One-Time Signatures Revisited: Have They Become Practical?

Dalit Naor* Amir Shenhav† Avishai Wool‡

December 2, 2005

Abstract

One-time signatures have been known for more than two decades, and have been studied mainly due to their theoretical value. Recent works motivated us to examine the practical use of one-time signatures in high-performance applications. In this paper we describe FMTseq — a signature scheme that merges recent improvements in hash tree traversal into Merkle's one-time signature scheme. Implementation results show that the scheme provides a signature speed of up to 35 times faster than a 2048-bit RSA signature scheme, for about one million signatures, and a signature size of only a few kilobytes. We provide an analysis of practical parameter selection for the scheme, and improvements that can be applied in more specific scenarios.

1 Introduction

1.1 Motivation

Commonly used digital signature algorithms, like RSA, are not sufficiently fast for many applications. As an example application, consider a financial news service that wishes to sign all its messages: such a service requires both low latency and high bandwidth. An efficient alternative to a digital signature is a Message Authentication Code (MAC). However, a MAC does not provide the asymmetry between the signer and the verifier which digital signatures provide — anyone who can verify the signature can also sign.

One-time signatures [Mer89], on the other hand, are digital signatures that are based on one-way functions without a trapdoor, thus they are much faster. One-time signatures have been known for more than two decades, but are usually considered to be impractical. This is due to their complex key management problems, and the fact that their signature length is significantly larger.

Over the last few years, there has been some increased interest in one-time signatures, mainly due to the multicast authentication problem (cf. [Per01]). New signature schemes have been presented, with key management solutions for some limited scenarios. These techniques and other efficient algorithms that improved key management aspects led us to re-examine whether one-time signatures can be made practical for high-performance tasks.

*IBM Haifa Research labs, Tel Aviv, Israel. dalit@il.ibm.com.

†School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel. amir.shenhav@gmail.com.

‡School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel. Supported in part by an IBM faculty award. yash@acm.org.

1.2 Related Work

One-time signatures have been known for a relatively long time. They were first presented by Lamport [Lam79] and Rabin [Rab78], but are mostly known in the form presented by Merkle and Winternitz [Mer87]. These signatures are based on one-way functions, as opposed to trapdoor functions that are used in public key signatures like RSA and ElGamal.

Although one-way functions are simpler to implement and are typically more computationally efficient than trapdoor functions, one-time signatures have been considered to be impractical for two main reasons: First, their "one-timed-ness", i.e., key generation is required for each usage, thus implying a complicated key management scheme; Second, the signature size is relatively long in comparison with common public-key signatures and MACs.

Bos and Chaum [BC92], and later, Bleichenbacher and Maurer [BM94, BM96b, BM96a] formalized a generalization of the problem and suggested signatures based on acyclic graphs. Even, Goldreich and Micali [EGM89] combine one time signatures and public key signatures to form a hybrid scheme. They introduced an on-line/off-line scheme which provides a medium sized signature whose message-dependent signature computation is short, but the preparations for the next signature, as well as signature verification, are as slow as a public key signature. Improvements to this work were presented by Rohatgi [Roh99], who focused mainly on reducing the signature size.

Recently, this area was revisited due to the need for fast, low computation, authentication solutions for IP-multicast and sensor networks. Perrig [Per01] presents a new one-time signature scheme that aims for very fast verification at the cost of signature time and key size. Reyzin and Reyzin [RR02] introduce a different scheme that has faster signature and verification times (for a single signature). This scheme is further improved by Pieprzyk et al. [PWX03]. These new schemes suggest the reuse of the same keys more than once. These "several times signatures" have a known compromise probability after a given number of reuses. Nevertheless, even if these schemes can sign several messages with the same keys, with reasonable security, the number of repeated uses is still small for all practical scenarios. Therefore, efficient key generation and management for a large sequence of one-time signatures still remains on open problem.

Merkle [Mer87, Mer89] addressed the problem of key management, introducing the method of *tree authentication*. In [Mer89], originally published in 1979, the concept of a *hash tree* is presented, which provides efficient key management for a large number of one-time signatures. In [Mer87], an *infinite tree signature* is presented, which theoretically enables the creation of an unlimited number of one time signatures. This latter scheme is impractical as the signature size is very large and grows (logarithmically) with the number of signatures. The recent works of [JLMS03, Szy04a, BKN04] improve Merkle's hash-tree method [Mer89] and give the ability to handle large hash trees more efficiently. Jakobsson et al.'s suggestion for using their algorithm for one-time signatures was implemented in [Col03]. However, the signature scheme that was implemented differs from Merkle's [Mer89] proposal, and the results obtained by [Col03] do not encourage the use of the scheme.

Perrig [Per01] addresses the key generation problem in a different way, by using *hash chains* for key management. However, his scheme provides authentication only for scenarios in which the signer and the verifiers are time synchronized, and it is difficult to adopt this solution for general scenarios.

Recently, a work of Seys and Preneel [SP05] provided a power consumption estimation of one-

time signatures schemes, for low power mobile platforms. Their work estimates the overall power consumption of the signature schemes of Winternitz and Reyzin, with the key management techniques of Jakobsson et al. [JLMS03] and Perrig [Per01]. They take into account other aspects of the signature schemes, including key generation and communication costs. Their estimations are based on measurements of the power consumption of AES block cipher and radio communications on a given mobile platform.

1.3 Contributions

Our first contribution is a design of a signature scheme we call FMTseq - Fractal Merkle Tree sequential signatures. FMTseq combines Merkle's one-time signatures with Jakobsson et al.'s algorithm for hash tree traversal. Our scheme differs from [JLMS03] by providing many more one-time signatures with the same hash tree[1]. Also, the main focus of [JLMS03] was to reduce the run-time space requirements of their scheme. We argue that many applications would prefer trading a few additional kilobytes of signer run-time storage to obtain faster signature rates. Thus, we focus on minimizing the signature speed and signature size – while keeping the run-time space reasonable (if not optimal).

