On the security and the efficiency of the Merkle signature scheme

Luis Carlos Coronado García
FB 20, Technische Universität Darmstadt
Hochschulstr. 10, D-64289, Darmstadt. Germany.
coronado@cdc.informatik.tu-darmstadt.de
http://www.cdc.informatik.tu-darmstadt.de/mitarbeiter/coronado.html

Abstract

This paper builds on the multi-time signature scheme proposed by Merkle. We prove that the original scheme is existentially unforgeable under adaptive chosen message attack. Moreover, we present an improved version which has three advantages: It is provably forward secure. The number of signatures that can be made with one private key is — in a practical sense — unlimited. Finally, the cost for key generation is kept low.

The theoretical exposition is complemented by experimental data about the efficiency of the improved Merkle signature scheme.

Keywords: Merkle Signature Scheme, Forward Security, Provably secure, Lamport-Diffie one-time Signature Scheme.

1 Introduction

Merkle presents in [Mer90] a multi-time signature scheme which employs a version of the Lamport-Diffie one-time signature scheme. Actually, the Merkle signature scheme transforms any one-time signature scheme in a multi-time one. Merkle did not give any proof of the security of his signature scheme.

The idea of the Merkle signature scheme is the following: Assume that we want to be able to make 2^N signatures. *Key generation* works as follows: Generate 2^N key pairs of the one-time signature scheme. Then, create a binary tree whose leaves are the hash values of the verification keys, and each parent is the hash value of the concatenation of its left and right children. The public value is the root of the tree. All the one-time key pairs taken together serve as the private key. The *signature* of message number k is the one-time signature with the k-th private one-time key, followed by the one-time verification key and N nodes from the tree which help to authenticate the verification key against the public value.

1.1 Our contribution

A proof of security. We show that assuming the existence of cryptographically secure hash functions and cryptographically secure one-time signature schemes, the Merkle signature scheme is not existentially forgeable under adaptive chosen message attack. To do this, we prove first that any alteration of any number of nodes of a Merkle tree that keeps the root intact yields an explicit collision for the underlying hash function. Then, we show that any existential forgery of signatures in the Merkle signature scheme leads to either an existential forgery of signatures for the underlying one-time signature scheme or a collision for the underlying hash function.

A forward secure version of the Merkle Signature Scheme. Roughly speaking, a forward secure signature scheme is one for which the validity of a public key is divided into periods, and the corresponding private key "evolves" after each period in such a way that if the private key is compromised by an adversary in some period, the adversary cannot succeed in forging a signature for a previous period. We modify the original Merkle scheme in order to transform it into a forward secure one. Note that if all the one-time signing keys are stored and each one of them is deleted after its use, the resulting Merkle signature scheme is forward secure, where each period consist of exactly one signature. In this case, the private key of the Merkle signature scheme is as big as the stored one-time signing keys. In our version, the size of the private key is reduced by employing a pseudorandom bit generator and a one-time signature scheme with deterministic key-generation. Bellare and Yee have shown in [BY03] how to construct a forward-secure pseudorandom bit generator from a cryptographically secure pseudorandom bit generator. The use of that bit generator enables us to prove that our new version of the Merkle signature scheme becomes forward secure.

Key generation process split through the signature process. In the Merkle signature scheme, the number of possible signatures is 2^N, where N is the depth of the Merkle tree. The bigger the parameter N is, the slower the key generation process becomes: during key generation $2^{N+1} - 1$ hash values and 2^N one-time key pairs have to be computed.

We present a version where part of the key generation takes place during the use of the signature scheme. This permits the use of sufficiently large parameters to allow for a practically unlimited number of signatures while keeping the cost of the initial key generation process low. The basic idea is to use one ("top") Merkle tree to authenticate the roots of a series of other, "bottom" trees. Only one of the bottom trees is kept at a time. During the use of one bottom tree, the next one is generated, namely two nodes at a time per signature.

1.2 Previous work

Merkle presents in [Mer90] his multi-time signature scheme. In that work, he improves the Lamport-Diffie one-time signature scheme [DH76], whose security hinges on that of the used hash function, and introduces his multi-time extension based on a binary tree, which we will call Merkle tree henceforth. No formal proof of the security of the scheme has been given, although its properties are widely known. See e.g. Micali [Mic00] who mentions, without proof, that the crucial property of Merkle trees is the difficulty of changing any node in the tree while keeping the root unaltered, provided that the used hash function is collision resistant.

1.3 Outline of the paper

We give a description of the Merkle signature scheme and discuss its security in Section 2. Then, in Section 3, we present our improved version and discuss its security. Finally, in section 4, we give some measurements of the performance of the scheme and extrapolate the collected data to obtain a full picture of the efficiency of our improved version.

For the convenience of the reader we attach two Appendices. The first contains the proofs of some auxiliary statements from Section 2. In the second, we recall the Diffie-Lamport one-time signature scheme.

1.4 Notation

A *signature scheme* is given as a triple (Gen, Sig, Ver) of probabilistic polynomial-time algorithms. A *one-time signature scheme* is, roughly speaking, a signature scheme which is guaranteed to be secure as long as each private key is not used more than once. In our context private keys of signature schemes will also be called *signing keys*, the public ones *verifying keys*.

Informally, we say that a scheme has (t, ϵ)-*property* $\mathcal{P}$ if no Adversary $\mathcal{A}$, modeled as a probabilistic algorithm which runs within time t, succeeds with probability larger than ϵ in breaking property $\mathcal{P}$. Here, we adopt the convention that the time-complexity is the total worst-case execution time of $\mathcal{A}$ plus the size of its code, all measured in some fixed model of computation.

The following notion will be needed for the description of the Merkle signature scheme. Assume we are given a rooted full binary tree of depth N. Let us label the leaves of the tree from 0 to $2^N - 1$. We call

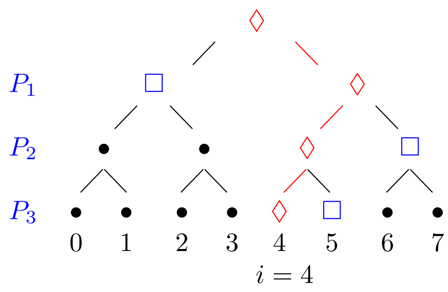

Figure 1: Example of an i-labeled sibling path for $N = 3$ which is depicted in $\square$. The path of the leaf i is pictured in $\Diamond$.

$(P_N, \ldots, P_1)$ the *i-labeled sibling path for the leaf* L if L is the leaf of the tree with label i and, for $1 \leq j \leq N$, P_j is the sibling of the node of the path from the root to L at depth j. The term is illustrated in Figure 1.

2 Efficiency and security of the Merkle signature scheme

In this Section we give a formal description of the Merkle signature scheme. Let $1Sign = (1Gen, 1Sig, 1Ver)$ be a one-time signature scheme, and let $H : \{0,1\}^m \to \{0,1\}^s$ be a hash function.
Key generator algorithm Gen. On input $N \in \mathbb{N}$ and security parameter s, Gen calls 2^N times $1Gen(s)$ in order to obtain the key pairs (X_i, Y_i) for $0 \leq i < 2^N$. Then it computes recursively the nodes of the tree: $K_{N,i} = H(Y_i)$ and $K_{j,k_j} = H(K_{j+1,2k_j}, K_{j+1,2k_j+1})$ for $0 \leq i < 2^N$, $0 \leq j < N$ and $0 \leq k_j < 2^j$. Finally, it sets $R = K_{0,0}$.

The public key output by Gen is (N, R). The private key consists of a concatenation of the key pairs (X_i, Y_i). (In case the one-time verifying keys Y_i can be computed from the X_i, it is sufficient to store only the one-time signing keys X_i.) The user must keep a counter which contains the number of previously created signatures. In the beginning, the counter is set to zero.
Signature algorithm Sig. Let i be the counter. On input of a message M and the secret key, Sig calls $1Sig(M, X_i)$ in order to obtain the one-time signature τ and computes $(P_N, \ldots, P_1)$, the i-labeled sibling path for the leaf $H(Y_i)$.

Finally, Sig outputs the signature $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$, and then increments the counter i by one.
Verification algorithm Ver. On input a message M, a signature $\sigma' = (i, \tau', Y', P'_N, \ldots, P'_1)$ and a public key (N, R), Ver accepts the signature σ' unless $1Ver(M, \tau', Y') = false$ or $W_0 \neq R$, where $W_N = H(Y')$ and $W_{j-1} := H(l_j, r_j)$ for $(0 < j \leq N)$, where $l_j := W_j$, $r_j := P_j$ in case that $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even or $l_j := P_j$, $r_j := W_j$ otherwise.

