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 Maurer. 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 \( A \) in Game 1 is negligible in the length of the bit string \( r \).

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

2.2 Hash functions

A hash function \( H : \{0,1\}^* \rightarrow \{0,1\}^k \) 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. \( H : X \rightarrow Y \). 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 \( y = H(x) \). 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 \( H(x) = H(x') \). 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 \( H(x) = H(x') \). 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 $\Sigma = (\text{KGen}, \text{Sig}, Vf)$ is a triple with a key generation algorithm $\text{KGen}$, a signing algorithm $\text{Sig}$ and a verification algorithm $Vf$.

The key generation algorithm $\text{KGen}$ is probabilistic and given the security parameter $k$ as input, $\text{KGen}$ returns the key pair $(sk, vk)$ 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 $\text{Sig}$ is either deterministic or probabilistic and given the secret signing key $sk$ and the message $m$, $\text{Sig}$ returns the signature $\sigma$ of $m$. The verification algorithm $Vf$ is deterministic and given the message $m$ and the signature $\sigma$, $Vf$ returns true if $\sigma$ is a valid signature for $m$. It should always be true that $Vf_{vk}(\text{Sig}_{sk}(m), m) \rightarrow$ true for $(sk, vk) \leftarrow \text{KGen}(k)$ and the message $m$.

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

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

![Fig. 1. The tree $T = [C_2 C_2]$ and its associated poset ($T^*, \leq$). The figure is copied from [6].](image)

2.4 Tree-based one-time signature schemes

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

Let $H : \{0,1\}^* \rightarrow \{0,1\}^k$ 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 leaves 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 leaves 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 $[T_1 T_2]$ 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 leaves are also the leaves 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 leaves 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

$$
(C_n^*, \leq) \cong C_n
$$

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

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

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

E.g. let $T = [C_2 C_2]$ 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 $U \sub W$ or $W \sub U$ and they are incomparable otherwise. A subset $A \sub T^*$ is called an antichain if all
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 $M$ to the antichain $A$, then we can use $A$ as an OTS scheme with signatures as the signature patterns because $A$ satisfies 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 be 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 $sk = (d, e)$ and the public verification key is $vk = a$.

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

Assume the message space is $M = \{0, 1\}$ because $w((T^*, \leq)) = 2$. Now we could define the mapping $G : M \rightarrow A$ 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 $c = H(e)$ 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 have 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 $sk_{OTS}$ and $vk_{OTS}$ for $i = 1, 2, \ldots, 2^H$. Each leaf in the Merkle tree is then the fingerprint of all the public verification keys in $vk_{OTS}$ 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 $H(m)$ of the message, then he sign the fingerprint using the $i$th OTS scheme located at the $i$th leaf which returns the signature $\sigma_{OTS}$. Next he computes the authentication path $A_i$, where the $h$th vertex in $A_i$ is the sibling to the $h'$th vertex in the path from the $i$th 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 $\sigma_{MSS} = (i, \sigma_{OTS}, vk_{OTS}, A_i)$.

Verification of the MSS signature $\sigma_{MSS}$ consists of two steps; first is the OTS scheme with the public verification keys $vk_{OTS}$ used to verify the signature $\sigma_{OTS}$ of $H(m)$. Then is the authentication path $A_i$ used to verify the public verification keys $vk_{MSS}$ 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 $A'$. 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 secure against:

**Theorem 2.** The FMTseq signature scheme is CMA-secure if

1. the used PRNG $R$ is secure, the used hash function $H$ is collision resistant and Merkle’s OTS scheme is CMA-secure.

Prove 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 $A'$ and $A$. $A'$ is trying to break $R$, i.e. tell whether the secret signing keys are truly random bit strings or output from $R$, and $A$ is trying to forge a signature of the FMTseq signature scheme, i.e. a chosen message attack (CMA). Our goal is to construct $A'$ such that he succeed by letting him use $A$ where he doesn’t care about how the CMA is carried out, i.e. we treat $A$ as a black box (hence the name black-box reduction). The construction of $A'$ is illustrated in Figure 3 where the secret signing keys used in the FMTseq signature scheme are generated by the oracle. $A$ outputs 1 if the CMA succeed, i.e. he has forge a signature of the FMTseq signature scheme, and otherwise he outputs 0. Likewise, $A'$ 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 $A$ outputs 0, $A'$ know it’s because the secret signing keys are truly random bit strings and he outputs “random”. Likewise, when $A$ outputs 1, $A'$ know it’s because the secret signing keys are generated by $R$ and he outputs “PRNG”. The above implies that $A'$ can break $R$ which is a contradiction because we assumed it was secure.