Next, we provide an efficient implementation of a scheme that significantly improves the poor performance that was reported in [Col03] (see also [Szy04b]). Our experimental results show that FMTseq is one- or two-orders of magnitude faster than RSA, with low signature sizes and signer storage requirements. We also show that the parameter values that produce the fastest signature times are different from those suggested in [JLMS03].

For scenarios in which repudiation and adaptive chosen-message attacks are not significant threats, we suggest using Bellare and Rogaway's Target-Collision-Resistant functions [BR97] instead of regular hash functions. Our analysis shows that this improves the signature speed and size by at least another 50%.

Finally we present a number of improvements that address forward security, online/offline variants, and allowing an unbounded number of signatures. We believe that our constructions bring one-time signatures into the realm of the practical, at least for some application areas.

Organization In the next section we provide a brief review of one-time signatures and an overview of the fractal Merkle tree traversal algorithm. In Section 3 we describe our construction for *Fractal Merkle Tree sequential one-time signatures*. Section 4 provides experimental results that are analyzed and discussed for trade-offs. Improvements for our construction that are attractive in more specific scenarios are presented in Section 5. In Section 6 we present our conclusions.

[1] Actually, our construction follows Merkle's original suggestion for using the hash tree.

Preparation	**Signature**	**Verification**
The signer: 1. Generates a set of t random secrets $\{sk_i\}_{i=1}^{t}$, where $t = l + \lceil \log_2 l \rceil$. 2. Calculates: $\{h(sk_i)\}_{i=1}^{t}$ a commitment on each secret value. $h()$ is a one-way hash function. 3. Gives the commitments, in an authenticated way, to the verifier.	Input: l-bit message M. 1. Calculate: $M' = M \| Checksum$. *Checksum* is the number of $'0'$ bits in M. 2. The signature: Output all the secrets sk_i that correspond to the $'1'$ bits of M'.	1. Calculate the checksum to get M'. 2. Check if exactly the appropriate secrets were revealed (for all the $'1'$ bits in M). 3. Check if hashing each secret sk_i provides its claimed commitment.

Figure 1: Merkle's one-time signature scheme for a message M of length l bits.

2 One-Time Signatures and Authentication Trees

2.1 One-Time Signatures

In this section we introduce the notion of one-time signatures. A comprehensive presentation of one-time signatures can be found in [Gol04].

One-time signatures are digital signature schemes that enable a signer to sign a single message with a given set of keys. One-time signatures require the signer to generate different keys for each message to be signed, in contrast with "regular" digital signatures like RSA, whose keys can be used for an unlimited number of times.

One-time signature keys can be viewed as a set of public *commitments* to a set of *secrets* chosen by the signer prior to signing a message. These commitments are given to the verifier in an authenticated manner. To sign a message, the signer reveals a subset of his secrets that correspond to the message content. The verifier determines the message authenticity by checking if the appropriate secrets were revealed, and verifies their correspondence to the commitments that were given earlier. The one-time signature is *secure* if an attacker, given a signature for a chosen message, can provide a valid signature for a different message with non-negligible probability.

2.1.1 Merkle's One-Time Signature

The best known one-time signature scheme was presented by Merkle in [Mer87], and is an improvement of Lamport's [Lam79] signature scheme. Merkle's scheme is described in Figure 1.

It is quite easy to be convinced that an adversary cannot alter the message without invalidating the signature: to do so he must publish a secret that corresponds to a bit either in the message or its checksum, thus demonstrating the ability to reverse the one-way function.

To sign an arbitrary length message, a message digest should be calculated using a hash function, and the scheme should be applied to the message digest. The security of the scheme is then implied by the irreversibility of the one-way function and the collision-resistance of the message digest

function. In practice, a message digest could be the SHA-1 [Nat95] function that has a 160 bits output, or SHA-256 [Nat02]. Therefore to sign this output one needs $t = 168$ or $t = 264$ secrets.

2.1.2 Other One-Time Signature Schemes

A one-time signature scheme called BiBa is presented in [Per01]. This scheme aims to lower the signature bandwidth and verification complexity, but requires a high computation effort at the signer side. It has a probabilistic nature that enables the re-use of the same keys for several signatures, but with a security degradation for each additional key re-use. In [RR02], a "better than BiBa" scheme is presented (also called HORS), that provides fast signature and verification, and keeps the short length of the signature, but is still probabilistic in nature.

However, both of these schemes require a larger number of secret values per signature (e.g., 1024 secret values per signature are recommended by [Per01]), although only a small part of them are exposed. Therefore, the key management problem, i.e., committing to a large number of secrets, remains a major concern. Perrig [Per01] suggests using a hash chain as a key management mechanism, such that each secret is a node in a separate chain, and its commitment is the preceding node. After each one-time signature, all the secrets are exposed and become the commitments for the next one-time signature. The disclosure of all the secrets makes the previous signature insecure, and thus, [Per01] uses a time synchronization mechanism to ensure that only the signer could have known the secrets that were used. This technique is applicable to the scenario of broadcast authentication, but not as a general digital signature.

2.2 Authentication Trees

2.2.1 Fractal Merkle Trees

Merkle [Mer89] introduced the idea of using a hash tree to authenticate a large number of one-time signatures (see the Appendix A.1 for details). An important notion in Merkle trees is that of an *authentication path* — the values of all the nodes that are siblings of nodes on the path between a given leaf and the root. Basically, the signer provides the authentication path with each signature.

Jakobsson et al. [JLMS03] present a new scheme for the sequential traversal of such a Merkle hash tree, i.e., providing the authentication path for each leaf when the leaves are used one after the other. The scheme requires a computational effort of $2 \log N / \log \log N$ and a run-time space of $1.5 \log^2 N / \log \log N$ nodes. Since this scheme is central to our work, we briefly describe it here.

Notation

A hash tree T of height H is divided into L *levels*, each of height h. The leaves of the hash tree are indexed $\{0, 1, ..., 2^H - 1\}$ from left to right. The *altitude* of a node n is defined as the height of the maximal subtree for which it is the root and ranges from 0 (for the leaves), to H (for the root). An *h-subtree* is "at level i" when the altitude of its root is ih for some $i \in \{1, 2, ..., L\}$. For each i there are 2^{H-ih} such h-subtrees at level i. A series of h-subtrees $\{Tree_i\}_{i=1}^{L}$ is a *stacked series* if for all $i < L$ the root of $Tree_i$ is a leaf of $Tree_{i+1}$. The notations are illustrated in Figure 2.