2.1 Security

Goldwasser et al. [GMR88] introduced the concepts of existentially forgeable and adaptive chosen message attack. We adopt concepts from [BMS03, EGM96] and [RS04] for existentially unforgeable under adaptive chosen message attack, one-time signature schemes and collision-resistant functions respectively. The main result of this Section is the Proposition 1. The lemma 1 proofs the important property of the Merkle trees when an i-labeled sibling path becomes an i-labeled authentication path in a Merkle tree.

Here we give the notion of an authentication path. Let $H : \{0,1\}^m \to \{0,1\}^s$ be a function, where $m = *$ or $m \geq 2s$, let $N \in \mathbb{N}$ and let $L, P_N, \ldots, P_1, R \in \{0,1\}^s$. Set $0 \leq i < 2^N$. We say that $(P_N, \ldots, P_1)$ is an *i-labeled authentication path of depth N for the leaf L with respect to the root R* if $R = W_0$, where $W_N := L$ and $W_{j-1} := H(l_j, r_j)$ for $(0 < j \leq N)$ and $l_j := W_j$, $r_j := P_j$ in case that $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even or $l_j := P_j$,

$r_j := W_j$ in the other case. Here $H(x,y)$ denote $H(x\|y)$ if $m = *$ and denote $H(x\|y\|0^{m-2s})$ in the other case, where $\|$ is the concatenation of strings.

Lemma 1 *Let H be a hash function, which image is in $\{0,1\}^s$. Let $R \in \{0,1\}^s$ and let $N \in \mathbb{N}$. Set $0 \le i < 2^N$. If $A = (P_N, P_{N-1}, \ldots, P_1)$ is an i-labeled authentication path of depth N for a leaf L and $B = (P'_N, P'_{N-1}, \ldots, P'_1)$ is an i-labeled authentication path of depth N for a leaf L' both of them with respect to the root R, then either $A \neq B$ or $L \neq L'$ implies an explicit collision for H.*

Proof

We define $W_N := L$ and $W_{j-1} := H(l_j, r_j)$ for $0 < j \le N$, where $l_j = W_j$ and $r_j = P_j$ in case of $\lfloor \frac{i}{2^{N-j}} \rfloor$ is even and $l_j = P_j$ and $r_j = W_j$ in the other case. In an analogous way we define W'_j for $0 \le j \le N$.

If $A \neq B$, set $k := \min_{1 \le j \le N}\{P_j \neq P'_j\}$. We know that $W_0 = W'_0 = R$, so if there exists $0 \le j_0 < k$ such that $W_{j_0} \neq W'_{j_0}$ and $W_j = W'_j$ for $0 \le j < j_0$ then $H(l,r) = W_{j_0-1} = W'_{j_0-1} = H(l', r')$, where $l = W_{j_0}, r = P_{j_0}, l' = W'_{j_0}, r' = P'_{j_0}$ in case of $\lfloor \frac{i}{2^{N-j_0}} \rfloor$ is even and $l = P_{j_0}, r = W_{j_0}, l' = P'_{j_0}, r' = W'_{j_0}$ in the other case. And then $(l,r) \neq (l',r')$ but $H(l,r) = H(l',r')$. Now we have that $W_{k-1} = W'_{k-1}$ and then $H(l,r) = W_{k-1} = W'_{k-1} = H(l',r')$, where $l = W_k, r = P_k, l' = W'_k, r' = P'_k$ in case of $\lfloor \frac{i}{2^{N-k}} \rfloor$ is even and $l = P_k, r = W_k, l' = P'_k, r' = W'_k$ in the other case, what implies that $(l,r) \neq (l',r')$ but $H(l,r) = H(l',r')$.

If $A = B$ and $L \neq L'$, then we can assume that $W_{N-1} = W_{N'-1}$ (because of $W_0 = W'_0$ if there exists $1 \le j_0 < N$ such that $W_j = W'_j$ for $0 \le j < j_0$ and $W_{j_0} \neq W'_{j_0}$ then we proceed in a similar way as in the case $A \neq B$ in order to find a collision for H) and so $H(l,r) = W_{N-1} = W_{N'-1} = H(l',r')$, where $l := L, r := P_N, l' := L', r' := P'_N$ in case that i is even or $l := P_N, r := L, l' := P'_N, r' := L'$ in the other case, but $(l,r) \neq (l',r')$. □

Proposition 1 ***Security of the Merkle signature scheme.*** *Let H be a (t_{cr}, ϵ_{cr}) collision resistant hash function and let $1Sign$ be a (t_{1s}, ϵ_{1s}) one-time signature scheme, such that $\epsilon_{cr} \le \frac{1}{2}$, $t_{cr} \ge t + (N+1)t_h + 2^N t_s + t_g$ and $t_{1s} \ge t + 2^N t_s + t_g$ for some t and some $N < \lfloor -\log_2 \epsilon_{1s} \rfloor$, where t_h is the time needed to compute a hash value, t_s is the time needed to compute a one-time signature and t_g is the time needed to generate a key pair for the Merkle signature scheme. Let $\epsilon = 2\max\{\epsilon_{cr}, 2^N \epsilon_{1s}\}$, then the Merkle signature scheme $Sign$ is $(t, \epsilon, 2^N)$ existentially unforgeable under adaptive chosen message attack.*

Proof

Suppose that the Merkle signature scheme can be existentially broken via a 2^N message attack in time t and probability ϵ. Then we prove that at least one of the following holds:

- A collision can be found for the underlying hash function with probability at least ϵ_{cr} and within time t_{cr}.
- The underlying one-time signature scheme can be existentially broken with probability at least ϵ_{1s} and within time t_{1s}.

Suppose that a forger $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ exists such that

$$Pr[Ver(M, \sigma', PK) = 1 \ : \ (SK, PK) \leftarrow Gen(s,N); T \leftarrow \mathcal{F}_1^{Sig(\cdot, SK)}(s); (M, \sigma') \leftarrow \mathcal{F}_2(T)] \ge \epsilon$$

in time t.

We construct at the same time two algorithms, $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ which try to forge a one-time signature and $\mathcal{B}$ which tries to find a collision.

$\mathcal{B}$ begins getting $(Pr, Pu) \leftarrow 1Gen(s)$ a pair of one-time signature keys. $\mathcal{A}_1$ chooses $0 \le i_0 < 2^N$ at random and sets $Y_{i_0} := Pu$. Now for $0 \le i < 2^N$ with $i \neq i_0$, $\mathcal{A}_1$ obtains $(X_i, Y_i) \leftarrow 1Gen(s)$ and completes the generation process for the Merkle signature scheme. Let R be the Merkle tree's root. This resulting generated Merkle key is taken by $\mathcal{B}$. $\mathcal{A}_1$ calls $\mathcal{F}_1$ and uses $1Sig(\cdot, Pr)$ at most once as an oracle, just in case $\mathcal{F}_1$ needs to compute the (i_0+1)-th signature. Note that $\mathcal{B}$ knows Pr, so it does not use any oracle to sign with that key. $\mathcal{A}_1$ outputs T, where T is the output of $\mathcal{F}_1$. $\mathcal{B}$ retains the number of messages k and the messages M_j for $0 \le j < k$ used by $\mathcal{F}_1$. On input T, $\mathcal{A}_2$ calls $\mathcal{F}_2$ with T as input and obtains (M, σ'), a forged signature σ' of a message M. Let $\sigma' = (i, \tau', Y', P'_N, \ldots, P'_1)$ be the forged signature for the message M, $\mathcal{A}_2$ outputs (τ', M). At this point, algorithm $\mathcal{A}$ has concluded its running time, while $\mathcal{B}$ continues. Let $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$ be an authentic signature for the message M_i, where $M_i = M$ if $i = k$ and $M_i \ne M$ if $i < k$. $\mathcal{B}$ sets $A = (P'_N, \ldots, P'_1)$ and $B = (P_N, \ldots, P_1)$. By hypothesis A is an i-labeled authentication path for the leaf $H(Y')$ and B is an i-labeled authentication path for the leaf $H(Y_i)$ both of them with respect to the root R. If either $A \ne B$ or $Y' \ne Y_i$ then $\mathcal{B}$ outputs either the collision which is found by Lemma 1 or (Y', Y_i) (note that $Y' \ne Y_i$ and $H(Y') = H(Y_i)$ implies a collision). In other case, $\mathcal{B}$ outputs $(\overline{1}, \overline{0})$.