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.

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

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 loss 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 $w((T^{*}, 0))$ for a tree $T$ defines the number of bits that can be signed, and given a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ there doesn’t exist a recursive method to compute a tree with $w((T^{*}, 0)) = k$ (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]:
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 $\log_2 \left(\frac{v(388)}{w(\mu(n))}\right) = 160$ 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 do 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 $M$ 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 knows 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.

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 $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ be the used hash function, i.e. we want to sign a $k$-bit message. The graph then consist of $B = \left\lceil \frac{k}{\log_2(w)} \right\rceil + \left\lceil \frac{k}{\log_2(p)} \right\rceil$ 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 $M$ 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 knows 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 leaves 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]:

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

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 $\Sigma$. In [5] is an upper bound on the efficiency for a tree $T$ given as:

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

where $\gamma_T$ is called the tree efficiency constant and they conjecture for a graph $G$ that the graph efficiency constant $\gamma_G$ is:

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

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.

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 $|(T^*, \leq |) = w(n)$, 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 $t = 2^H$ leaves which is also the number of $k$-bit messages that we can sign using $T_{FMTseq}$. Now each leaf in $T_{FMTseq}$ are the root of the tree $T_{Mer}$ representing Merkle’s OTS scheme with $l = k + \lceil \log_2(k) \rceil$ leaves.
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 leaves 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              |
| $[C_2C_1]$   | 6   | 2   | 7              | 2              |
| $[C_2C_1]$   | 7   | 2   | 10             | 2              |
| $[C_2C_3]$   | 8   | 2   | 13             | 3              |
| $[C_3C_2]$   | 9   | 2   | 17             | 3              |
| $[C_3C_2C_2]$| 10  | 3   | 22             | 5              |
| $[C_3C_2C_3]$| 11  | 3   | 31             | 7              |
| $[C_3C_2C_4]$| 12  | 3   | 41             | 8              |
| $[C_3C_3]$   | 13  | 3   | 53             | 9              |
| $[C_3C_4]$   | 14  | 4   | 71             | 12             |
| $[C_3C_2][C_2C_3]$ | 15 | 4 | 101 | 19 |
| $[C_3C_3][C_2C_4]$ | 16 | 4 | 131 | 21 |
| $[C_3C_4][C_2C_4]$ | 17 | 4 | 171 | 24 |
| $[C_2C_2][C_2C_3]$ | 18 | 4 | 222 | 33 |
| $[C_2C_2][C_2C_2][C_2]$ | 19 | 5 | 311 | 51 |
| $[C_2C_2][C_2C_2][C_3]$ | 20 | 5 | 411 | 61 |
| $[C_2C_2][C_2C_2][C_4]$ | 21 | 5 | 531 | 75 |
| $[C_2C_2][C_2C_3][C_2C_3]$ | 22 | 6 | 711 | 91 |
| $[C_2C_2][C_2C_3][C_2C_4]$ | 23 | 6 | 1011 | 141 |
| $[C_2C_2][C_2C_3][C_3C_3]$ | 24 | 6 | 1311 | 163 |
| $[C_2C_2][C_2C_3][C_3C_4]$ | 25 | 6 | 1711 | 222 |
| $[C_2C_2][C_2C_3][C_4C_4]$ | 26 | 6 | 2221 | 251 |
| $[C_2C_2][C_2C_3][C_3C_3][C_3C_3]$ | 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:

$$n_{FMTseq} = t - 1 + t \cdot n_{Mer}$$

$$= t - 1 + t \cdot (3 \cdot l - 1)$$

$$= 3 \cdot l \cdot t - 1$$

(8)

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 were unable to compute the values for trees of size $n > 27$, we are limited to trees of size $n < 30$ 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 $5 \cdot |T_{Mer}^*|, \leq)$ in their appendix for trees of size $n < 30$ 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:

$$\eta(T_{Mer}) = \frac{k}{n_{Mer} + 1}$$

$$= \frac{k}{(3 \cdot l - 1) + 1}$$

$$= \frac{k}{3 \cdot l}$$

(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:

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

(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 $l$ leaves we have to apply the hash function $|H_{T_{Mer}}| = n - l$ times. In
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 \( |sk| = k \cdot l \)-bit and \( |vk| = k \)-bit because each key is \( k \)-bit.