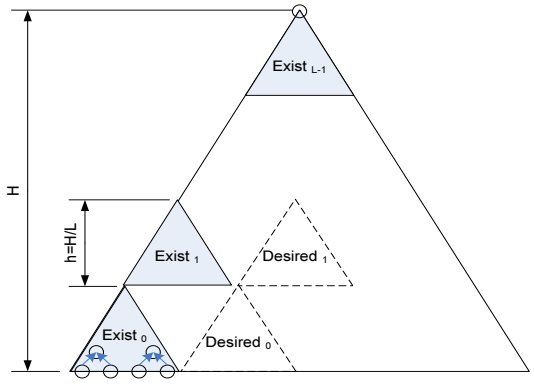


Figure 2: Fractal merkle tree notations

2.2.2 Algorithm Intuition

The goal of [JLMS03] algorithm is to provide the authentication path of the current leaf (signature) with minimal storage and computation. Instead of storing the whole hash tree at the signer side, [JLMS03] stores only two sets of subtrees:

Exist subtrees are a stacked series of h-subtrees, indexed $\{Exist_i\}_{i=1}^{L}$, that include the authentication path for the current signature.

Desired subtrees are a stacked series of h-subtrees, indexed $\{Desired_i\}_{i=1}^{L-1}$. Each *Desired* subtree is adjacent to the an *Exist* subtree of the same level. When an *Exist* subtree no longer contains the next authentication path, it is replaced with its *Desired* subtree counterpart. The *Desired* subtrees are built incrementally after each output of the algorithm, thus *amortizing* the operations required to evaluate the subtree.

In the key generation phase, the signer initializes all the *Exist* and *Desired* subtrees and evaluates the root of the hash tree. The signer discards all the computed values except the *Exist* subtrees. The root of the hash tree can then be handed to the verifier in an authenticated manner. Note that all the hash tree nodes are evaluated during the key generation phase. Therefore, the overall computation effort requires 2^H leaf evaluations and $2^H - 1$ hash computations.

For each output operation (signature) the signer also updates its run-time data structures in preparation for future signatures by performing two operations for every *Desired* subtree. Each operation can be either a calculation of an internal node or a leaf evaluation. For a detailed description of the algorithm refer to [JLMS03].

3 Fractal Merkle Tree Sequential One-Time Signatures

In this section we describe our scheme for sequential one-time signatures using fractal Merkle tree algorithm. In our scheme, the secrets of each one-time signatures are generated by a pseudo-random number generator. The value of each leaf of the fractal Merkle tree is a hash over all the

commitments of a single one-time signature. Therefore, *each leaf serves as a public commitment to a one-time signature*[2]. For each one-time signature, the signer regenerates the next unused leaf, reveals the required secrets, and outputs the commitments of the unrevealed secrets and the authentication path. We call this scheme FMTseq: Fractal Merkle Tree Sequential One-Time Signature.

The FMTseq scheme consists of three algorithms: key generation, signing and verification. The scheme is described in Figure 3.

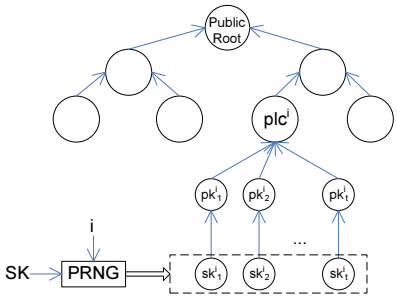
Key generation and initialization

Definitions:

1. SK - a k-bit secret key.
2. sk_j^i - the secrets of each one-time signature where i is the signature number, and j is the index of the secret, $j = 1, \ldots, t$.
3. $sk_j^i = R(SK, i, j)$ R is a pseudo-random number generator (PRNG).
4. $pk_j^i = h(sk_j^i)$ - the commitments for each sk_j^i.
5. *Public leaf commitment* - $plc^i = h(pk_1^i | \ldots | pk_t^i)$, the hash of all the commitments of a single one-time signature.

Computation:

1. Input: a secret, k-bit key, SK.
2. Initialize a Fractal Merkle hash tree of height H, with the plc^i values as its leaves ($i = 1, \ldots, 2^H$).
3. Publish the tree root (or provide it to the verifiers securely).
4. Set Signature number $i = 0$.



FMTseq key generation phase.

Signing a message

Input: a message M.

1. Increment i.
2. Calculate a message digest $md = g(M)$ and C=number of $'0'$-bits in md.
3. Sign m with Merkle's one-time signature scheme:
   - Let $m = md \| C$,
   - Let $J = \{j | m_j = 1\}$ and $\bar{J} = \{j | m_j = 0\}$
   - Generate $\{sk_j^i\}_{j=1}^t$ using the PRNG R.
   - Output $S = \left\{ sk_j^i \; \forall j \in J \, , pk_j^i \; \forall j \in \bar{J} \right\}$.
4. Output $\{ap^i\}$, the authentication path of leaf i, using fractal Merkle tree algorithm [JLMS03].
5. Perform the update operations of [JLMS03].

Verifying a message

Input: a message M and a signature S, $\{ap^i\}$.

1. Calculate a message digest $md = g(M)$. and C=number of $'0'$-bits in md.
2. Verify the Merkle one-time signature:
   - Let $m' = md \| C$.
   - Let $J = \left\{ j | m'_j = 1 \right\}$.
   - Set $S' = S$, and denote the members of S' by $S' = \{s'_j\}_{j=1}^t$.
   - Calculate and update $s'_j \leftarrow h(s'_j)$, $\forall j \in J$.
   - Calculate $plc' = h(s'_1 | \ldots | s'_t)$
3. Iteratively hash the result, plc' with the authentication path.
4. Compare the result to the public root of the tree.

Figure 3: The FMTseq scheme algorithms and data structure.

