys by $\mathsf{vk}^i_j = \mathsf{H}(\mathsf{sk}^i_j)$.
c) Denote the i'te set of secret signing keys and public verification keys as sk^i_{OTS} and vk^i_{OTS} respectively.
d) Commit to the i'te public verification key by computing $plc_i = \mathsf{H}(\mathsf{vk}^i_1 \| \mathsf{vk}^i_2 \| \ldots \| \mathsf{vk}^i_t)$. plc_i is then the i'te leaf in the Merkle tree.

3) Publish the root of the Merkle tree, which we denote vk_{MSS}, and set the signature number $i = 0$.

Signing: The input is a message M and the output is a signature σ_{MSS} of M.

1) Increment i.
2) Sign the fingerprint of the message with Merkle's OTS scheme:

a) Calculate the number of 0-bits in $\mathsf{H}(M)$ and denote it C. Let $d = \mathsf{H}(M) \| C$.
b) Regenerate the i'te secret signing keys sk^i_{OTS} with the PRNG R.
c) Compute the signature: $\sigma_{OTS} = \mathsf{Sig}_{\mathsf{sk}^i_{OTS}}(M) = \{\mathsf{sk}^i_j \in \mathsf{sk}^i_{OTS} | d_j = 1\} \cup \{\mathsf{vk}^i_j \in \mathsf{vk}^i_{OTS} | d_j = 0\}$.

3) Create the authentication path from the i'te leaf to the root using [10] and denote it A^i.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, \mathsf{vk}^i_{OTS}, A^i)$.

Verifying: The input is a message M and a signature σ_{MSS} and the output is a boolean decision true or false.

1) Verify the message with Merkle's OTS scheme:
   a) Calculate the number of 0-bits in $\mathsf{H}(M)$ and denote it C. Let $\overline{d} = \mathsf{H}(M)\|C$.
   b) Let $J = \{j|\overline{d_j} = 1\}$ be indices and denote the elements in σ_{OTS} as $\overline{s_j}$ for $j = 1, 2, \ldots, t$.
   c) Update the elements in σ_{OTS} corresponding to the secret signing keys by $\overline{s_j} = \mathsf{H}(\overline{s_j})$ for all $j \in J$.
   d) Verify the signature by checking that $\overline{s_j} = \mathsf{vk}^i_j$ for $j = 1, 2, \ldots, t$ where $\mathsf{vk}^i_j \in \mathsf{vk}^i_{OTS}$.
2) Compute $\overline{plc} = \mathsf{H}(\overline{s_1}\|\overline{s_2}\| \ldots \|\overline{s_t})$.
3) Compute the root of the Merkle tree $\overline{\mathsf{vk}_{MSS}}$ using $\overline{plc}$ and the received authentication path A^i.
4) If $\overline{\mathsf{vk}_{MSS}} = \mathsf{vk}_{MSS}$ then the signature is valid and we output true, otherwise it's invalid and we output false.

Appendix C
The Winternitz signature scheme

The Winternitz signature scheme is a MMS using Winternitz's OTS scheme. The following is a recap of the Winternitz signature scheme where we use the definition of Winternitz's OTS scheme from [8].

Key generation:

1) Choose a secure PRNG R, a secret key $SK \in \{0,1\}^k$ (the seed of R) and a collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^l$.
2) Construct a Merkle tree of height H:
   a) Define $t = t_1 + t_2$ where $t_1 = \lceil \frac{l}{w} \rceil$ and $t_2 = \lceil \frac{\lfloor \log_2(t_1) \rfloor + 1 + w}{w} \rceil$. The value w defines the number of bits to be signed simultaneously.
   b) Generate the t secret signing keys for each OTS scheme by using R, such that $\mathsf{sk}^i_j = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
   c) Use the hash function H to compute the t public verification keys by applying it $2^w - 1$ times: $\mathsf{vk}^i_j = \mathsf{H}^{2^w-1}(\mathsf{sk}^i_j)$.
   d) Denote the i'te set of secret signing keys and public verification keys as sk^i_{OTS} and vk^i_{OTS} respectively.
   e) Commit to the i'te public verification key by computing $plc_i = \mathsf{H}(\mathsf{vk}^i_1\|\mathsf{vk}^i_2\| \ldots \|\mathsf{vk}^i_t)$. plc_i is then the i'te leaf in the Merkle tree.
3) Publish the root of the Merkle tree, which we denote vk_{MSS}, and set the signature number $i = 0$.

Signing: The input is a message M and the output is a signature σ_{MSS} of M.