As illustrated in Figure 8 we have to compute \( |H_{\text{Sig}}| = l \) and \( |H_{\text{Ver}}| = l \) 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 \( |H_{\text{Ver}}| = l + (l - 1) \). 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 \( T_{\text{Win}} \) 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 \( T_{\text{Win}} \) is the root of the tree \( T_{\text{Win}} \) representing Winternitz’s OTS scheme. \( T_{\text{Win}} \) has \( l = l_1 + l_2 \) leafs where \( l_1 = \lceil \log_2(t) \rceil \) and \( l_2 = \lceil \log_2(t - l_1) \rceil + 1 \) as defined in [8]. As defined previously \( w \) is the number of bits to be signed simultaneously.

Each \( T_{\text{Win}} \) consists of \( n_{\text{Win}} = l \cdot 2^w + l - 1 \) 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 \( T_{\text{Win}} \) is then:

\[
n_{\text{Win}} = t - 1 + t \cdot n_{\text{Win}} = t - 1 + t \cdot (l \cdot 2^w + l - 1)
\]

#### 4.3.1 Analysis of \( |T_{\text{Win}}^n|, w((T_{\text{Win}}^n, \leq)) \) and \( \eta(T_{\text{Win}}) \)

In Table 6 are the values of \( |T_{\text{Win}}^n|, w((T_{\text{Win}}^n, \leq)) \) and \( \eta(T_{\text{Win}}) \) for trees of size \( n \leq 30 \) computed. It’s clear that Winternitz’s OTS is far away from being optimal according to \( \nu(n) \) 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 \( \log_2(w((T_{\text{Win}}^n, \leq))) \) = \( \log_2(155) \approx 7 \)-bit message.

Finally we use Equation 5 to compute the efficiency of \( T_{\text{Win}}^n \):

\[
\eta(T_{\text{Win}}) = \frac{k}{n_{\text{Win}} + 1} = \frac{k}{(l \cdot 2^w + l - 1) + 1} = \frac{k}{l \cdot 2^w + l}
\]

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 \( \eta(T_{\text{Win}}) \) when using the definition of \( l \) and \( w = 2 \) is:

\[
l_{k \to \infty} \eta(T_{\text{Win}}) = \lim_{k \to \infty} \frac{k}{l \cdot 2^w + l} = \frac{2}{5}
\]

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 \( T_{\text{Win}} \) with \( n \) vertices and \( l \) leafs we have to apply the hash function \( |H_{\text{Win}}| = n - l \) 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 \( |sk| = k \cdot l \)-bit and \( |vk| = k \)-bit. In worst case we need \( |H_{\text{Sig}}| = (2^w - 1) \cdot l \) and \( |H_{\text{Ver}}| = (2^w - 1) \cdot l \) 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 \( |H_{\text{Ver}}| = (2^w - 1) \cdot l + (l - 1) \). Finally the signature consists

---

**Table 6**

| \( k \times n \times l \) | \( |T_{\text{Win}}^n, \leq| \) | \( w((T_{\text{Win}}^n, \leq)) \) |
|---|---|---|
| 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**

<table>
<thead>
<tr>
<th>( k \times \eta(T_{\text{Win}}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
</tbody>
</table>
The number of hash operations used to generate the tree $T_{\text{Mer}}$ ($|H_{T_{\text{Mer}}}| = n - l$), to sign ($|H_{\text{Sig}}| = l$) and verify ($|H_V| = l + (l - 1)$) a $k$-bit message, the size of the secret signing keys ($|sk| = k$), the public verification keys ($|vk| = k$) and the signature ($|\sigma| = k \cdot l$) in bits. $n$ is the number of vertices and $l$ is the number of leaves in $T_{\text{Mer}}$.

| $k$  | $n$  | $l$  | $|H_{T_{\text{Mer}}}|$ | $|sk|$ | $|vk|$ | $|H_{\text{Sig}}|$ | $|\sigma|$ | $|H_V|$ |
|------|------|------|----------------|-------|-------|----------------|-------|-------|
| 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 $T_{\text{BMtree}}$ with $n$ vertices using data from Table 2. The function of the trendline is given in Equation 14.