The performance in FMTseq is dominated by the leaf calculation time, which is defined as the time to generate all the secrets, commit each one of them with a one-way hash function, and hash all these commitments to one value. Therefore, to minimize the leaf calculation time, we

[2]In contrast to the suggestion of Jakobsson et al. [JLMS03] that each leaf serves as a commitment to a single secret, i.e., to a single bit.

Key length	Public exponent	Signature size	Signature time	Verification time	Key generation time
1024 bits	17	128 Bytes	5.9 mSec	0.24 mSec	0.15 Sec
2048 bits	17	256 Bytes	33.8 mSec	0.6 mSec	1.2 Sec

Table 1: RSA parameters and performance as computed on our system (Pentium IV, 1.7GHz with Windows XP).

chose the basic Merkle's one-time signature scheme. Winternitz's improvement to Merkle's one-time signature [Mer87] reduces the signature size but requires more hash calculations, thus, it increases the leaf calculation time. Taking a different one-time signature scheme, like BiBa or HORS, requires a larger number of secrets, and therefore also increases the leaf calculation time significantly. Moreover, within the FMTseq construction, using BiBa or HORS does not provide shorter signatures, since *all* the commitments of each one-time signatures need to be provided to the verifier with every signature (not only the secret ones as these schemes assume).

4 Experimental Results

4.1 Implementation

We implemented the FMTseq scheme in C and tested its performance on a Pentium IV, 1.7GHz, running Microsoft Windows XP. The implementation of RC4, MD5, SHA-1 and SHA-256 was based on code from [Dev05]. The MD5 code was slightly optimized to achieve a more efficient hash for the committing function[3]. The RC4 stream cipher was used as a pseudo-random number generator. The program ran with real-time priority, measuring time using operating system time functions.

We compared the FMTseq results to those of a reference public-key signature. We chose to compare with the popular RSA signature. The RSA parameters and performance values were measured on the same platform using the Crypto++ code library [Dai04]. The RSA results on our system are listed in Table 1.

4.2 Selecting Hash Functions for FMTseq

The hash functions that are used in our scheme determine its performance (both speed and signature size) as well as security. We selected the hash functions so that they provide a level of security that is comparable with RSA signatures. According to [Len01], 2048-bit RSA is approximately equivalent to a 128-bit key of a symmetric cipher, and 1024-bit RSA is approximately equivalent to a 75-bit symmetric key.

Recall that cryptographic hash functions provide two notions of security:

- Second pre-image resistance - for a *given* hash $h(x)$ it is infeasible to find another pre-image x' such that $h(x') = h(x)$.

[3]Since we hash short bit strings we could remove some code that supports arbitrarily long strings.

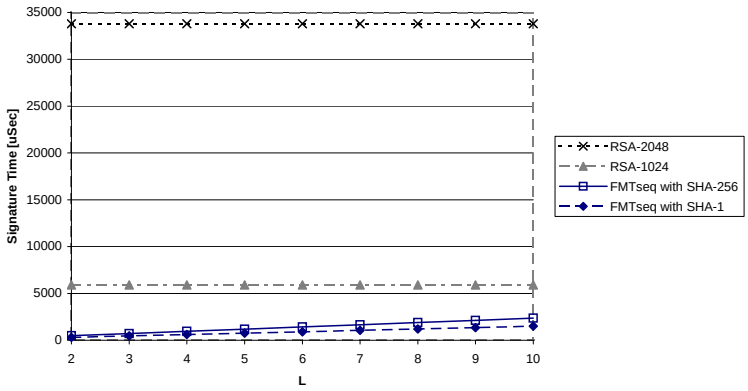


Figure 4: Average signature time as a function of the number of levels L in the FMTseq tree for $L = 1$ to 10, in comparison with RSA signature times. Each point is an average of 128 runs, each consisting of 128 sequentially generated signatures over randomly chosen messages.

- Collision resistance - it is infeasible to find any two values x, x' such that $h(x') = h(x)$.

For the FMTseq message digest hash function, $g()$, collision resistance is required. Thus, a fair comparison should match SHA-1 to the 1024-bit RSA, and SHA-256 to the 2048-bit RSA. The number of secrets for each one-time signature will be 168 or 264 respectively.

For committing secrets and for hash operations in the Merkle hash tree, we argue that the security level is determined by the second pre-image resistance and not by the collision-resistance property of the hash function. Therefore, we argue that MD5 is sufficient as the one-way hash function, $h()$, since its 128-bit output provides the required security level, and its software implementation is almost twice as fast as SHA-1. If one uses SHA-1 instead of MD5 then the performance becomes twice slower and both signature size and run-time space grow by 25%.

We note that the recent attacks on both SHA-1 and MD5 [WY05, BC04, BCJ$^+$05] are against collision-resistance, and no efficient attacks are known for finding a second pre-image for these functions. Therefore, these attacks are irrelevant against a one-way function that is used to commit the secrets and to hash the nodes of the Merkle tree.

4.3 Signature Time

Figure 4 shows the average signature time of FMTseq versus RSA signatures. The figure shows that even for the worst choice of the number of levels, L ($L = 10$), FMTseq-SHA-1 is roughly 4 times faster than 1024-bit RSA, and FMTseq-SHA-256 is more than 14 times faster than 2048-bit RSA. Moreover, other choices of L provide even better performance, e.g., for $L = 4$ we get a 10 times speedup over 1024-bit RSA and more than 35 times speedup over 2048-bit RSA.

Our benchmark measures the average signature time of FMTseq. The raw results are presented in Tables 3 and 4 in the Appendix. Each point in Figure 4 is an average of 128 runs, each consisting of 128 sequentially generated signatures over randomly chosen messages. Thus, we average over the variability caused by the amortization operations and by the different number of $'1'$-bits in the message digests. We found that for a given number of levels, L, the signature time is practically

Signature Scheme	Average Verification Time
FMTseq with SHA-1	76 uSec
FMTseq with SHA-256	114 uSec
RSA-1024	240 uSec
RSA-2048	600 uSec

Table 2: Average Verification Time for $H \leq 20$

invariant with the total tree height H. Since L must divide H, L can have only a few possible values for each H. Therefore, the value of each point on the graph represents a selection of a value of L for an appropriate tree height H.