1) Increment i.
2) Sign the fingerprint of the message with Winternitz's OTS scheme:
   a) Divide $d = \mathsf{H}(M)$ into t_1 bit strings of length w (we prepend with 0's if needed) and denote them $d_t, d_{t-1}, \ldots, d_{t-t_1+1}$.
   b) Compute the check sum $C = \sum_{u=t-t_1+1}^{t}(2^t - d_u)$.
   c) Divide the binary representation of C into t_2 bit strings of length w (we prepend with 0's if needed) and denote them $d_{t_2}, d_{t_2-1}, \ldots, d_1$.
   d) Regenerate the i'te secret signing keys sk^i_{OTS} with the PRNG R.
   e) Compute the signature: $\sigma_{OTS} = \mathsf{Sig}_{\mathsf{sk}^i_{OTS}}(M) = \{\mathsf{H}^{d_t}(\mathsf{sk}_t), \ldots, \mathsf{H}^{d_2}(\mathsf{sk}_2), \mathsf{H}^{d_1}(\mathsf{sk}_1)\}$.
3) Create the authentication path from the i'te leaf to the root and denote it A^i.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, \mathsf{vk}^i_{OTS}, A^i)$.

Verifying: The input is a message M and a signature σ_{MSS} and the output is a boolean decision true or false.

1) Verify the signature with Winternitz's OTS scheme:
   a) The bit strings $\overline{d_t}, \overline{d_{t-1}}, \ldots, \overline{d_1}$ are computed as in the signing algorithm.
   b) Denote the elements in σ_{OTS} as $\overline{s_j}$ for $j = 1, 2, \ldots, t$.
   c) Verify the signature σ_{OTS} by checking that $\mathsf{H}^{2^w-1-d_j}(\overline{s_j}) = \mathsf{vk}^i_j$ for $j = 1, 2, \ldots, t$ where $\mathsf{vk}^i_j \in \mathsf{vk}^i_{OTS}$.
2) Compute $\overline{plc} = \mathsf{H}(\overline{s_1}\|\overline{s_2}\| \ldots \|\overline{s_t})$.
3) Compute the root of the Merkle tree $\overline{\mathsf{vk}_{MSS}}$ using $\overline{plc}$ and the received authentication path A^i.
4) If $\overline{\mathsf{vk}_{MSS}} = \mathsf{vk}_{MSS}$ then the signature is valid and we output true, otherwise it's invalid and we output false.

Appendix D
The Bleichenbacher-Mauer-Tree signature scheme

The Bleichenbacher-Mauer-Tree signature scheme is a MMS using the tree described in section 2.4 as the OTS scheme. The following is a recap of the Bleichenbacher-Mauer-Tree signature scheme:

Key generation:

1) Choose a secure PRNG R, a secret key $SK \in \{0,1\}^k$ (the seed of R) and a collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^l$.
2) Construct a Merkle tree of height H:
   a) Generate t secret signing keys for each OTS scheme by using R, such that $\mathsf{sk}^i_j = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
   b) Use the hash function H to generate the i'te public verification key vk^i. I.e. we have generated the i'te tree T_i as described in section

2.4 with t secret signing keys as the leafs and the root as the public verification key. T_i represent the OTS scheme for signing a l-bit message, i.e. $\log_2(w((T^*, \leq))) = l$, and the root vk^i in T_i is the i'te leaf in the Merkle tree.

c) Denote the i'te set of secret signing keys as sk_{OTS}^i.

3) Publish the root of the Merkle tree, which we denote vk_{MSS}, and set the signature number $i = 0$.

Signing: The input is a message M and the output is a signature σ_{MSS} of M.

1) Increment i.
2) Sign the fingerprint $d = \mathsf{H}(M)$ with the tree T_i:
   a) Regenerate the i'te secret signing keys sk_{OTS}^i (with the PRNG R) and the tree T_i.
   b) Let A be the largest antichain in T_i and $\mathsf{G} : \mathcal{M} \to A$ be a mapping from the message space $\mathcal{M}$ to the antichain A.
   c) Compute the signature $\sigma_{OTS} = \mathsf{G}(m)$, i.e. the signature is equal one of the signature patterns in A of size t.
3) Create the authentication path from the i'te leaf to the root and denote it A^i.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, \mathsf{vk}^i, A^i)$.

Verifying: The input is a message M and a signature σ_{MSS} and the output is a boolean decision true or false.

1) Verify the message with the tree T_i:
   a) Generate the tree T_i using the signature σ_{OTS} and denote the root $\overline{\mathsf{vk}}$.
   b) Verify the signature by checking that $\overline{\mathsf{vk}} = \mathsf{vk}^i$.
2) Compute the root of the Merkle tree $\overline{\mathsf{vk}_{MSS}}$ using $\overline{\mathsf{vk}}$ and the received authentication path A^i.
3) If $\overline{\mathsf{vk}_{MSS}} = \mathsf{vk}_{MSS}$ then the signature is valid and we output true, otherwise it's invalid and we output false.