![Figure 11](image)

Table 5

<table>
<thead>
<tr>
<th>$k$</th>
<th>$n$</th>
<th>$\eta(T_{\text{BMtree}})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>310</td>
<td>0.411575563</td>
</tr>
<tr>
<td>160</td>
<td>330</td>
<td>0.386706949</td>
</tr>
<tr>
<td>256</td>
<td>388</td>
<td>0.41311054</td>
</tr>
<tr>
<td>512</td>
<td>621</td>
<td>0.411575563</td>
</tr>
<tr>
<td></td>
<td>643</td>
<td>0.397515528</td>
</tr>
<tr>
<td></td>
<td>1242</td>
<td>0.411906677</td>
</tr>
<tr>
<td></td>
<td>1267</td>
<td>0.403785489</td>
</tr>
</tbody>
</table>

The number of leaves $\eta(T_{\text{BMtree}})$ 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 $T_{\text{BMtree}}$ for $k = 2$ is illustrated in Figure 11. The size of $T_{\text{BMtree}}$ is then:

$$n_{\text{BMtree}} = t + 1 + t \cdot n_{\text{BMtree}}$$ (15)

4.4.1 Analysis of $|(T_{\text{BMtree}}^*)|$, $\omega((T_{\text{BMtree}}^*)|, \eta(T_{\text{BMtree}})$

As described previous the optimal values of $|(T_{\text{BMtree}}^*)|$ and $\omega((T_{\text{BMtree}}^*)|$ 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 $T_{\text{BMtree}}$ is defined by:

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

In Table 9 is the efficiency of $T_{\text{BMtree}}$ computed using Equation 16 with various values of $k$ and the estimated val-
values 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 $|H_{T_{BMtree}}| = n - l$ 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. $|sk| = k \cdot l$-bit and $|vk| = k$-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. $|H_{Sig}| = |G|$. 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. $|H_{Vf}| = n - l - 1$. 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:

$$n_{BMGraph} = t - 1 + t \cdot n_{BMgraph}$$

$$= t - 1 + t \cdot ((w \cdot (w+1)) \cdot B + 5)$$  \hspace{1cm} (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.

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 $|H_{BMgraph}| = n - (B + 1) \cdot w$ times where $(B + 1) \cdot w$ is the number of "leafs" in the graph (the first row of vertices in each block...
The number of hash operations used to generate the tree $T_{BMtree}$ ($|H_{BMtree}| = n - l$), to sign ($|H_{Sig}| = |H_{G}|$) and verify ($|H_{Vf}| = n - l - 1$) a $k$-bit message in worst case, the size of the secret signing keys ($|sk| = k \cdot l$), the public verification keys ($|vk| = k$) and the signature ($|\sigma| = k \cdot l$) in bits. $|H_{G}|$ 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 leaves in $T_{BMtree}$ where $l$ is estimated using Equation 14.

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

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

| OTS  | $n$ | $l$ | $|(T^\ast, \leq)|$ | $w((T^\ast, \leq))$ |
|------|-----|-----|-----------------|-------------------|
| $T_{Mer}$ | 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.

<table>
<thead>
<tr>
<th>OTS</th>
<th>$n$</th>
<th>$\eta(T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{Mer}$</td>
<td>503</td>
<td>0.317460317</td>
</tr>
<tr>
<td>$T_{Win}$</td>
<td>424</td>
<td>0.376470588</td>
</tr>
<tr>
<td>$(T_{BMtree})_{n=\perp}$</td>
<td>388</td>
<td>0.411311054</td>
</tr>
<tr>
<td>$(T_{BMtree})_{n=\top}$</td>
<td>409</td>
<td>0.390243902</td>
</tr>
<tr>
<td>$G_{BMgraph}$</td>
<td>365</td>
<td>0.437158470</td>
</tr>
</tbody>
</table>