Let $\wp$ be the probability that (Y', A) obtained by $\mathcal{F}$ and (Y, B) obtained by the signature algorithm are different. If $\wp \ge \frac{1}{2}$ then $\mathcal{B}$ finds a collision for H within time t_{cr} with probability at least $\frac{\epsilon}{2} \ge \epsilon_{cr}$. Else with probability $\frac{1}{2^N}$, the forged signature output by $\mathcal{F}$ satisfies that $i = i_0$. Hence, the attack by $\mathcal{A}$ on the one-time signature scheme succeeds with probability at least $\frac{1}{2}\frac{\epsilon}{2^N} \ge \epsilon_{1s}$ within time t_{1s}. □

2.2 Efficiency

Let l_{sec} be the maximal size of a one-time signing key, let l_{ver} be the maximal size of the corresponding one-time verification key, let l_{sig} be the maximal size of the corresponding one-time signature and let l_{int} be the number of bits needed to store the counter. Merkle suggested in his work an improved version of the Lamport-Diffie one-time signature. In Appendix B that one-time signature scheme is described. Using the Lamport-Diffie one-time signature scheme improved by Merkle we have $l_{sec} = l_{ver} = s(s + 1 + \lfloor \log_2 s \rfloor)$, $l_{sig} = s^2$, and for $N = 14$, $s = 160$ and $l_{int} = 32$ we have that the private key size $\approx 2^{25.71}$ bytes $\approx$ 52MB, and the public key size is 24 bytes and the signature maximal size $\approx$ 3.48KB.

3 Improvements to the Merkle signature scheme

We give a version of the Merkle signature scheme, which is forward secure and the number of possible signatures is practically unlimited. We proceed in two steps. First we give the forward secure version and then the one with respect to the number of signatures.

3.1 First improvement to the Merkle Siganture Scheme

We can save space for the private key in the Merkle signature scheme if we can compute a specific one-time-signature key pair any time we need it, that is why we need a deterministic key generator and, therefore, we must make some changes at the one-time signature scheme.

Informally, by a *signature scheme with deterministic key generation* we mean a triple of algorithms (Gen, Sig, Ver), where Gen is deterministic with an input parameter *seed* such that if *seed* is selected randomly, then Gen seems to be probabilistic. In an analogous way we have the concept for a *one-time signature scheme with deterministic key generation*.

Now let us consider a pseudorandom bit generator PRG. We have adopted the definitions given in [BY03]. Roughly speaking, a pseudorandom bit generator $g : \{0,1\}^l \to \{0,1\}^{l+l'}$ is *cryptographic secure* if $g(\mathcal{U}_l)$ and $\mathcal{U}_{l+l'}$ are computationally indistinguishable, where $\mathcal{U}_\alpha$ denote the uniform distribution on $\{0,1\}^\alpha$.

We assume that there exists $PRG : \{0,1\}^s \to \{0,1\}^u$ which is a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator and $u \ge 2s$.

Improved key generator algorithm Gen. On input s and $N \in \mathbb{N}$, Gen chooses $g_{-1} \in_R \{0,1\}^s$ and obtains the key pair $(X_i, Y_i) \leftarrow 1Gen(s, seed_i)$ for each $0 \le i < 2^N$, where $(g_i, seed_i) \leftarrow PRG(g_{i-1})$. As before, Gen computes the corresponding Merkle tree, i.e. K_{j,i_j} for $0 \le j \le N$ and $0 \le i_j < 2^j$. Gen outputs the tree's depth N and the tree's root R as the public key, and g_{-1} as the private key. The signer must keep a counter in order to sign sequentially. At this point such counter must be initialized to zero.

Improved signature algorithm Sig. Let i be the counter. On input the message M and the private key g_{i-1}. Sig obtains $(g_i, seed_i) \leftarrow PRG(g_{i-1})$ in oder to call $(X_i, Y_i) \leftarrow 1Gen(s, seed_i)$, obtains $\tau \leftarrow 1Sig(M, X_i)$ and computes $(P_N, \ldots, P_1)$, the i-labeled sibling path for the leaf $H(Y_i)$, then Sig outputs the

signature $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$ and, finally, Sig deletes g_{i-1} and $seed_i$, and after that increments the counter i by one.
Verification algorithm remains the same.

3.1.1 Security

Bellare and Yee define in [BY03] the concept of forward-secure pseudorandom bit generator and give its construction from any pseudorandom bit generator. This tool permits that the new version of the Merkle signature scheme remains not only secure, but also forward secure as we mention below.

Proposition 2 *Let H be a (t_{cr}, ϵ_{cr}) collision resistant hash function, let $1Sign$ be a (t_{1s}, ϵ_{1s}) one-time signature scheme and let PRG be a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator, such that $2^{N+2}\epsilon_{prg} \leq 1$, $\epsilon_{cr} \leq \frac{1}{4}$, $t_{cr} \geq t + (N+1)t_h + 2^N t_s + t_g$, $t_{1s} \geq t + 2^N t_s + t_g$ and $t_{prg} = t + t_g + 2^N t_s + t_v + O(2^{N+1}s)$ for some t and some $N < \lfloor -\log_2 \frac{\epsilon_{1s}}{2} \rfloor$, where t_h is the time needed to compute a hash value, t_s is the time needed to compute a one-time signature, t_v is the time needed to verify a Merkle signature and t_g is the time needed to generate a key pair for the Merkle signature scheme. Let $\epsilon = \max\{4\epsilon_{cr}, 2^{N+2}\epsilon_{1s}, 2^{N+2}\epsilon_{prg}\}$, then the improved Merkle signature scheme Sign is $(t, 2^N, \epsilon)$ existentially unforgeable under adaptive chosen message attack.*

3.1.2 Forward security

In the rump session of Eurocrypt 97, Anderson introduced the idea of forward secure scheme [And02]. The idea is that the use of a public key is divided in periods. During that use, the public key remains the same whereas the private key changes in each period. If the private key is compromised in a certain period, the signatures which were made in previous periods remain secure, i. e. they cannot be forged.

Bellare and Miner [BM99] formalize these ideas and present a forward secure scheme based on the hardness of factoring.

Krawczyk [Kra00] suggests the idea of using Merkle certification tree where the leaves are the period certificate information which includes the period's public key and period's number but not a signature in order to make a conventional signature scheme a forward secure one. Maklin et al. [MMM02] construct a forward-secure signature scheme which is also based on conventional signature schemes.

Bellare and Miner give in [BM99] the definition of the key-evolving signature scheme which consists of four processes (Gen, Upd, Sig, Ver), where Gen is the key generation process, Upd is the update process, Sig is the signature process and Ver is the verification process. The Gen receives as input the security parameter, the number of periods and maybe other information. The Upd process is called in order to update the private key.

Roughly speaking a key-evolving signature scheme is forward secure if any adversary who obtains the private key in a certain period cannot succeed in forging a signature of any of the previous periods. In this case, the adversary calls the Upd process any time he wants.

In our case, the update process is intrinsic to the signature process and cannot be called at any time. Each period consists of a number of signatures instead of time.

We adopt the convention that a secret key in a period T is the null string, where T is the number of periods for the signature scheme.

Remark 1 *With the notation of the description of the original Merkle signature scheme given in Section 2, let us modify the scheme as follows:*

Signature algorithm *Sig. On input a message M, Sig uses (X_i, Y_i) in order to obtain $\tau \leftarrow 1Sig(M, X_i)$. It computes the i-labeled sibling path for the leaf $H(Y_i)$, outputs $(i, \tau, Y_i, P_N, \ldots, P_1)$, deletes X_i and then increments i by one.*

Key generation algorithm and the verification algorithm *of the original Merkle signature scheme remain the same.*

The private key at the period i is either $SK_i = \{X_j\}_{i \leq j < 2^N}$ or $SK_i = \{(X_j, Y_j)\}_{i \leq j < 2^N}$.

With the hypothesis of Proposition 1, this version of the original Merkle signature scheme is $(t, \epsilon, 2^N)$ forward secure, where the number of periods is 2^N and each period consists of only one signature.

Proposition 3 *With the hypothesis of the proposition 2. The improved version of the Merkle signature scheme is forward secure, with each period consisting of only one signature.*

3.1.3 Efficiency

We will consider the improvement due to Szydlo [Szy03] which computes sequential tree leaves and authentication path data in time N and space less than $3N$, where the units of computation are hash function evaluations or leaf value computations, and the units of space are the number of node values stored.

The signer must keep not only the counter i in order to sign sequentially, but also up to $3N$ node values needed for Szydlo's method and, additionally, some g_j in order to reduce the computation of the key pair (X_i, Y_i). Let p be an integer such that $0 \leq p \leq N$. Set $q = N - p$. If we store $\{g_{k2^p-1}\}_{0 \leq k < 2^q}$, then we must call PRG no more than 2^p times to compute $(g_i, seed_i)$ for $0 \leq i < 2^N$.

