A Server-Assisted Hash-Based Signature Scheme

Ahto Buldas[1], Risto Laanoja[1,2], and Ahto Truu[1,2,✉] [⋆]

[1] Tallinn University of Technology, Akadeemia tee 15a, 12618 Tallinn, Estonia
[2] Guardtime AS, A. H. Tammsaare tee 60, 11316 Tallinn, Estonia
ahto.truu@guardtime.com

Abstract. We present a practical digital signature scheme built from a cryptographic hash function and a hash-then-publish digital time-stamping scheme. We also provide a simple proof of existential unforgeability against adaptive chosen-message attack (EUF-ACM) in the random oracle (RO) model.

1 Introduction

All the digital signature schemes in use today (RSA [42], DSA [22], ECDSA [30]) are known to be vulnerable to quantum attacks by Shor's algorithm [46]. While the best current experimental results are still toy-sized [35], it takes a long time for new cryptographic schemes to be accepted and deployed, so it is of considerable interest to look for post-quantum secure alternatives already now. Error-correcting codes, discrete lattices, and multi-variate polynomials have been used as foundations for proposed replacement schemes [4]. However, these are relatively complex structures and new constructions in cryptography, so require significant additional scrutiny before gaining trust.

Hash functions, on the other hand, have been studied for decades and are widely believed to be quite resistant to quantum attacks. The best currently known quantum results against hash functions are using Grover's algorithm [25] to find a pre-image of a given k-bit value in $2^{k/2}$ queries instead of the 2^k queries needed by a classical attacker, and Brassard *et al.*'s modification [7] to find a collision in $2^{k/3}$ instead of $2^{k/2}$ queries. To counter these attacks, it would be sufficient to deploy hash functions with correspondingly longer outputs when moving from pre-quantum to post-quantum setting.

2 Related Work

The earliest digital signature scheme constructed from hash functions is due to Lamport [31, 19]. Merkle [37] introduced two methods for reducing the key sizes, one proposed to him by Winternitz. The Winternitz scheme has subsequently

⋆ This research was supported by the European Regional Development Fund through the Estonian smart specialization program NUTIKAS. The final publication is available at Springer via https://doi.org/10.1007/978-3-319-70290-2_1

been more thoroughly analyzed and further refined by Even *et al.* [23], Dods *et al.* [21], Buchmann *et al.* [9], and Hülsing [27]. All of these schemes are *one-time*, and require generation of a new key pair and distribution of a new public key for each message to be signed.

Merkle's arguably most important contribution in [37] was the concept of *hash tree*, which enables a large number of public keys to be represented by a single hash value. With the hash value published, any one of the N public keys can be shown to belong to the tree with a proof consisting of $\log_2 N$ hash values, thus combining N instances of a one-time scheme into an N*-time* scheme. Buldas and Saarepera [16] and Coronado García [17] showed the aggregation to be secure if the hash function used to build the tree is collision resistant. Rohatgi [43] used the XOR-tree construct proposed by Bellare and Rogaway [3] to create a variant of hash tree whose security is based on second pre-image resistance of the hash function instead of collision resistance. Dahmen *et al.* [18] proposed a similar idea with a more complete security proof.

A drawback of the above hash tree constructs is that the whole tree has to be built at once, which also means all the private keys have to be generated at once. Merkle [38] proposed a certification tree that allows just the root node of the tree to be populated initially and the rest of the tree to be grown gradually as needed. However, to authenticate the lower nodes of the tree, a chain of full-blown one-time signatures (as opposed to a chain of sibling hash values) is needed, unless the protocol is used in an interactive environment where the recipient keeps the public keys already delivered as part of earlier signatures. Malkin *et al.* [34] and Buchmann *et al.* [8, 11] proposed various multi-level schemes where the keys authenticated by higher-level trees are used to sign roots of lower-level trees to enable the key sets to be expanded incrementally.

Buchmann *et al.* [10] proposed XMSS, a version of the Merkle signature scheme with improved efficiency compared to previous ones. Hülsing *et al.* [28] introduced a multi-tree version of it. Hülsing *et al.* [29] described a modification hardened against so-called multi-target attacks where the adversary will succeed when it can find a pre-image for just one of a large number of target output values of a hash function.