APPENDIX E
THE BLEICHENBACHER-MAUER-GRAPH SIGNATURE SCHEME

The Bleichenbacher-Mauer-Tree signature scheme is a MMS using the graph described in section 3.4 as the OTS scheme. The following is a recap of the Bleichenbacher-Mauer-Graph signature scheme:

Key generation:

1) Choose a secure PRNG R, a secret key $SK \in \{0,1\}^k$ (the seed of R) and a collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^l$.
2) Construct a Merkle tree of height H:
   a) Define $B = \lceil \frac{l}{\log_2(p)} \rceil + \lceil \log_p \left(\lceil \frac{l}{\log_2(p)} \rceil \right) \rceil$ where p depend on the value w which define the size of the blocks.
   b) Generate $t = w \cdot (B+1)$ secret signing keys for each OTS scheme by using R, such that $\mathsf{sk}_j^i = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
   c) Use the hash function H to generate the graph G_i as described in section 3.4 and denote the root as the i'te public verification key vk^i. The graph consists of B blocks and the t "leafs" correspond to the secret signing keys. G_i represent the OTS scheme for signing a l-bit message and the root vk^i in G_i is the i'te leaf in the Merkle tree.
   d) Denote the i'te set of secret signing keys as sk_{OTS}^i.
3) Publish the root of the Merkle tree, which we denote vk_{MSS}, and set the signature number $i = 0$.

Signing: The input is a message M and the output is a signature σ_{MSS} of M.

1) Increment i.
2) Sign the fingerprint $d = \mathsf{H}(M)$ with the graph G_i:
   a) Regenerate the i'te secret signing keys sk_{OTS}^i (with the PRNG R) and the graph G_i.
   b) Let A be the largest antichain in G_i and $\mathsf{G} : \mathcal{M} \to A$ be a mapping from the message space $\mathcal{M}$ to the antichain A.
   c) Compute the signature $\sigma_{OTS} = \mathsf{G}(m)$ such that the size of the signature pattern mapped to is l, i.e. the signature is equal one of the signature patterns in A of size l.
3) Create the authentication path from the i'te leaf to the root and denote it A^i.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, \mathsf{vk}^i, A^i)$.

Verifying: The input is a message M and a signature σ_{MSS} and the output is a boolean decision true or false.

1) Verify the message with the graph G_i:
   a) Generate the graph G_i using the vertices in the signature σ_{OTS} and denote the root $\overline{\mathsf{vk}}$.
   b) Verify the signature σ_{OTS} by checking that $\overline{\mathsf{vk}} = \mathsf{vk}^i$.
2) Compute the root of the Merkle tree $\overline{\mathsf{vk}_{MSS}}$ using $\overline{\mathsf{vk}}$ and the received authentication path A^i.
3) If $\overline{\mathsf{vk}_{MSS}} = \mathsf{vk}_{MSS}$ then the signature is valid and we output true, otherwise it's invalid and we output false.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, 1978.
[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1985.
[3] L. Lamport, "Constructing digital signatures from a one-way function," Tech. Rep., 1979.
[4] R. C. Merkle, "A certified digital signature," in *Proceedings on Advances in Cryptology*, ser. CRYPTO '89. Springer-Verlag New York, Inc., 1989.

[5] D. Bleichenbacher and U. Maurer, "On the efficiency of one-time digital signatures," 1996.
[6] D. Bleichenbacher and U. M. Maurer, "Optimal tree-based one-time digital signature schemes," in *In STACS 96: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*. Springer-Verlag, 1996.
[7] I. Damgård, "Definitions and results for authentication systems," Aarhus University, Lecture Notes in Computer Science, 2012.
[8] J. Buchmann, E. Dahmen, and M. Szydlo, "Hash-based digital signature schemes," in *Post-Quantum Cryptography*. Springer Berlin Heidelberg, 2009.
[9] D. Naor, A. Shenhav, and A. Wool, "One-time signatures revisited: Have they become practical?" Cryptology ePrint Archive, Report 2005/442, 2005.
[10] M. Jakobsson, F. T. Leighton, S. Micali, and M. Szydlo, "Fractal merkle tree representation and traversal." in *CT-RSA*, ser. Lecture Notes in Computer Science. Springer, 2003.
[11] C. Dods, N. Smart, and M. Stam, "Hash based digital signature schemes," ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005.
[12] A. Hevia and D. Micciancio, "The provable security of graph-based one-time signatures and extensions to algebraic signature schemes," in *Advances in Cryptology ASIACRYPT 2002*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002.