In order to compute a leaf j from $\{g_{i-1}\} \cup \{g_{l2^p-1}\}_{\lfloor \frac{i}{2^p} \rfloor < l < 2^q}$, where $i \leq j < 2^N$, the algorithm Sig could use the algorithm represented in Figure 7.

The Signature algorithm Sig must use the Szydlo technique in order to compute $(P_N, \ldots, P_1)$ the i-labeled sibling path for the leaf $H(Y_i)$.

This time the maximal size of the private key is $(3N + 2^{q+1})s$, and the size of the public key is $s + l_{int}$ and the maximal size of the signature is $sN + l_{sig} + l_{int}$, where l_{sig} is the maximal size of a signature of the one-time signature scheme. For instance, using the Lamport-Diffie one-time signature scheme improved by Merkle we have $l_{sig} = s^2$, and for $N = 14$, $p = \frac{N}{2}$, $s = 160$ we have that the maximal size of the private key is 5960 bytes $\approx$ 5.8KB, and the size of the public key is 24 bytes and the maximal size of the signature $\approx$ 3.4KB.

3.2 Second improvement to the Merkle signature scheme

Let $MSign = (MGen, MSig, MVer)$ be the Merkle signature scheme as described in the Subsection 3.1. We describe a new version of the Merkle signature scheme.

Improved key generation algorithm Gen. On input the security parameter s and the parameter N, for 2^{2N} possible signatures, Gen calls twice $MGen$ to compute $(\chi_{-1}, (N, R))$ and $(\chi_{0,-1}, (N, R_0))$ and then computes $\zeta_0 = MSig(R_0, \chi_{-1})$ and outputs (N, R) as the public key and $(\chi_{0,-1}, \chi_0)$ as the private key, where $(\chi_0, seed_0) \leftarrow PRG(\chi_{-1})$.

The signer must keep two counters. One of them counts the number of generated signatures modulo 2^N, which at this point must be initialized to zero, and the other counts the number of signatures created by the first generated Merkle key pair, which at this point must be initialized to one. The signer must also keep (ζ_0, R_0).

Improved signature algorithm Sig. Let i and j be the counters described in the previous improved key generation algorithm. On input a message M and the secret key $(\chi_{j-1,i-1}, \chi_{j-1})$, Sig computes $\tau_i \leftarrow MSig(M, \chi_{j-1,i-1})$ and sets $\sigma = (\tau_i, R_{j-1}, \zeta_{j-1})$, then computes $(\chi_{j-1,i}, seed_{j-1,i}) \leftarrow PRG(\chi_{j-1,i-1})$ and, after that, increments i by one. If at this point $i \cong 0 \mod 2^N$, Sig computes $(\chi_j, seed_j) \leftarrow PRG(\chi_{j-1})$ and calls $MGen$ to obtain $(\chi_{j,-1}, (N, R_j))$ then computes $\zeta_j \leftarrow MSig(R_j, \chi_j)$. Sig sets $i \leftarrow 0$ and increments j by one. Finally, Sig outputs the signature σ.

The signer must keep (ζ_{j-1}, R_{j-1}) for further signatures.

Improved verification algorithm Ver. On input a signature $\sigma = (\tau, R', \zeta)$ and a message M, Ver accepts the signature if both $MVer(M, \tau, (N, R'))$ and $MVer(R', \zeta, (N, R))$ are $true$, and rejects it otherwise.

3.2.1 Security

This improved version of the Merkle signature scheme uses random and independent instances of the Merkle signature scheme. This time, there is a base instance, whose public key is used as the public key of the scheme and the public key of one of the other instances is signed by the base instance. This improved version is as secure as the Merkle signature scheme.

Proposition 4 *Let $MSign$ be a $(t_{ms}, 2^N, \epsilon_{ms})$ Merkle signature scheme, such that $t_{ms} \geq t+2^N(2^N t_{sig}+t_{gen})$ and $2^{N+1}\epsilon_{ms} \leq 1$ for some t, where t_{sig} is the time needed for $MSig$ to sign a message and t_{gen} is the time needed for $MGen$ to generate a key pair. Then the improved version of the Merkle signature scheme $Sign$ is $(t, 2^{2N}, \epsilon)$ existentially unforgeable under adaptive chosen message attack, where $\epsilon = 2^{N+1}\epsilon_{ms}$.*

3.2.2 Efficiency

The key generation algorithm and the verification algorithm of this version take twice the time for the Merkle signature scheme for the same parameter N, but this time the number of possible signatures is 2^{2N} instead of only 2^N.

The signature algorithm must compute a key pair of the Merkle signature scheme after 2^N created signatures. This can be improved if it computes two nodes of the Merkle tree after each signature process. We describe the algorithm bellow.

The Szydlo method computes upto N leaves of the Merkle tree which are spread. Therefore we must have an efficient way to compute such leaves. If $0 \leq p \leq N$ and $q = N - p$, then we can store 2^q PRG's outputs in order to compute each Merkle tree's leaf with upto 2^p calls to the PRG. In the algorithm represented in Figure 8 we set $p = \left\lfloor \frac{N}{2} \right\rfloor$. The auxiliary values $Auth_k$ are needed in the Szydlo method, that is why they are also computed in the algorithm represented in Figure 8.

The Szydlo Method `Szydlo_method` will be used in the algorithm represented in Figure 9 in order to compute the i-th sibling path of a Merkle tree's leaf needed to compute a Merkle signature in an efficient way. This method uses as input the values $Auth_k$, which are computed in the algorithm represented in Figure 8. The auxiliary $Keep_k$ and $Need_k$ are updated during each use of `Szydlo_method`, for $0 \leq k < N$. The Szydlo Method outputs the sibling paths sequentially. The values $Keep_k$ and $Need_k$ can be initialized to the null string for $i = 0$.

In the Szydlo method a leaf must be computed calling `Leaf-Calc`. In Figure 7 an algorithm is represented, which obtains the value of leaf $leaf$ from $Pr = \{\chi_{i-1}\} \cup \{\chi_{l2^p-1}\}_{\left\lfloor \frac{i}{2^N} \right\rfloor < l < 2^q}$

The Merkle verification algorithm is represented in Figure 10. The improved versions for the key generator, the signature and the verification algorithms are represented in Figures 2, 3 and 4, respectively.

4 Experimental Results

Our experiments are estimates of the size of the Merkle keys and the Merkle signature and they are also estimates of the time needed for the Key Generation, Signature and Verification algorithms. These experiments were made on a SUN 4 ultra SPARC Sun-Blade-100, Sun OS 5.8, at 500MHz and we have used RIPEMD160 as hash function, our improved version of the Lamport-Diffie one-time signature scheme, which is described in Section B, and the number of possible signatures is 2^N. The cryptographic library used in the computations of RIPEMD160 values is OpenSSL [Ope] version 0.9.7d. and the used pseudorandom bit generator is ISAAC [ISA].

In Table 1 is shown the time consumption for signing and verifying with RSA. The time consumption depends on the length of the RSA keys. The security of the RSA signature scheme increases with the length of the key. In Table 2 the comparison between our first version of the Merkle signature scheme (1MSS) and our second one (2MSS) is shown. The number of possible signatures is 2^N and the public key size is 24 bytes.

Here we compare the efficiency for signing and verifying. In this case, the improved version of the Merkle signature scheme is more efficient than RSA with a 2048-bit key for verifying with the parameter $N \leq 40$ and is almost as efficient as RSA with 2048-bit key for signing with parameter $N \leq 16$. The number of possible signatures is 2^N.

Now we compare the security of these signature schemes. In our experiment we have measured the time for calculating a RIPEMD160 hash value of input size of 320 bits and we have an estimation of $(7.8 \times 10^{-5} sec)(500MHz) = 3.9 \times 10^4$ instructions for a RIPEMD160 value computation. Assuming that there exists a hash function H such that the computation of a hash value were 3.9×10^4 instructions, the best

RSA Signature Scheme		
key size (bits)	Signature	Verification
	milliseconds	
512	28.01	1.08
1024	55.52	2.09
2048	224.55	6.50

Table 1: Timing and Size for RSA.