A risk with the N-time schemes is that they are *stateful*: as each of the one-time keys may be used only once, the signer will need to keep track of which keys have already been used. If this state information is lost (for example, when a previous state is restored from a backup), keys may be re-used by accident.

Perrig [39] proposed BiBa which has small signatures and fast verification, but rather large public keys and slow signing. Reyzin and Reyzin [41] proposed the HORS scheme that provides much faster signing than BiBa. These two are not strictly one-time, but so-called *few-time* schemes where a private key can be used to sign several messages, but the security level decreases with each additional use. Bernstein *et al.* [5] proposed SPHINCS, which combines HORS with XMSS trees to create a *stateless* scheme that uses keys based on a pseudo-random schedule that makes the risk of re-use negligible even without tracking the state.

3 Our Contribution

We propose a signature scheme with a hash function as its sole underlying primitive. At the time of writing, XMSS and SPHINCS are the state of the art in the stateful and stateless hash signature schemes, respectively, so these are what new schemes should be measured against.

XMSS has fast signing and verification, and small signatures, but requires careful management of key state [36]. Our scheme has comparable efficiency, but the private key to be used is determined by signing time, which removes the risk of accidental roll-backs. Also, a single private key can be used to sign multiple messages simultaneously, so no synchronization is required when the scheme is deployed in multi-threaded or multi-processor environments.

SPHINCS has small keys and efficient verification, but quite large signatures and rather expensive signing. Our scheme requires orders of magnitude less computations for signing and produces signatures roughly a tenth the size.

A more general feature is that each signature produced by our scheme is inherently time-stamped. Most other schemes require time-stamping as a separate step after signing to handle key expirations, key revocations, and time-limited signing authority. Due to the time-stamping component, our scheme is necessarily server-assisted. While this may look like a disadvantage, it may in fact be beneficial in enforcing various key usage policies and limiting damage in case of a key leakage. For these reasons, even the technically off-line schemes are usually deployed within on-line frameworks in practice.

4 Preliminaries

Hash Trees. Introduced by Merkle [37], a hash tree is a tree-shaped data structure built using a 2-to-1 hash function $h\colon \{0,1\}^{2k} \to \{0,1\}^k$. The nodes of the tree contain k-bit values. Each node is either a leaf with no children or an internal node with two children. The value x of an internal node is computed as $x \leftarrow h(x_l, x_r)$, where x_l and x_r are the values of the left and right child, respectively. There is one root node that is not a child of any node. We will use $r \leftarrow T^h(x_1, \ldots, x_N)$ to denote a hash tree whose N leaves contain the values $x_1, \ldots, x_N$ and whose root node contains r.

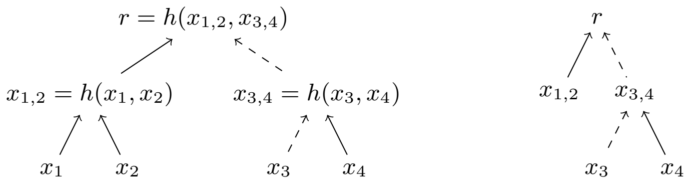

Fig. 1. The hash tree $T^h(x_1, \ldots, x_4)$ and the corresponding hash chain $x_3 \rightsquigarrow r$.

Hash Chains. In order to prove that a value x_i participated in the computation of the root hash r, it is sufficient to present values of all the siblings of the nodes on the unique path from x_i to the root in the tree. For example, to claim that x_3 belongs to the tree shown on the left in Fig. 1, one has to present the values x_4 and $x_{1,2}$ to enable the verifier to compute $x_{3,4} \leftarrow h(x_3, x_4)$, $r \leftarrow h(x_{1,2}, x_{3,4})$, essentially re-building a slice of the tree, as shown on the right in Fig. 1. We will use $x \overset{c}{\rightsquigarrow} r$ to denote that the hash chain c links x to r in such a manner.

Intuitively, it seems obvious that if the function h is one-way, the existence of such a chain whose output equals the original r is a strong indication that x was indeed the original input. However, this result was not formally proven until 25 years after the hash tree construct was proposed [16, 17].