Figure 4 shows that the average signature time is linear with L, thus, we see that the amortization time dominates the total signature time as the number of levels increases. Note that in the analysis of [JLMS03], two equally weighted operations were counted for each of $L-1$ *Desired* subtrees, meaning that hashing internal nodes and leaf computation were considered to have an equivalent time complexity. In contrast, in FMTseq the process of leaf computation (generating the secrets and commitments for a one-time signature), takes much longer. Therefore, the bound of [JLMS03] on the amortization time no longer represents the average case of FMTseq. Since there is an equal number of leaves and internal nodes, we argue that the average amortization time becomes the time for approximately $L-1$ leaf evaluations. The average leaf computation time T_{leaf} on our system is 0.15 milliseconds (using SHA-1) or 0.235 milliseconds (using SHA-256), and a close inspection of our data shows that for each value of L, the average signature time is almost exactly $L \cdot T_{leaf}$.

Table 2 shows the average signature verification time for FMTseq and RSA. The FMTseq verification time comprises of a one-time signature and authentication path verification. We found that the verification time is practically invariant to the tree parameters (see Tables 3 and 4 in the Appendix). Therefore, we provided only the average of all the results.

For a practical number of one million signatures (using $L = 4$), comparing FMTseq with 2048-bit RSA (using SHA-256 as a message digest) shows that the verification time is about 5 times faster, and signing time is approximately 35 times faster. Compared with 1024-bit RSA (using SHA-1 as a message digest), FMTseq verification is 3 times faster, and signing is about 10 times faster.

4.4 Signature Size

The signature size depends on the one-time signature parameters and on the length of the authentication path. The latter only slightly affects the overall size in practical constructions since it is logarithmic with the number of leaves. The one-time signature size is defined by the hash output length as shown in Figure 5.

For about one million signatures (H=20), we compared the signature size of FMTseq to that of RSA. Using SHA-256 as a message digest, the signature size is approximately 18 times larger than 2048-bit RSA, but is still reasonable for many applications: about 4.5 KBytes.

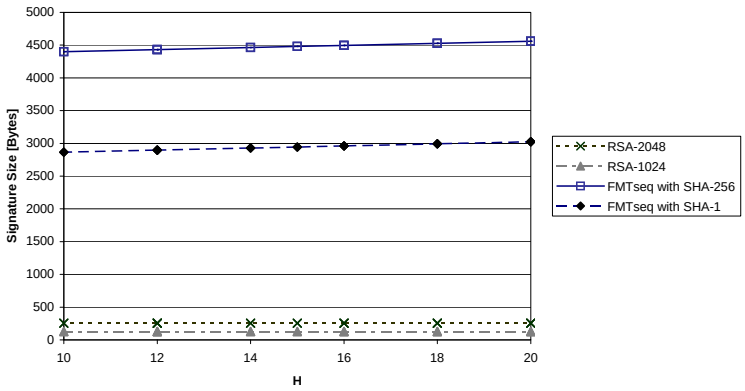


Figure 5: Signature size as a function of the tree height H in comparison with RSA signatures.

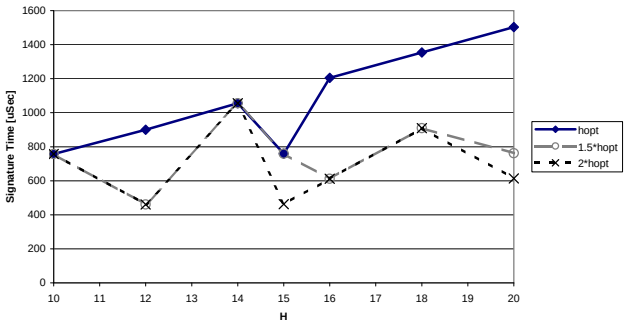


Figure 6: The improvement in performance that can be achieved, by allowing up to twice the optimal run-time space that was defined by [JLMS03].

4.5 A Memory versus Signature Time Trade-off

In [JLMS03] much effort is spent in search of run-time space optimality: reducing the memory requirements for running the fractal Merkle tree algorithm to a minimum. The authors show that minimal space is achieved with subtree height of $h_{opt} \approx \ln H$. However, h and L must be integers, and we argue that signature time is more important than run-time space in many applications.

Figure 6 demonstrates the improvements in total signature time when allowing a larger space allocation relative to the space that is required when $h = h_{opt}$. For example, when $H = 20$, $h_{opt} = 2$, (since 3 does not divide 20). If we allow up to twice the run-time space compared with h_{opt} (about 4.6 KBytes instead of 3KBytes), the signature time becomes more than twice as fast. The "dip" at $H = 15$ is due to the fact that $h_{opt} \approx 2$, but for $H = 15$, we must use $h = 3$, since h must divide H. For all other choices of H we have $h_{opt} = 2$.

4.6 Key Generation and Initialization

Key generation involves committing to all one-time signatures and initializing the fractal Merkle tree algorithm subtrees. From Figure 7 it can be seen that, as expected, the initialization time is

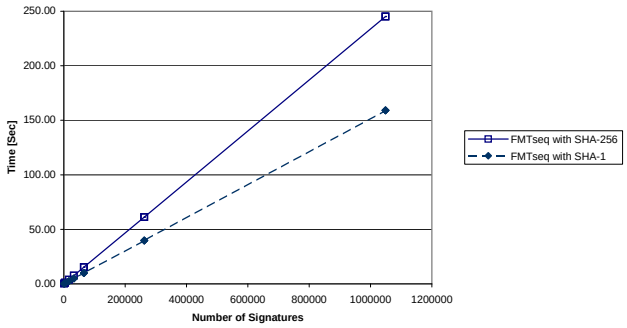


Figure 7: Initialization Time of FMTseq

linear with the number of signatures and depends on the hash function selection. For FMTseq with message digest of SHA-256, for about one million signatures, the initialization takes 245 seconds (about 4 minutes).

4.7 Selecting Number of Signatures

Constraining the parameters L and h to natural numbers is another aspect of FMTseq parameters selection. This constraint causes some constructions to be inefficient. For example, the results in Tables 3 and 4 (in the Appendix) show that seleting $H = 14$ and $L = 2$ is a bad choice. By choosing $H = 15$ and $L = 3$, one can achieve twice the number of signature with less than half the run-time space required.