	Key Generation Time		Private key size Kb		Signature Time		Signature size Kb		Verification Time in ms	
N	1Mss	2Mss	1Mss	2Mss	1Mss	2Mss	1Mss	2Mss	1Mss	2Mss
8	1.72 sec	0.222 sec	804 bytes	0.633	217 ms	105 ms	1.96	3.78	1.53	2.4
10	6.89 sec	0.438 sec	1.21 Kb	0.906	337 ms	135 ms	2	3.82	1.7	2.56
12	27.6 sec	0.869 sec	1.96 Kb	1.02	564 ms	158 ms	2.04	3.86	1.86	2.73
14	1.84 min	1.73 sec	3.32 Kb	1.45	1.03 sec	204 ms	2.07	3.89	2.03	2.89
16	7.35 min	3.45 sec	5.94 Kb	1.57	2.04 sec	230 ms	2.11	3.93	2.19	3.06
18	29.4 min	6.9 sec	11.1 Kb	2.31	4.23 sec	317 ms	2.15	3.97	2.36	3.23
20	1.96 hrs	13.8 sec	21.2 Kb	2.43	9 sec	350 ms	2.19	4.01	2.53	3.39
22	7.84 hrs	27.6 sec	41.3 Kb	3.8	19.4 sec	531 ms	2.23	4.05	2.69	3.56
24	1.31 days	55.1 sec	81.4 Kb	3.91	41.8 sec	577 ms	2.27	4.09	2.86	3.72
26	5.23 days	1.84 min	162 Kb	6.53	1.5 min	973 ms	2.31	4.13	3.02	3.89
28	20.9 days	3.68 min	322 Kb	6.65	3.22 min	1.05 sec	2.35	4.17	3.19	4.06
30	83.6 days	7.35 min	642 Kb	11.8	6.89 min	1.93 sec	2.39	4.21	3.36	4.22
32	335 days	14.7 min	1.25 Mb	11.9	14.7 min	2.05 sec	2.43	4.25	3.52	4.39
34	3.67 years	29.4 min	2.5 Mb	22	31.2 min	4.01 sec	2.46	4.29	3.69	4.55
36	14.7 years	58.8 min	5 Mb	22.1	1.1 hrs	4.24 sec	2.5	4.32	3.85	4.72
38	58.7 years	1.96 hrs	10 Mb	42.2	2.32 hrs	8.56 sec	2.54	4.36	4.02	4.89
40	235 years	3.92 hrs	20 Mb	42.4	4.89 hrs	9.01 sec	2.58	4.4	4.19	5.05

Table 2: Timing for RSA and comparison between our first improvement to the Merkle signature scheme (1Mss) and our final version (2Mss): Key Generation Time and Private Key Size, Signature Process Time and Signature Size and Verification Time. The number of possible signatures is 2^N and the public-key size is 24 bytes.

attack for finding collisions were the birthday attack, and the Moore's law would apply, and the $500MHz$ microprocessors were available in the middle of 2001 then from [BK04] we have that with 2^{80} trials $\approx 2^{95.25}$ instructions, the probability for finding a collision is $< \frac{1}{2}$. Taking into account the considerations made by Lenstra and Verheul in [LV01], which are summarized in the formula

$$\frac{L[2^k]}{IMY(y) * 2^{12(y-1999)/r}} \geq \frac{L[2^{512}]}{10^4}$$

we find $y = 2055.55$ and then $k = 2128$, i.e. Merkle is as secure as RSA with such key size. Analogous estimates can be done for hash functions such as SHA-256, SHA-384, and SHA-512.

Brassard et al. present in [BHT88] a quantum algorithm for finding a collision in arbitrary r-to-one function $F : X \to Y$, such an algorithm returns a collision after an expected number of $\Theta(\sqrt[3]{|X|})$ evaluations and uses space $\Theta(\sqrt[3]{|X|})$. Using that algorithm we have that it will be found a collision for H in 44556 years from now with an hypothetical $500MHz$ quantum computer using $\Theta(2^{\frac{160}{3}})$ qubits under the assumption that the quantum computing does not improve the time for computing H values and the Moore's Law does not apply in quantum computers.

5 Conclusion

We showed that the Merkle signature scheme is existentially unforgeable under adaptive chosen message attack and we also presented a forward-secure signature scheme from any cryptographic-secure pseudorandom bit generator, one-time signature scheme and hash function. Using ISAAC, RIPEMD160 and an improved version of the Lamport-Diffie one-time signature scheme shown in Appendix B, we presented a signature scheme which is competitive with RSA for verifying, although the number of possible signatures must be previously fixed. With parameter $N = 38$ or 40 the number of possible signatures is 2^N, which is practically unlimited. This scheme is also secure in the quantum computing epoch provided that the best quantum collision finder is the one proposed in [BHT88].

References

[And02] Ross Anderson. Two remarks on public key cryptology. Technical Report 549, University of Cambridge, December 2002.

[BHT88] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In A. V. Moura C. L. Lucchesi, editor, *LATIN'98, LNCS*, volume 1380, pages 163–169. Springer-Verlag Berlin Heidelberg, 1988.

[BK04] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In Jan Camenisch Christian Cachin, editor, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 401–418. Springer-Verlag Heidelberg, 2004.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - Crypto'99*, volume 1666, August 1999.

[BMS03] Dan Boneh, Ilya Mironov, and Victor Shoup. A secure signature scheme from bilinear maps. In *Topics in Cryptology – CT-RSA*, 2003.

[BY03] Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In M. Joye, editor, *Topics in Cryptology - CT-RSA'03*, Lectures Notes in Computer Science. Springer-Verlag, 2003.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. In *Siam Journal on Computing*, pages 281–308, 1988.

[ISA] ISAAC. Indirection, shift, accumulate, add, and count. http://www.burtleburtle.net/bob/rand/-isaacafa.html.

[Kra00] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In *7th ACM Conference on Computer and Communications Security*, November 2000.

[LV01] A.K. Lenstra and E.R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[Mer90] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology - CRYPTO'89 LNCS*, volume 435. Springer-Verlag Berlin Heidelberg 1990, 1990.

[Mic00] Silvio Micali. Cs proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of times periods. In *Eurocrypt 2002*, volume 2332. Springer-Verlag Berlin Heidelberg, 2002.

[Ope] OpenSSL. Openssl project. http://www.openssl.org/.

[RS04] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Lecture Notes in Computer Science*, volume 3017, pages 371 – 388. Springer-Verlag Heidelberg, 2004.

[Szy03] Michael Szydlo. Merkle tree traversal in log space and time. In *Eurocrypt 2004, LNCS*, volume 3027, pages 541–554, 2003.

A Some Proofs

A stateful generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, b, n)$ is a (t, ϵ) *forward-secure pseudorandom bit generator*, if for all probabilistic algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ running in time at most t

$$Adv_{\mathcal{G}}^{\text{fsprg}}(t) = |Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] - Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1]| < \epsilon,$$

where $\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}$ and $\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}$ are described in Table 3. The probability is taken over the coins tosses of $\mathcal{G}.key, \mathcal{A}_1$ and $\mathcal{A}_2$.

Proposition 5 (Proven in [BY03]) *Let $G : \{0,1\}^s \rightarrow \{0,1\}^{s+b}$ be a $(t_{prg}, \epsilon_{prg})$ pseudorandom bit generator, let n be an integer such that $2n\epsilon_{prg} \leq 1$. Define $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, b, n)$, where $\mathcal{G}.key$ outputs $St \in_R \{0,1\}^s$ and on input St, $\mathcal{G}.next$ obtains $X \leftarrow G(St)$ and outputs (St', Out), where $X = St' \| Out$, $St' \in \{0,1\}^s$ and $Out \in \{0,1\}^b$. Then $\mathcal{G}$ is a $(t_{fsprg}, \epsilon_{fsprg})$ forward secure pseudorandom bit generator, where $\epsilon_{fsprg} = 2n\epsilon_{prg}$ and $t_{prg} = t_{fsprg} + O(n \cdot (s+b))$.*

Proof of Proposition 2

Suppose that the improved Merkle signature scheme can be existentially broken via a 2^N message attack in time t and probability ϵ. Then, there exists $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ a forger such that $Pr[Ver(M, \sigma', PK) = 1 \;:\; (SK, PK) \leftarrow Gen(s, N); T \leftarrow \mathcal{F}_1^{Sig(\cdot, SK)}(s); (M, \sigma') \leftarrow \mathcal{F}_2(T)] \geq \epsilon$ in time t.