Hash-Then-Publish Time-Stamping. The general idea of time-stamping information by publishing its hash value was used already by Galilei and Hooke in the XVII century. In more modern cryptographic times, Haber and Stornetta [26] were the first to propose time-stamping a sequence of records by having each of them contain the hash of the previous one, in a manner that was later popularized as the *blockchain* structure. Bayer *et al.* [2] proposed using hash trees to aggregate the inputs in batches and then linking the roots of the trees instead of individual records. The most recent results on security bounds of such schemes are by Buldas *et al.* [15, 13, 14].

5 Description of the Scheme

The principal idea of our signature scheme is to have the signer commit to a sequence of keys such that each key is assigned a time slot when it can be used to sign messages and will transition from signing key to verification key once the time slot has passed.

Signing itself then consists of time-stamping the message-key pair in order to prove that the signing operation was performed at the correct time. For simplicity of presentation, we count time in aggregation rounds of the time-stamping service and use the expression "at time t" to mean "during aggregation round t".

More formally, the classic triple of procedures for key generation, signature generation, and signature verification [24] is as follows:

Key generation. To prepare to sign messages at times $1, \ldots, N$, the signer:

1. Generates N signing keys: $(z_1, \ldots, z_N) \leftarrow \mathcal{G}(N, k)$.
   We assume the keys are unpredictable values drawn from $\{0,1\}^k$.
2. Binds each key to its time slot: $x_i \leftarrow h(i, z_i)$ for $i \in \{1, \ldots, N\}$.
3. Computes the public key p by aggregating the key bindings into a hash tree: $p \leftarrow T^h(x_1, \ldots, x_N)$.

The resulting data structure is shown in Fig. 2 and its purpose is to be able to extract hash chains $c_i \leftarrow h(i, z_i) \rightsquigarrow p$ for $i \in \{1, \ldots, N\}$.

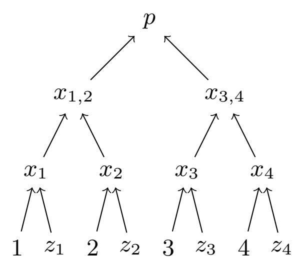

Fig. 2. Computation of public key for $N = 4$.

Signing. To sign message m at time t, the signer:

1. Uses the appropriate key to authenticate the message: $y \leftarrow h(m, z_t)$.
2. Time-stamps the authenticator: $a_t \leftarrow y \rightsquigarrow r_t$.
   Here r_t is the root hash of the aggregation tree built by the time-stamping service for the aggregation round t. We assume the root is committed to in some reliable way, such as broadcasting it to all interested parties, but place no other trust in the service.
3. Outputs the tuple (t, z_t, a_t, c_t), where t is the signing time, z_t is the signing key for time slot t, a_t is the hash chain from the time-stamping service linking the key usage to r_t, and c_t is the hash chain linking the binding of z_t and time slot t to the signer's public key p.

Note that the signature is composed and emitted after the time-stamping step, which makes it safe for the signer to release the key z_t as part of the signature: the aggregation round t has ended and any future uses of the key z_t can no longer be stamped with time t.

Verification. To verify that the message m and the signature $s = (t, z, a, c)$ match the public key p, the verifier:

1. Checks that z was committed as signing key for time t: $h(t, z) \overset{c}{\rightsquigarrow} p$.
2. Checks that m was authenticated with key z at time t: $h(m, z) \overset{a}{\rightsquigarrow} r_t$.

6 Security Proof

Goldwasser *et al* [24] proposed a framework for studying security of signature schemes where the attackers have various levels of access to signing oracles and various requirements on what they need to achieve for the attack to be considered successful (and the scheme broken).

As the highest security level, they defined the concept of *existential unforgeability* (EUF) where an attacker should be unable to forge signatures on any messages, even nonsensical ones.

They also defined the *chosen-message attack* where the attacker can submit a number of messages to be signed by the oracle before having to come up with a forged signature on a new message, and in particular, as the one giving the attacker the most power, the *adaptive chosen-message attack* (ACM) where the attacker will receive each signature immediately after submitting the message and can use any information gained from previous signatures to form subsequent messages.

Luby [33] defined the *time-success ratio* as a way to express the resilience of a cryptographic scheme against attacks as the relationship of the probability that the attack will succeed to the computation time the attacker is allowed to spend.

We will now combine these notions to define and prove the security of our signature scheme.