5 Improvements

5.1 Using HMAC

The number of commitments in Merkle's signature depends on the output length of the hash function that is used to compute the message digest. The hash function is required to provide both collision resistance and second pre-image resistance to achieve unforgeability and non-repudiation. By reducing the length of the message digest, faster one-time signature with smaller signature size can be achieved. One way to shorten the message digest by half, is to remove the requirement to protect against birthday attacks. Birthday attacks can be used to generate two higher level attacks:

- Repudiation – A signer can find a pair of messages with the same hash, sign one message and later claim to sign the other.
- Adaptive chosen message attack – An adversary finds a pair of messages with the same hash, then asks the signer to sign one message, but actually gets him sign a different message.

However, protecting against these notions of security is not always essential. There are scenarios in which one signs a message that *he* has created to provide integrity and authentication. The verifier trusts the signer not to claim later that his signature was applied to a different message.

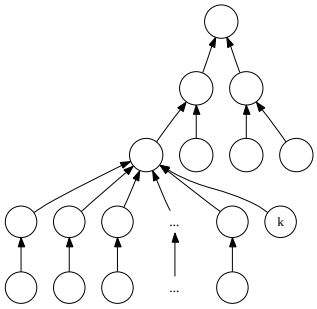


Figure 8: A TCR based FMTseq scheme. The HMAC key (marked 'k'), for each one-time signature, is committed by the leaf that represents it.

For example, users of an on-line financial news service may trust the data source but want to protect from adversaries who may alter the data or impersonate the service provider.

Bellare and Rogaway [BR97] (following Naor and Yung [NY89]) define *Target Collision-Resistance* (TCR) hash functions. A *family* of keyed hash functions, H_k, provides target collision-resistance if a hash function $h_k() \in H_k$, selected *after* choosing a message M, is resistant to finding a message M' such that $h_k(M') = h_k(M)$. Since the hash function key is selected independently, *after* selecting the message M, birthday attacks are prevented, despite the adversary having the ability to select M. As an outcome, the output of such TCR hash function can be smaller, for example 80 or 128 bits, instead of 160 or 256 bits of a standard collision-resistance hash function.

Assuming a signature scheme $(Gen, Sign, Verify)$, Bellare and Rogaway suggest that TCR functions can be used in signatures instead of standard hash functions in the following manner:

- To sign a given message M:
  - Generate a random key k. This key determines which hash function will be used.
  - Sign the message and the key using: $Sign(k||h_k(M))$
  - The key is published as part of the signature and is *not secret*.
- To verify the signature use $Verify(k||h_k(M))$.

Since there is no standard construction for TCR function we adopt the suggestion of [BR97] to use an *ad hoc* keyed hash function. We chose to use HMAC [BCK96] with SHA-1 as a keyed hash function. HMAC can be used with a truncated output, e.g., taking only 80 of the bits in its output. Hence, using 80-bit truncated HMAC-SHA1, instead of, for example, SHA-1, we can construct an FMTseq scheme that requires a smaller number of commitments, and thus, is more efficient in time and space, under the security assumptions that were presented.

In contrast to Bellare and Rogaway, we suggest adding the HMAC keys to the commitments of each one-time signature (see Figure 8). In this way, the HMAC keys are authenticated by the public root, and cannot be altered by an adversary[4].

To summarize, we construct the following HMAC-SHA1-80 based scheme of FMTseq:

[4]Rohatgi [Roh99] uses a similar approach for a hybrid scheme of one-time signatures and online/offline signatures.

- Key generation and initialization - same as FMTseq, except $t = 88$, since the truncated HMAC output is 80 bits and only 7 bits of checksum are required. The HMAC key is located in the 88th value, i.e., pk^i_{88} (See Figure 8).
- Sign - Similar to FMTseq with the additional output of the HMAC key as part of the signature.
- Verify - Similar to FMTseq except s'_{88} is taken as the HMAC key.

With this construction, each leaf in the authentication tree contains only 88 secrets and their commitments. Thus, the time to calculate the leaf becomes about half of the time that was required earlier. Since the total signature time is approximately linear with the leaf calculation time, the total signature is twice as fast[5].

The one-time signature size is also reduced since only about half of the secrets are required. For HMAC-SHA1-80 the one-time signature length is 1424 bytes. The total signature length depends on the authentication tree height, e.g., for $H = 20$ we get a signature size of 1.7 KBytes.

5.2 Beyond Using TCR for the Message Digest

Using TCR hash functions for committing one-time signatures (and not only for the message digest) can further reduce both the signature size and the space which is required for the hash tree traversal. Since only second pre-image resistance is required, and a birthday attack is inapplicable, a shorter output of the TCR hash functions is sufficient. However, two problems arise when looking for such a solution: (i) TCR functions are keyed hash functions, therefore a keying scheme should be provided; (ii) There is no standard choice for such TCR function.

In [Roh99], where such approach is presented, the TCR functions are keyed with a different key for each level of the tree. These keys are published together with the public root of the hash tree. The TCR function is taken to be SHA-1 Compress, which is a primitive that is used in SHA-1 algorithm. Notice that taking the TCR function to be HMAC function is not adequate, since each HMAC requires two hash operations and therefore the scheme will be less efficient.

If one accepts these suggestions, each node of the data structure (which includes the secrets, commitments and the hash tree) can be reduced from 128-bits to 80-bits. The result is a decrease of about 38% in both signature size and the space required to handle the hash tree.

5.3 Using the Online/Offline Approach

In [EGM89] it is suggested to divide the total signature time to online operations that are performed to sign a concrete message, and offline operations that can be performed until the next signature is needed. If we apply this approach to FMTseq, the online signature time excludes the amortization operations and thus it is independent of the tree parameters. This leads to a speedup factor of about L in the signature time. The verification time remains unchanged.

[5]Since only the inner hash is calculated over the whole message, the HMAC calculation time is practically equal to a standard unkeyed hash calculation time.