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 $|(T^\ast, \leq)|$, $w((T^\ast, \leq))$ and $\eta(T)$

First we investigate the values of $|(T^\ast, \leq)|$ and $w((T^\ast, \leq))$ 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 the potential to sign a $\log_2(w((T^\ast_{Mer}, \leq))) = \log_2(70) \approx 6$-bit message. Likewise has the tree $T_{Win}$ representing Winternitz’s OTS scheme the potential to sign a $\log_2(w((T^\ast_{Win}, \leq))) = \log_2(155) \approx 7$-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.
The number of hash operations used to generate the graph $G_{BM_{graph}}$, $|H_{G_{BM_{graph}}}| = n - (B + 1) \cdot w$, to sign $|H_{Sig}| = |H_{C}|$ and verify $|H_{Ver}| = (B \cdot w^2 + 2 \cdot (w - 1))$ a $k$-bit message in worst case, the size of the secret signing keys $|sk| = k \cdot (B + 1) \cdot w$, the public verification keys $|vk| = k$ and the signature $|\sigma| = w \cdot (SP \cdot k)$ in bits. $|H_{C}|$ 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 $w \cdot (SP)$ is the size of the largest signature pattern in $G_{BM_{graph}}$.

| $k$ | $n$ | $l$ | $|H_{G_{BM_{graph}}}|$ | $|sk|$ | $|vk|$ | $|H_{Sig}|$ | $|\sigma|$ | $|H_{Ver}|$ |
|-----|-----|-----|-----------------|------|------|-------|-------|--------|
| 128 | 293 | 75  | 218             | 9600 | 128  | $|H_{C}|$ | $w \cdot (SP) \cdot 128$ | 220    |
| 160 | 365 | 93  | 272             | 14880| 160  | $|H_{C}|$ | $w \cdot (SP) \cdot 160$ | 274    |
| 256 | 596 | 144 | 425             | 36864| 256  | $|H_{C}|$ | $w \cdot (SP) \cdot 256$ | 427    |
| 512 | 1121| 282 | 839             | 144384| 512  | $|H_{C}|$ | $w \cdot (SP) \cdot 512$ | 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 $T_{BM_{tree}}$ we have three versions, two as described previously and the last one $(T_{BM_{tree}})_{n=1}^{20}$, as the tree $(T_{BM_{tree}})_{n=8} = [[C_3C_1][C_2C_3][C_5C_4]]$ 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 $(T_{BM_{tree}})_{n=1}^{20}$ 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 $T_{Merk}$ has almost twice the size of signature and secret signing keys compared to the others. That $(T_{BM_{tree}})_{n=L}$ and $(T_{BM_{tree}})_{n=\top}$ 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 $(T_{BM_{tree}})_{n=8}^{20}$. Otherwise if we want a small tree, i.e. we want few hash operations to generate the tree and verify messages we should use $G_{BM_{graph}}$ but only if $|H_{C}|$ is cheap. Else we have to use $T_{Win}$ which also have short signatures and keys because of its few leaves 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 $|H_{C}|$ 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 $\sigma_{OTS}$ of Merkle’s OTS scheme or find a collision for the hash function $H : \{0,1\}^* \to \{0,1\}^k$. 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 $\sigma_{OTS}$ using Merkle’s OTS scheme. $O$ is given the security parameter $k$ as input:

   a) Generates a key pair by running the probabilistic algorithm $KGen: (sk_{Mer},vk_{Mer}) \leftarrow KGen(k)$. $O$ 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 $H : \{0,1\}^* \to \{0,1\}^k$.

   b) Selects an index $i \in [1, \ldots, 2^H]$.

   c) Generates $2^H$ key pairs by running the probabilistic algorithm $KGen: (sk_i^{OTS},vk_i^{OTS}) \leftarrow KGen(k)$, where $i = 1, \ldots, 2^H$ is the signature number and each key pair is used to sign and verify one message.

   d) Update the $\epsilon_i$ public verification key to the one received from $O$: $vk_i^{MSS} = vk_{Mer}$.