Table 3: Experiments

Experiment $\text{Exp}_{\mathcal{G}}^{\texttt{fsprg-1}}(\mathcal{A})$	Experiment $\text{Exp}_{\mathcal{G}}^{\texttt{fsprg-0}}(\mathcal{A})$
$St_{-1} \leftarrow \mathcal{G}.key$	$St_{-1} \leftarrow \mathcal{G}.key$
$i \leftarrow -1$; $h \leftarrow \lambda$	$i \leftarrow -1$; $h \leftarrow \lambda$
Repeat	Repeat
$i \leftarrow i+1$	$i \leftarrow i+1$
$(St_i, Out_i) \leftarrow \mathcal{G}.next(St_{i-1})$	$(St_i, Out_i) \leftarrow \mathcal{G}.next(St_{i-1})$
$(d,h) \leftarrow \mathcal{A}_1(Out_i, h)$	$Out_i \leftarrow \mathcal{U}_b$
Until (d = guess) or (i=n-1)	$(d,h) \leftarrow \mathcal{A}_1(Out_i, h)$
$g \leftarrow \mathcal{A}_2(St_i, h)$	Until (d = guess) or (i=n-1)
Return g	$g \leftarrow \mathcal{A}_2(St_i, h)$
	Return g

Assuming that H is a (t_{cr}, ϵ_{cr}) collision resistant hash function and $1Sign$ is a (t_{1s}, ϵ_{1s}) one-time signature scheme, we prove that it can be found a distinguisher $\mathcal{A}$ for the $(t_{fsprg}, \epsilon_{fsprg})$ forward-secure pseudorandom bit generator $\mathcal{G} = (\mathcal{G}.key, \mathcal{G}.next, s, 2^N)$ obtained from PRG, where $t_{fsprg} = t + t_g + 2^N t_s + t_v$, $\epsilon_{fsprg} = 2^{N+1}\epsilon_{prg}$, and $\mathcal{G}$ is as in Proposition 5.

We construct an Algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ as follows. h is initialized as the null string λ. On input Out and h, $\mathcal{A}_1$ updates $h \leftarrow h \| Out$ and outputs $(\texttt{not\_guess}, h)$. On input St and h, $\mathcal{A}_2$ obtains Out_i from h for $0 \le i < 2^N$ and computes $(X_i, Y_i) \leftarrow 1Gen(s, Out_i)$ $\forall i$ and the Merkle tree's nodes K_{j,k_j} from Y_i $(0 \le j \le N,\ 0 \le k_j < 2^j)$. $\mathcal{A}_2$ retains all X_i, Y_i and K_{j,k_j} and only deletes X_i after its use. $\mathcal{A}_2$ calls $\mathcal{F}_1(s)$ and uses X_i, Y_i and K_{j,k_j} for signing queries from $\mathcal{F}_1$ as an oracle for $\mathcal{F}_1$. $\mathcal{A}_2$ uses the output T from $\mathcal{F}_1$ as input for $\mathcal{F}_2$ in order to obtain (M, σ'). If $\mathcal{A}_2$ obtains a forged signature, then outputs 1. In other case it outputs 0.

Note that in $\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the improved Merkle signature scheme and that in $\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A})$, $\mathcal{A}_2$ obtains a random instance of the original Merkle signature scheme.

Now, by hypothesis $Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] \ge \epsilon$ within time t_{fsprg} and $Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] < \frac{\epsilon}{2}$ within time t_{fsprg} because of the hypothesis, we have that the original Merkle signature scheme is $(t, 2^N, \frac{\epsilon}{2})$ existentially unforgeable under adaptive chosen message attack as proved in Proposition 1.

And so, $Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-1}}(\mathcal{A}) = 1] - Pr[\text{Exp}_{\mathcal{G}}^{\text{fsprg-0}}(\mathcal{A}) = 1] > \frac{\epsilon}{2} \ge 2^{N+1}\epsilon_{prg} \ge \epsilon_{fsprg}$ within time t_{fsprg}, which contradicts the Proposition 5.

□

Proof of Remark 1

Suppose that the Merkle signature scheme can be existentially broken via a 2^N message attack in time t and probability ϵ. Then we prove that at least one of the following holds:

- It can be found a collision for the underlying hash function with probability at least ϵ_{cr} and within time t_{cr}.
- The underlying one-time signature scheme can be existentially broken with probability at least ϵ_{1s} and within time t_{1s}.

Suppose that exists $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ a forger such that $Pr[Ver(M, \langle i, \zeta\rangle) = true\ :\ (SK_0, PK) \leftarrow Gen(s, T, I); (info, k) \leftarrow \mathcal{F}_1^{Sig(\cdot, SK)}(PK); (M, \langle i, \zeta\rangle) \leftarrow \mathcal{F}_2(SK_k, info)$ and $i < k] < \epsilon$ in time t.

Let denote $\sigma' = \langle i, \zeta\rangle$. We construct at the same time two algorithms, $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ which try to forge a one-time signature and $\mathcal{B}$ which tries to find a collision.

$\mathcal{B}$ begins getting $(Pr, Pu) \leftarrow 1Gen(s)$ a pair of one-time signature keys. $\mathcal{A}_1$ chooses $0 \le i_0 < 2^N$ at random and sets $Y_{i_0} := Pu$. Now for $0 \le i < 2^N$ with $i \ne i_0$, $\mathcal{A}_1$ obtains $(X_i, Y_i) \leftarrow 1Gen(s)$ and completes the generation process for the Merkle signature scheme. Let R be the Merkle tree's root. This resulting generated Merkle key is taken by $\mathcal{B}$. $\mathcal{A}_1$ calls $\mathcal{F}_1$ and uses $1Sig(\cdot, Pr)$ at most once as an oracle, just in case $\mathcal{F}_1$ needs to compute the (i_0+1)-th signature. Note that $\mathcal{B}$ knows Pr, so it does not use any oracle to sign with that key. $\mathcal{A}_1$ outputs $info$, where $info$ is the output of $\mathcal{F}_1$. $\mathcal{B}$ retains the number of messages k and the messages M_j for $0 \leq j < k$ used by $\mathcal{F}_1$. On input $info$, $\mathcal{A}_2$ calls $\mathcal{F}_2$ with $info$ and $\{X_j\}_{k \leq j < 2^N}$ as input and obtains (M, σ'), a forged signature σ' of a message M for the previous period i, where $i < k$. Let $\sigma' = (i, \tau', Y', P'_N, \ldots, P'_1)$ be the forged signature for the message M, $\mathcal{A}_2$ outputs (τ', M). At this point, algorithm $\mathcal{A}$ has concluded its running time, while $\mathcal{B}$ continues. Let $\sigma = (i, \tau, Y_i, P_N, \ldots, P_1)$ an authentic signature for the message M_i, where $M_i \neq M$. $\mathcal{B}$ sets $A = (P'_N, \ldots, P'_1)$ and $B = (P_N, \ldots, P_1)$. By hypothesis A is an i-labeled authentication path for the leaf $H(Y')$ and B is an i-labeled authentication path for the leaf $H(Y_i)$ both of them with respect to the root R. If either $A \neq B$ or $Y' \neq Y_i$ then $\mathcal{B}$ outputs either the collision which is found by Lemma 1 or (Y', Y_i) (note that $Y' \neq Y_i$ and $H(Y') = H(Y_i)$ implies a collision). In other case, $\mathcal{B}$ outputs $(\overline{1}, \overline{0})$.

Let $\wp$ be the probability that (Y', A) obtained by $\mathcal{F}$ and (Y, B) obtained by the signature algorithm are different. If $\wp \geq \frac{1}{2}$ then $\mathcal{B}$ finds a collision for H within time t_{cr} with probability at least $\frac{\epsilon}{2} \geq \epsilon_{cr}$. Else with probability $\frac{1}{2^N}$, the forged signature output by $\mathcal{F}$ satisfies that $i = i_0$. Hence, the attack by $\mathcal{A}$ on the one-time signature scheme succeeds with probability at least $\frac{1}{2}\frac{\epsilon}{2^N} \geq \epsilon_{1s}$ within time t_{1s}.

□

Proof of Proposition 4

If the improved version can be existentially broken via a 2^{2N} message attack in time t and probability ϵ, we construct an algorithm that given two random instance of the Merkle signature scheme, the algorithm breaks them within time t_{ms} and probability ϵ_{ms}.

Let (N, R_A) and (N, R_B) be the public keys of two random instance of the Merkle signature scheme and let $\mathcal{O}_X$ be the oracle for signing messages with the private key corresponding to the public key (N, R_X), where $X = A, B$.

Suppose that exists $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ a forger such that

$$Pr[Ver(M, \sigma', PK) = 1 \ : \ (SK, PK) \leftarrow Gen(1^s, N); T \leftarrow \mathcal{F}_1^{Sig(\cdot, SK)}(1^s); (M, \sigma') \leftarrow \mathcal{F}_2(T)] \geq \epsilon$$

in time t.