5.4 Forward Secure Signatures

A requirement for a "forward secure" signature defines a signature scheme that preserves the security of signatures, which have been issued earlier to the time the key is compromised. In [BM99], a forward secure signature scheme is presented, using the *key evolution* paradigm. Key evolution refers to iterative derivation of a key from the previous one, erasing the old key. At each period a new secret key is derived using a one-way function, and the former is destroyed. Thus, an exposure of K_i, the key of period i, does not compromise the keys K_j for periods $j < i$. The public key of the signature scheme does not change and stays fixed, so that the signature verification scheme is unchanged.

At first glance, applying forward security to FMTseq seems to be straight forward. Instead of using a fixed long-term secret key SK, that all the signatures' secrets are derived from, we can apply a key evolution scheme. For example, the secret key SK_i for each one-time signature will be derived as a hash of the preceding signature's key SK_{i-1}. After each signature the next key is derived from the current key, and the current key is erased.

However, using such a naive key evolution for FMTseq will lead to poor performance, since the amortization operations of the fractal Merkle tree traversal calculate the leaves in a non-sequential pattern. For example, after the first signature, the amortization of the uppermost *Desired* subtree, requires the calculation of leaves that are far "in the future" from the first one. Thus, the linear complexity of the naive key evolution scheme leads to performing many key derivation operations after each signature — which will destroy the fast signature time of FMTseq. Instead, one should use an efficient hierarchical key evolution algorithm (cf. [Dae97]). A key evolution scheme with a logarithmic complexity will let us maintain the high performance of FMTseq.

5.5 Unbounded Number of Signatures

The bound on the number of signatures is one of the main disadvantages of the scheme. Yet, if another FMTseq tree can be generated while the current tree is used, then the last signature of the current tree can be used to sign the public root of the next tree. After this public root update, the new tree can be used as usual. The public root update can be done using a protocol between the signer and the verifiers, or by adding the signed root to every signature.

Notice that the generation of the next tree can be done exactly like building the *Desired* subtrees, i.e., by taking two additional amortization calculations after each signature. In this way, a new tree will be ready just in time. A new trade-off is introduced in this way: one can select to work with small hash trees (which leads to faster signatures), at the cost of more frequent public root update.

6 Conclusions

In this work we defined FMTseq - a one-time signature scheme that combines Merkle's one-time signatures with the efficient hash tree traversal of [JLMS03]. Benchmarking FMTseq against RSA signatures, shows that a speedup of up to 35 times can be achieved if signature size of a few kilobytes is acceptable. The FMTseq scheme differs from the [JLMS03] suggestion for a signature scheme, in the method by which the leaves of the hash tree represent the one-time signatures. FMTseq

followed Merkle's suggestion [Mer89] for using the hash tree, and thus achieves many signatures with the same hash tree.

We demonstrated that when speed is the bottleneck factor, the requirement of [JLMS03] for run-time space optimality can be relaxed. A different selection of the parameters leads to a significant improvement in performance at the cost of a small increase in memory consumption.

We believe that when fast signatures are required, FMTseq or one of its suggested improvements can be a promising alternative to the public-key signatures like RSA.

References

[BC92] Jurjen N. Bos and David Chaum. Provably unforgeable signatures. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, pages 1–14. Springer, 1992.

[BC04] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In *Advances in Cryptology - CRYPTO 2004*, pages 290–305. Springer, 2004.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, pages 36–57. Springer, 2005.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO '96*, pages 1–15. Springer, 1996.

[BKN04] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for Merkle tree traversal. *Electronic Colloquium on Computational Complexity (ECCC)*, (049), 2004.

[BM94] Daniel Bleichenbacher and Ueli M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, pages 75–82. Springer, 1994.

[BM96a] Daniel Bleichenbacher and Ueli M. Maurer. On the efficiency of one-time digital signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96*, pages 145–158. Springer, 1996.

[BM96b] Daniel Bleichenbacher and Ueli M. Maurer. Optimal tree-based one-time digital signature schemes. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science*, pages 363–374, 1996.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - CRYPTO '99*, pages 431–448. Springer, 1999.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *Advances in Cryptology - CRYPTO '97*, pages 470–484. Springer, 1997.

[Col03] Dominic F. Coluccio. C++ implementation of a hash-based digital signature scheme using fractal Merkle tree representation. `http://cs1.cs.nyu.edu/~dfc218/hashsig.html`, 2003.

[Dae97] Joan Daemen. Management of secret keys: Dynamic key handling. In *State of the Art in Applied Cryptography*, LNCS 1528, pages 264–276. Springer, 1997.

[Dai04] Wei Dai. Crypto++ library 5.2.1. `http://www.eskimo.com/~weidai/cryptlib.html`, 2004.

[Dev05] Christophe Devine. Crypto source code, GNU public license. http://www.cr0.net:8040, 2001-2005.

[EGM89] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital schemes. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, pages 263–275. Springer, 1989.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, 2004.

[JLMS03] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal Merkle tree representation and traversal. In *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003*, pages 314–326. Springer, 2003.

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

[Len01] Arjen K. Lenstra. Unbelievable security. Matching AES security using public key systems. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, pages 67–86. Springer, 2001.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO '87*, pages 369–378. Springer, 1987.

[Mer89] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, pages 218–238. Springer, 1989.

[Nat95] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, April 1995. Supersedes FIPS PUB 180 1993 May 11.

[Nat02] National Institute of Standards and Technology. *FIPS PUB 180-2: Secure Hash Standard (SHS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, August 2002.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty-first annual ACM Symposium on Theory of Computing, Seattle, Washington, May 15–17, 1989*, pages 33–43, New York, NY, USA, 1989. ACM Press.

[Per01] Adrian Perrig. The BiBa one-time signature and broadcast authentication protocol. In *ACM Conference on Computer and Communications Security*, pages 28–37, 2001.

[PWX03] Josef Pieprzyk, Huaxiong Wang, and Chaoping Xing. Multiple-time signature schemes against adaptive chosen message attacks. In *Selected Areas in Cryptography, SAC 2003*, pages 88–100. Springer, 2003.

[Rab78] M. O. Rabin. Digitalized signatures. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 155–168. Academic Press, 1978.