   e) Sends the set of public keys $vk_i^{OTS}$, 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:

\begin{verbatim}
|\sigma| \hspace{1cm} \forall i \in [1, \ldots, 2^H] : \mathcal{A} \rightarrow \mathcal{F} \hspace{1cm} \text{sign}(H(i), \sigma) \leftarrow \mathcal{A}
\end{verbatim}
TABLE 16
Comparison of the trees with \( n \) vertices and \( l \) leaves 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       | \( n \) | \( l \) | \( |H_T| \) | \( |sk| \) | \( |vk| \) | \( |\sigma| \) | \( |H_{\sigma}| \) |
|-----------|--------|--------|----------|--------|--------|--------|----------|
| \( T_{\text{Mer}} \) | 503    | 135    | 335      | 26880  | 160    | 168    | 26880    | 335      |
| \( T_{\text{Win}} \) | 424    | 85     | 339      | 13600  | 160    | 255    | 13600    | 339      |
| \( (T_{\text{BMtree}})_{n=1} \) | 388    | 97     | 291      | 15520  | 160    | 15520  | 15520    | 290      |
| \( (T_{\text{BMtree}})_{n=7} \) | 409    | 102    | 307      | 16320  | 160    | 16320  | 16320    | 306      |
| \( (T_{\text{BMtree}})_{n=8} \) | 520    | 120    | 110      | 400    | 160    | 960    | 960      | 380      |
| \( G_{BMgraph} \) | 365    | 93     | 272      | 14880  | 160    | 160    | 160      | 274      |

a) \( F \) sends a message \( M \) and a signature number \( q \) to \( A \) where \( 1 \leq q \leq 2^d \).

b) If \( q = c \) then \( A \) asks the oracle \( O \) for a signature of \( M \). \( O \) computes the signature by \( \sigma_{\text{OTS}} = \text{Sig}_{\text{OTS}}(M) \) and sends it to \( A \). \( A \) then forwards \( \sigma_{\text{OTS}} \) to \( F \).

c) Else if \( q \neq c \) then \( A \) computes the signature of \( M \) using Merkle’s OTS scheme by \( \sigma_{\text{OTS}} = \text{Sig}_{\text{OTS}}(m) \) and sends it to \( F \).

d) With \( \varepsilon_F \) the forger \( F \) returns a forged signature of the FMTseq signature scheme, i.e. he returns to \( A \) a message \( M \neq M \) and a signature \( \sigma_{\text{MSS}} = (s,\sigma_{\text{OTS}},\text{vk}_{\text{MSS}},A^s) \) where \( s \) is the signature number and \( \text{Vf}_{\text{MSS}}(\mathcal{M},\sigma_{\text{MSS}}) \rightarrow \text{true} \) for the root \( \text{vk}_{\text{MSS}} \) of the Merkle tree.

4) The adversary \( A \):

a) Compares the received signature \( \sigma_{\text{MSS}} \) of \( M \) with the signature \( \sigma_{\text{MSS}} = (s,\sigma_{\text{OTS}},\text{vk}_{\text{MSS}},A^s) \) of \( M \).

b) If \( (\text{vk}_{\text{MSS}}^s,A^s) \neq (\text{vk}_{\text{MSS}}^s,A^s) \) then \( A \) returns a collision for the hash function \( H \).

c) Else if \( (\text{vk}_{\text{MSS}}^s,A^s) = (\text{vk}_{\text{MSS}}^s,A^s) \) and \( s = c \) then \( A \) returns a forged signature of Merkle’s OTS scheme.

A.3. \( A \)’s advantage

Either \( A \) finds a collision for \( H \) or he forges a signature for Merkle’s OTS scheme. \( A \)’s probability \( \varepsilon_{CR} \) for returning a collision is at least \( \varepsilon_F \). Likewise is \( A \)’s probability \( \varepsilon_{OTS} \) for returning a forged signature at least \( \frac{1}{n} \cdot \varepsilon_F \) because it depends on whether \( s = c \) or not where \( \Pr[s = c] = \frac{1}{n} \).

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 \( H : \{0,1\}^* \rightarrow \{0,1\}^l \).