We construct two algorithms $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ and $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2\}$ at the same time as follows:

On input (N, R_A), $\mathcal{A}_1$ sets $R_0 \leftarrow R_A$. $\mathcal{A}_1$ and $\mathcal{B}_1$ generate the same instance of the improved version of the Merkle signature scheme. On input (N, R_B), $\mathcal{B}_1$ chooses $0 \leq j_0 < 2^N$ at random and sets $Pu_{j_0} \leftarrow (N, R)$. $\mathcal{A}_1$ and $\mathcal{B}_1$ call $\mathcal{F}_1$ and sign queries from $\mathcal{F}_1$ in the following way: let i be the number of created signatures, set $j = \left\lfloor \frac{i}{2^N} \right\rfloor$. $\mathcal{B}_1$ computes $(Pr_j, Pu_j) \leftarrow MGen(1^s, N)$ in case that $j \neq j_0$ and $i \cong 0 \mod 2^N$. $\mathcal{A}_1$ obtains $\zeta_j \leftarrow \mathcal{O}_A(Pu_j)$. $\mathcal{B}_1$ uses $MSig(\cdot, Pr_j)$ in case of $\left\lfloor \frac{i}{2^N} \right\rfloor \neq j_0$ and uses the oracle $\mathcal{O}_B$ in the other case.

$\mathcal{B}_1$ and $\mathcal{A}_1$ output T, the output of $\mathcal{F}_1$.

On input T, $\mathcal{B}_2$ and $\mathcal{A}_2$ call $\mathcal{F}_2$ with input T. Let $(M, (\tau, Pu, \zeta))$ be the forged signature output by $\mathcal{F}_2$. $\mathcal{A}_2$ outputs (Pu, ζ) and $\mathcal{B}_2$ outputs (M, τ).

We have that $\epsilon \leq Pr[Ver(M, (\tau, Pu, \zeta), (N, R_0)) = true] = Pr[MVer(M, \tau, Pu) = true \wedge MVer(Pu, \zeta, R_0) = true]$. The signatures have the form $\tau = (i \mod 2^N, \tau_{1s}, \Upsilon_{1s}, P_N, \ldots, P_1)$ and $\zeta = (\left\lfloor \frac{i}{2^N} \right\rfloor, \zeta_{1s}, \Upsilon'_{1s}, Q_N, \ldots, Q_1)$, with probability $\frac{1}{2^N}$, the forged signature output by $\mathcal{F}_2$ satisfies that $\left\lfloor \frac{i}{2^N} \right\rfloor = j_0$. Let $\wp$ be the probability that the i obtained by $\mathcal{F}_2$ is 0 modulo 2^N. If $\wp > \frac{1}{2}$, then the attack by $\mathcal{A}$ succeeds with probability at least $\frac{1}{2}\frac{1}{2^N}\epsilon \geq \epsilon_{ms}$. If $\wp \leq \frac{1}{2}$, then the attack by $\mathcal{B}$ succeeds with probability at least $\frac{1}{2}\frac{1}{2^N}\epsilon \geq \epsilon_{ms}$.

□

B One-time Signature schemes

Let H be a hash function which output has bit size s. Suggested improved version of the Lamport-Diffie one-time signature given by Merkle.

Key generator algorithm *Gen*. On input s, Gen outputs a pair (X, Y). $X = (x_0, \ldots, x_{s'-1})$ and $Y = (y_0, \ldots, y_{s'-1})$, where $s' = s + 1 + \lfloor \log_2 s \rfloor$, $x_i \in_R \{0,1\}^s$ and $y_i = H(x_i)$ for $0 \leq i < s'$. X is the secret key and Y is the verification key.
Signature algorithm *Sig*. On input the message M and the secret key $X = (x_0, \ldots, x_{s'})$, Sig computes $H(M) \parallel Z = m_0 \cdot \ldots \cdot m_{s'-1}$, where Z is the $1 + \lfloor \log_2 s \rfloor$-bit representation of the quantity of zeros in the bit representation of $H(M)$, and outputs $\tau = (x_{i_1}, \ldots, x_{i_k})$, where $0 \leq i_1 < \cdots < i_k < s'$ and $m_j = 1$ iff $j \in \{i_1, \ldots, i_k\}$. τ is the signature.
Verification algorithm *Ver*. On input the message M, a signature $\tau' = (z_1, \ldots z_l)$ and a verification key Y, Ver computes $m_0 \cdot \ldots \cdot m_{s'-1}$ from M as in the signature process, and outputs $true$ if $l = k$ and $y_{i_j} = H(z_j)$ for $j \in \{\alpha \,:\, m_{i_\alpha} = 1\}$ or outputs $false$ in other case.

In our experiments we have implemented an improved version of the Lamport-Diffie one-time signature scheme. Suppose that $H : \{0,1\}^* \to \{0,1\}^s$ is a hash function and let s'' be an integer such that $s'' = \lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \rceil$

$$M = \overbrace{\boxed{\quad H(\mathcal{M}) \quad}}^{s}\ \overbrace{\boxed{\ \mathrm{Z}\ }}^{s'}\ \overbrace{\boxed{\ \mathrm{O}\ }}^{s'}\ \overbrace{\boxed{\ \mathrm{T}\ }}^{s'} \qquad M = H(\mathcal{M}) \| Z \| O \| T,\ s' = \lfloor \log_2 s \rfloor.$$

Let $\mathcal{M}$ be a message. We represent $H(\mathcal{M})$ as quits (quaternary digits) and let Z, O and T be the corresponding bit string representation of the quantity of zeros, ones and twos in the representation of $H(\mathcal{M})$ as quits, respectively.
Deterministic key generator algorithm *Gen*. On input s and $seed \in \{0,1\}^s$, Gen sets $g_{-1} \leftarrow seed$ and computes $(g_i, x_i) \leftarrow PRG(g_{i-1})$ $0 \leq i < s''$. Gen outputs $Y = H(H^3(x_0), \ldots, H^3(x_{s''-1}))$ as the verifying key and $g = seed$ as the private key.
Signature algorithm *Sig*. On input a message $\mathcal{M}$ and the private key g, Sig computes $M = m_0 \cdot \ldots \cdot m_{s''-1}$ as quits from $\mathcal{M}$. Sig outputs the signature $\tau = (H^{m_0}(x_0), \ldots, H^{m_{s''-1}}(x_{s''-1}))$, where x_i is computed from g as in Gen for $0 \leq i < s''$.
Verification algorithm *Ver*. On input a message $\mathcal{M}$, a signature τ' and a public key Y, Ver computes $M = m_0 \cdot \ldots \cdot m_{s''-1}$ as quits from $\mathcal{M}$. Suppose that $\tau' = (z_0, \ldots, z_{s''-1})$. Ver outputs $true$ if $Y = H(H^{3-m_0}(z_0), \ldots, H^{3-m_{s''-1}}(z_{s''-1}))$ and $false$ in other case.

Figure 2: improved Merkle Key Generation and auxiliary values for signing efficiently

Algorithm: improved Merkle Key Generation *iMGen*

Input: s, N
Output: (Pr, Pu) and auxiliary values
Procedure:

$(Pr_A, Pu_A, \{Auth_{A,k}\}_{0\leq k<N}) \leftarrow MGen(s, N)$;
$(Pr_B, Pu_B, \{Auth_{B,k}\}_{0\leq k<N}) \leftarrow MGen(s, N)$;
$\chi \leftarrow \chi_{A,-1}$; /*$\chi_{A,-1}$ obtained from Pr_A*/
$\chi_C \leftarrow \{0,1\}^s$ /*chosen at random*/
i_mod_2^N = 0;
i_div_2^N = 0;
j = 0;
FL = TRUE;
k_at_level $\leftarrow \vec{0}$;

$\zeta \leftarrow MSig(Pu_B, Pr_A,$i_div_$2^N$,
  $\{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N-1}$);
$Pu = (N, Pu_A)$ /*the public key*/
$Pr = (Pr_A, Pr_B)$ /*the private key*/
/*
Keep the auxiliary values
$\{Auth_{X,k}, Keep_{X,k}, Need_{X,k}\}_{0\leq k<N, X=A,B}$, (ζ, Pu_B),
i_mod_2^N = 0, i_div_2^N = 1, j, FL and k_at_level
*/
return $(Pr, Pu$, and the auxiliary values);

Figure 3: improved Merkle Signature Algorithm using the Szydlo method for computing a sibling path efficiently

Algorithm: improved Merkle Signature *iMSig*

Input: The message M, the private key $Pr = (Pr_A, Pr_B$, and the auxiliary values)
Output: The signature $\sigma = (\tau, Pu_B, \zeta)$
Procedure:

$\tau \leftarrow MSig(M, Pr_B,$i_mod_$2^N$,
  $\{Auth_{B,k}, Keep_{B,k}, Need_{B,k}\}_{k=0}^{N-1}$);
$\sigma \leftarrow (\tau, Pu_B, \zeta)$;
$mMGen(s, N, Pr_C, Pu_C, \chi_C, \{Auth_{C,k}\}_{k=0}^{N-1}$, j,
k_at_level, stack, FL);
if (i_div_2^N <= i_mod_2^N)
  $NLCom(\chi$, i_mod_2^N, $index$, needed_index,
needed_leaf_values);
i_mod_2^N += 1;
if (0 == i_mod_2^N mod 2^N) {
  $(Pr_B, Pu_B) \leftarrow (Pr_C, Pu_C)$;
  $(Pr_C, Pu_C) \leftarrow$ empty_key_pair;
  $\chi_C \leftarrow \{0,1\}^s$ /*chosen at random*/
  j = 0;
  k_at_level $\leftarrow \vec{0}$;
  stack $\leftarrow$ empty_stack;
  FL = TRUE;
  $\chi \leftarrow \chi_{A,\text{i\_div\_}2^N\text{-}1}$;
  $index$ = 0;
  Obtain needed_index from Szydlo_method with
$\{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N-1}$
  needed_leaf_values $\leftarrow$ vector_of_null_bit_strings;
  $\zeta \leftarrow MSig(Pu_B, Pr_A,$i_div_$2^N$,
    $\{Auth_{A,k}, Keep_{A,k}, Need_{A,k}\}_{k=0}^{N-1}$);
  i_mod_2^N = 0;
}
return σ;

Figure 4: improved Merkle Verification Algorithm

Algorithm: improved Merkle Verification $iMVer$

```
Input:  The message $M$, the signature $\sigma = (\tau, Y, \zeta)$ and the public key $(N, R)$
Output:  $true$ if the signature is a valid one or $false$ in other case
Procedure:

if ($MVer(M, \tau, (N, Y))$ == $false$) return $false$;
if ($MVer(Y, \zeta, (N, R))$ == $false$) return $false$;
return $true$
```

Figure 5: modified Merkle Key Generation and auxiliary values for signing efficiently

Algorithm: modified Merkle Key Generation $mMGen$

```
Input:  $s$, $N$, $(Pr_C, Pu_C)$, $\chi$, $\{Auth_{C,k}\}_{0 \leq k < N}$, j, k_at_level, stack and FL
Output:  updated $(Pr_C, Pu_C)$, $\chi$, $\{Auth_{C,k}\}_{0 \leq k < N}$, j, k_at_level, stack and FL
Procedure:

int computed;
hash_value $K$, left, right, root;

computed = 0;
while (0 < j) {
  if (FL) {
    j = $N$;
    if (0 == k_at_level[$N$] $\bmod 2^p$)
      update_Pr($\chi$, $Pr_C$);
/*$\{\chi_{l2^p-1})\}_{0 \leq l < 2^q}$ is being stored */
    $(\chi, seed) \leftarrow PRG(\chi)$;
    $(Pr_{1s}, Pu_{1s}) \leftarrow 1Gen(seed)$;
    $K \leftarrow$H$(Pu_{1s})$;
    computed += 1;
    push(stack, $K$, j, k_at_level[j]);
    k_at_level[j] += 1;
  }
  if (computed == 2) {
    FL = not(two_at_same_level(stack));
    return;
  }
  while(two_at_same_level(stack)) {
    pop(stack, right, j, right_at_level_j);
    pop(stack, left, j, left_at_level_j);
    if (right_at_level[j] == 1)
      $Auth_{C,N-j}$ $\leftarrow$ right;
    j -= 1;
    $K$ = H(concat(left, right));
    computed += 1;
    push(stack, $K$, j, k_at_level[j]);
    k_at_level[j] += 1;
    if (computed == 2) {
      FL = not(two_at_same_level(stack));
      return;
    }
  }
  FL = TRUE;
}
pop(stack, root, j, k_at_level[0]);
$Pu_C \leftarrow$ ($N$, root);
$Pr_C \leftarrow$ ($\{\chi_{l2^p-1}\}_{0 \leq l < 2^q}, \{Auth_{C,k}\}_{k=0}^{N-1}$);
return;
```

Figure 6: Computation of the needed leaf values in the Szydlo method, which will be used in the improved Merkle Signature Algorithm

Algorithm: needed leaf computation $NLCom$

```
Input:  χ, leaf_index, index, needed_index, needed_leaf_values
Output:  updated values for χ, index, needed_index, needed_leaf_values
Procedure:

(χ, seed) ← PRG(χ);
if (leaf_index == needed_index[index]){
  (X, Y) ← 1Gen(s, seed);
  needed_leaf_values[index] ← H(Y);
  index += 1;
}
```

Figure 7: Leaf-Calc Algorithm

Algorithm: Computing a Merkle tree's leaf Leaf-Calc

Input: the private key $Pr = \{\chi_{i-1}\} \cup \{\chi_{l2^p-1}\}_{\lfloor \frac{i}{2^N} \rfloor < l < 2^q}$, the index $leaf$ and the counter i

Output: the value of the leaf whose index is $leaf$

Procedure:

$k \leftarrow \lfloor \frac{leaf}{2^p} \rfloor$;
if i < $k2^p${
  $(G, seed) \leftarrow PRG(\chi_{k2^p-1})$;
  $k \leftarrow k2^p$;
}
else {
  $(G, seed) \leftarrow PRG(\chi_{i-1})$;
  $k \leftarrow i$;
}
for (count=k; count $< leaf$; count++) {
  $(G, seed) \leftarrow PRG(G)$;
}
compute $(Pr_{leaf}, Pu_{leaf}) = 1Gen(s, seed)$;
$L \leftarrow H(Pu_{leaf})$;
return L;

Figure 8: Merkle Key Generation and auxiliary values for signing efficiently

Algorithm: Merkle Key Generation $MGen$

Input: s, N
Output: (Pr, Pu) and $\{Auth_k\}_{0\leq k<N}$
Procedure:

```
int j, k_at_level[N_max];
hash_value K;

χ ← {0,1}^s /* χ_{-1} at random */
k_at_level ← 0⃗;
j = N;
while (0 < j) {
  j = N;
  if (0 == k_at_level[N] mod 2^p) store(χ);
/*{χ_{l2^p-1}}_{0≤l<2^q} is being stored */
  (χ, seed) ← PRG(χ);
  (Pr_{1s}, Pu_{1s}) ← 1Gen(seed);
  K ← H(Pu_{1s});
  push(stack, K, j, k_at_level[j]);
  k_at_level[j] += 1;
  while(two_at_same_level(stack)) {
    right = pop(stack);
    left = pop(stack);
    if (k_at_level[j] == 1)
    Auth_{N-j} ← right.value;
    j = left.j - 1;
    K = H(concat(left.value, right.value));
    push(stack, K, j, k_at_level[j]);
    k_at_level[j] += 1;
  }
}
root = pop(stack);
Pu ← (N, root.value);
Pr ← ({χ_{l2^p-1}}_{0≤l<2^q})
```

Figure 9: Merkle Signature Algorithm using the Szydlo method for computing a sibling path efficiently

Algorithm: Merkle Signature $MSig$

Input: The message M, the private key Pr, the counter i, and $\{Aut_k, Keep_k, Need_k\}_{0\leq k<N}$
Output: The signature $\tau = (i, \tau_{1s}, Y_{1s}, P_N, \ldots, P_1)$
Procedure:

```
(χ_i, seed_i) extract_from_private_key(Pr, i)
(Pr_{1s}, Pu_{1s}) ← 1Gen(s, seed_i)
τ_{1s} ← 1Sig(M, Pr_{1s})
(P_N, ..., P_1) ← Szydlo_method({Auth_k, Keep_k,
                  Need_k}_{0≤k<N}, Pr)
τ ← (i, τ_{1s}, Pu_{1s}, P_N, ..., P_1)
Delete χ_{i-1} from Pr and include χ_i in Pr
Increment i by one
```

Figure 10: Merkle Verification Algorithm

Algorithm: Merkle Verification $MVer$

Input: The message M, the signature $\tau = (i, \tau_{1s}, Y, P_N, \ldots, P_1)$ and the public key (N, R)
Output: $true$ if the signature is a valid one or $false$ in other case
Procedure:

```
hash_value W, left, right;
int j;

if (1Ver(τ_1s, Y) == false) return false;
W = H(Y);
for (j = N; 0 < j; j--) {
  if (⌊i / 2^(N-j)⌋ mod 2 == 0) {
    left = W;
    right = P_j;
  }else {
    left = P_j;
    right = W;
  }
  W = H(concat(left, right));
}
if (W ≠ R) return false;
return true;
```