[Roh99] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *ACM Conference on Computer and Communications Security*, pages 93–100, 1999.

[RR02] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Information Security and Privacy, 7th Australian Conference, ACISP 2002*, pages 144–153. Springer, 2002.

[SP05] Stefaan Seys and Bart Preneel. Power consumption evaluation of efficient digital signature schemes for low power devices. In *Proceedings of the 2005 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (IEEE WiMob 2005)*, pages 79–86. IEEE, 2005.

[Szy04a] Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology - EUROCRYPT 2004*, pages 541–554. Springer, 2004.

[Szy04b] Michael Szydlo. Recent improvements in the efficient use of Merkle trees: Additional options for the long term. RSA Laboratories: Technical Notes and Reports, 2004.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, pages 19–35. Springer, Advances in Cryptology - EUROCRYPT 2005.

A Appendix

A.1 Merkle's hash trees

Merkle [Mer89] presents a scheme for key management of one-time signatures. The scheme enables the verifier to authenticate a large number of one-time signatures commitments using low storage requirements. This is achieved by using the technique of a *hash tree*. A Merkle hash tree is a complete binary tree with an assignment of a value to each node, such that the value of the parent node is the output of a one-way hash function (OWHF) on its children nodes values, i.e.,

$$\phi(parent) = OWHF(\phi(left)|\phi(right))$$

The hash tree is used for key management of one-time signatures in the following way:

- Each leaf is a public commitment to a one-time signature.
- The root of the hash tree is given to the verifier in a secure, authenticated manner. The root serves as a short public commitment to all the one-time signatures.

- To authenticate the commitments of a single one-time signature, the signer provides the verifier with an *authentication path* – the values of all nodes that are siblings of nodes on the path between the leaf that represents the commitment and the root (See Figure 9).
- The verifier authenticates the one-time signature's commitment by iterative hashing of the commitment with the authentication path values.
- If the computed root value equals the public root commitment then the signature's commitment is authenticated.

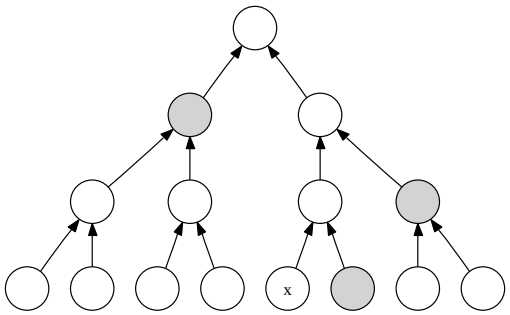


Figure 9: Authentication path. The shaded nodes are the authentication path of the marked leaf.

Notice that only the signer has to know the whole hash tree. The verifier is given only the root and the authentication path that is provided with the one-time signature. The verifier does not have to be aware of the fact that those values are the nodes of a binary tree or to store these values. The verifier only needs to perform the computation and compare the result with the public root.

For a large number of leaves, storing the whole tree at the signer side requires a large amount of space. Merkle [Mer89] shows that if the signer uses the leaves sequentially, he can compute the the authentication path with $O(\log n)$ time and $O(\log^2 n)$ space. This method is further improved in the work of [JLMS03] which is discussed in section 2.2.1.

Merkle [Mer87] presents a different approach that enables to authenticate an infinite number of one-time signature using a public root of a binary tree. However this method is impractical as each node in the tree holds commitments for three one-time signatures. As the number of signatures grows the amount of memory required to store the tree and the size of the signature become very large.

A.2 Raw Performance Data

Tables 3 and 4 summarize the measurements of FMTseq on a 1.7 GHz Pentium IV, running Windows XP.

H	L	Total # of Signatures	Initialization Time [Sec]	Signature Time [uSec]	Verification Time [uSec]	Run-time Memory [Bytes]	Signature Size [Bytes]
10	2	1024	0.16	311	73	2156	2864
10	5	1024	0.16	757	73	1072	2864
12	2	4096	0.62	312	74	4222	2896
12	3	4096	0.62	461	74	1788	2896
12	4	4096	0.62	609	74	1402	2896
12	6	4096	0.62	907	74	1398	2896
14	2	16384	2.5	313	75	8336	2928
14	7	16384	2.5	1055	75	1760	2928
15	3	32768	5	463	76	3378	2944
15	5	32768	5	759	76	1892	2944
16	2	65536	10	314	76	16546	2960
16	4	65536	10	612	76	2534	2960
16	8	65536	10	1205	76	2158	2960
18	3	262144	40.2	464	77	6504	2992
18	6	262144	40.2	909	77	2436	2992
18	9	262144	40.2	1354	78	2592	2992
20	4	1048576	161	614	79	4690	3024
20	5	1048576	161	762	79	3352	3024
20	10	1048576	161	1503	79	3062	3024

Table 3: Implementation results for FMTseq with SHA-1 as a message digest.

H	L	Total # of Signatures	Initialization Time [Sec]	Signature Time [uSec]	Verification Time [uSec]	Run-time Memory [Bytes]	Signature Size [Bytes]
10	2	1024	0.24	481	111	2156	4400
10	5	1024	0.24	1184	111	1072	4400
12	2	4096	0.96	481	110	4222	4432
12	3	4096	0.96	716	112	1788	4432
12	4	4096	0.96	950	112	1402	4432
12	6	4096	0.96	1420	115	1398	4432
14	2	16384	3.8	482	117	8336	4464
14	7	16384	3.8	1653	112	1760	4464
15	3	32768	7.7	717	115	3378	4480
15	5	32768	7.7	1185	114	1892	4480
16	2	65536	15	483	112	16546	4496
16	4	65536	15	952	116	2534	4496
16	8	65536	15	1888	116	2158	4496
18	3	262144	61	719	116	6504	4528
18	6	262144	61	1421	116	2436	4528
18	9	262144	61	2126	112	2592	4528
20	4	1048576	245	954	120	4690	4560
20	5	1048576	245	1188	113	3352	4560
20	10	1048576	245	2358	114	3062	4560

Table 4: Implementation results for FMTseq with SHA-256 as a message digest.