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 \( sk_j^i = R(SK,j,i) \) where \( j = 1,2,\ldots,t \) and \( i = 1,2,\ldots,2^j \).

b) Use the hash function \( H \) to compute the \( t \) public verification keys by \( \text{vk}_j^i = H(sk_j^i) \).

c) Denote the \( i'th \) set of secret signing keys and public verification keys as \( sk_{\text{OTS}}^i \) and \( \text{vk}_{\text{OTS}} \) respectively.

d) Commit to the \( i'th \) public verification key by computing \( plc_i = H(\text{vk}_j^i||\text{vk}_j^i||\ldots||\text{vk}_j^1) \). \( plc_i \) is then the \( i'th \) leaf in the Merkle tree.

3) Publish the root of the Merkle tree, which we denote \( \text{vk}_{\text{MSS}} \), and set the signature number \( i = 0 \).

Sizing: The input is a message \( M \) and the output is a signature \( \sigma_{\text{MSS}} \) 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 \( H(M) \) and denote it \( C \). Let \( d = H(M)\|C \).

b) Regenerate the \( i'th \) secret signing keys \( sk_j^i \) with the PRNG \( R \).

c) Compute the signature: \( \sigma_{\text{OTS}} = \text{Sig}_{\text{OTS}}(M) = \{sk_j^i \in \text{sk}_{\text{OTS}}|d_j = 1\} \cup \{ \text{vk}_j^i \in \text{vk}_{\text{OTS}}|d_j = 0\} \).
3) Create the authentication path from the i’te leaf to the root using [10] and denote it $A_i$.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, vk_{OTS}, A_i)$.

Verifying: The input is a message $M$ and a signature $\sigma_{MSS}$ 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 $H(M)$ and denote it $C$. Let $d = H(M) \| C$.
   b) Let $J = \{j | d_j = 1\}$ be indices and denote the elements in $\sigma_{OTS}$ as $\pi_j$ for $j = 1, 2, \ldots, t$.
   c) Update the elements in $\sigma_{OTS}$ corresponding to the secret signing keys by $\pi_j = H(\pi_j)$ for all $j \in J$.
   d) Verify the signature by checking that $\pi_j = vk_j$ for $j = 1, 2, \ldots, t$ where $vk_j \in vk_{OTS}$.
2) Compute $plc = H(\pi_1, \pi_2, \ldots, \pi_t)$.
3) Compute the root of the Merkle tree $vk_{MSS}$ using $plc$ and the received authentication path $A_i$.
4) If $vk_{MSS} = \sigma_{MSS}$ 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 $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$.
2) Construct a Merkle tree of height $H$:
   a) Define $t = t_1 + t_2$ where $t_1 = \lceil \log_2(t_1) \rceil$ and $t_2 = \lceil \log_2(t_1) + 1 + w \rceil$. 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 $sk_{OTS}^i = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
   c) Use the hash function $H$ to compute the $t$ public verification keys by applying it $2^w - 1$ times: $vk_{OTS}^i = H^{2^w - 1}(sk_{OTS}^i)$.
   d) Denote the $i$’te set of secret signing keys and public verification keys as $sk_{OTS}^i$ and $vk_{OTS}^i$ respectively.
   e) Commit to the $i$’te public verification key by computing $plc_i = H(vk_{OTS}^i, \| vk_{OTS}^j, \| \ldots \| vk_{OTS}^j)$. If $plc_i$ 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 $\sigma_{MSS}$ of $M$.

1) Increment $i$.
2) Sign the fingerprint of the message with Winternitz’s OTS scheme:
   a) Divide $d = H(M)$ into $t_1$ bit strings of length $w$ (we prepend with 0’s if needed) and denote them $d_1, d_{t_1-1}, \ldots, d_{t_1}+1$.
   b) Compute the check sum $C = \sum_{d_i \in d_1 \ldots d_{t_1}+1} (2^l - d_i)$. 
   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 $d_2, d_{t_2-1}\ldots d_{t_2}+1$.
   d) Regenerate the i’te secret signing keys $sk_{OTS}^i$ with the PRNG $R$.
   e) Compute the signature: $\sigma_{OTS} = \text{Sig}_{sk_{OTS}}(M) = \{H_{d_i}(sk_i), \ldots, H_{d_{t_2}}(sk_{t_2}), H_{d_{t_1}}(sk_{t_1})\}$.
3) Create the authentication path from the i’te leaf to the root and denote it $A_i$.
4) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, vk_{OTS}^i, A_i)$.

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

1) Verify the signature with Winternitz’s OTS scheme:
   a) The bit strings $d_1, d_{t_1}, \ldots, d_{t_1}$ are computed as in the signing algorithm.
   b) Denote the elements in $\sigma_{OTS}$ as $\pi_j$ for $j = 1, 2, \ldots, t$.
   c) Verify the signature $\sigma_{OTS}$ by checking that $H^{2^w - 1 - d_i}(\pi_j) = vk_{OTS}^j$ for $j = 1, 2, \ldots, t$ where $vk_{OTS}^j \in vk_{OTS}$.
2) Compute $plc = H(\pi_1, \pi_2, \ldots, \pi_t)$.
3) Compute the root of the Merkle tree $vk_{MSS}$ using $plc$ and the received authentication path $A_i$.
4) If $vk_{MSS} = \sigma_{MSS}$ 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 $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$.
2) Construct a Merkle tree of height $H$:
   a) Generate $t$ secret signing keys for each OTS scheme by using $R$, such that $sk_{OTS}^i = R(SK, j, i)$ 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...
The Bleichenbacher-Mauer-Tree signature scheme is a MMS and the output is a boolean decision true or false.

Verify: The input is a message $M$ and a signature $\sigma_{MSS}$ 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 $\sigma_{OTS}$ and denote the root $\overline{v_k}$.
   b) Verify the signature by checking that $\overline{v_k} = v_k^i$.

2) Compute the root of the Merkle tree $\overline{v_{MSS}}$ using $\overline{v_k}$ and the received authentication path $A'$. 
3) If $\overline{v_{MSS}} = v_{MSS}$ 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 $H : \{0,1\}^\ast \rightarrow \{0,1\}^l$.
2) Construct a Merkle tree of height $H$:
   a) Define $B = \left\lceil \log_2(p) \right\rceil + \left\lceil \log_p \left( \left\lceil \log_2(p) \right\rceil \right) \right\rceil$ 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 $sk^i_j = R(SK, j, i)$ where $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, 2^H$.
   c) Use the hash function $H$ to generate the graph $G_i$ as described in section 3.4 and denote the root of the $i$’te public verification key $vk$. The graph consists of $B$ blocks and the $t$ “leaves” 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 $sk^i_{OTS}$.

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

Sign: The input is a message $M$ and the output is a signature $\sigma_{MSS}$ of $M$.

1) Increment $i$.
2) Sign the fingerprint $d = H(M)$ with the tree $T_i$:
   a) Regenerate the $i$’te secret signing keys $sk^i_{OTS}$ (with the PRNG $R$) and the tree $T_i$.
   b) Let $A$ be the largest antichain in $T_i$ and $G : M \rightarrow A$ be a mapping from the message space $M$ to the antichain $A$.
   c) Compute the signature $\sigma_{OTS} = G(m)$.
   d) Output the signature $\sigma_{MSS} = (i, \sigma_{OTS}, vk^i, A)$.

Verifying: The input is a message $M$ and a signature $\sigma_{MSS}$ 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 $\sigma_{OTS}$ and denote the root $\overline{v_k}$.
   b) Verify the signature by checking that $\overline{v_k} = v_k^i$.

2) Compute the root of the Merkle tree $\overline{v_{MSS}}$ using $\overline{v_k}$ and the received authentication path $A'$. 
3) If $\overline{v_{MSS}} = v_{MSS}$ then the signature is valid and we output true, otherwise it’s invalid and we output false.

REFERENCES

digital signatures,” 1996.

time digital signature schemes,” in *In STACS 96: Proceedings of the
13th Annual Symposium on Theoretical Aspects of Computer Science.* Springer-Verlag, 1996.

Aarhus University, Lecture Notes in Computer Science, 2012.

signature schemes,” in *Post-Quantum Cryptography.* Springer

Have they become practical?” Cryptology ePrint Archive, Report

merkle tree representation and traversal,” in *CT-RSA,* ser. Lecture

schemes,” ser. Lecture Notes in Computer Science. Springer
Berlin Heidelberg, 2005.

based one-time signatures and extensions to algebraic signature
schemes,” in *Advances in Cryptology ASIACRYPT 2002,* ser. Lecture