Definition 1. *A signature scheme is S-secure existentially unforgeable against adaptive chosen-message attacks (EUF-ACM), if any T-time adversary, having access to a signer's public key p and to a signing oracle $\mathcal{S}$ to obtain signatures $s_1 \leftarrow \mathcal{S}(m_1)$, …, $s_n \leftarrow \mathcal{S}(m_n)$ on adaptively chosen messages m_1, …, m_n, can produce a new message-signature pair (m, s) such that $m \notin \{m_1, \ldots, m_n\}$, but s is a valid signature on m, with probability at most T/S.*

```
Oracle $\mathcal{S}$ (signing oracle)
Query Sig($m, t$):
    return $h(m, z_t)$
Query Get($t$):
    If $c \geq t$ then:
        return $(z_t, x_t \rightsquigarrow p)$
    else:
        return $\perp$
```

```
Oracle $\mathcal{R}$ (repository)
Initialize:
    $c \leftarrow 0$
Query Put($r$):
    $c \leftarrow c + 1$
    $r_c \leftarrow r$
Query Get($t$):
    If $c \geq t$ then:
        return $r_t$
    else:
        return $\perp$
```

Fig. 3. The oracles used in the security condition.

To formalize our security assumptions, we introduce three oracles:

We model the publishing of the root hashes of the time-stamping aggregation trees as the oracle $\mathcal{R}$ (Fig. 3, right) that allows each r_t to be published just once.

The signing oracle $\mathcal{S}$ (Fig. 3, left) will compute the message authenticators at any time, but will release only the keys that have already expired for signing (transitioned to verification keys).

We model the hash function h as a random oracle using the *lazy sampling* technique: every time h is queried with a previously unseen input, a new return value is generated by uniform random sampling from $\{0,1\}^k$; when h is queried with a previously seen input, the same value is returned as last time.

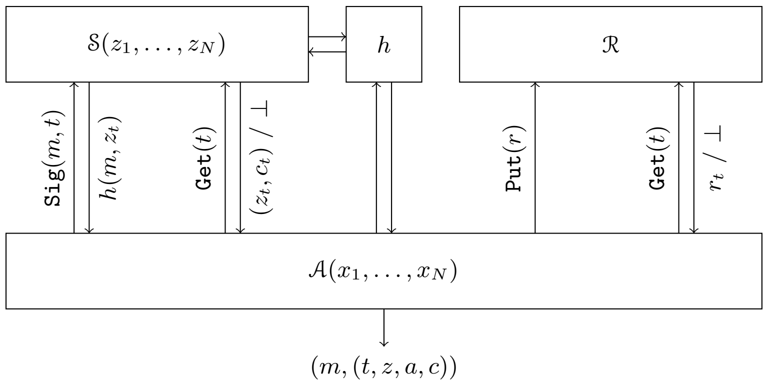

Fig. 4. The adversary's interactions with the oracles.

The adversary $\mathcal{A}$ will be interacting with the oracles as shown in Fig. 4 with the goal of producing a forgery.

To model the fact that the signer needs to keep secret only the keys $z_1,\ldots,z_N$, we explicitly initialize the adversary with $x_1,\ldots,x_N$. Note that the verification rule still assumes that the verifier has access only to the signer's public key p, which means the adversary is not limited to presenting hash chains that were actually extracted from $T^h(x_1,\ldots,x_N)$.

Also note that we leave the aggregation process of the time-stamping service fully under the adversary's control; only the repository $\mathcal{R}$ needs to be trusted to operate correctly.

As normally signing message m involves first calling $\mathcal{S}.\mathtt{Sig}(m,t)$, then committing to $\mathcal{R}$ the root of a hash tree that includes the return value, and then calling $\mathcal{S}.\mathtt{Get}(t)$, we formalize the forgery condition by demanding that the adversary can't make the two $\mathcal{S}$ calls in that order:

Definition 2. *The pair (m,s) produced by an adversary is a successful forgery if s is a valid signature on m, but the adversary did not make the calls $\mathcal{S}.\mathtt{Sig}(m,t)$, $\mathcal{S}.\mathtt{Get}(t)$, in that order, for any $t \in \{1,\ldots,N\}$.*

Theorem 1. *Our signature scheme, when instantiated with a hash function $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ indistinguishable from a random oracle, is at least $2^{k/2-1}$-secure existentially unforgeable against adaptive chosen-message attacks by any T-time adversary.*

Proof. We will directly show an upper bound on the success probability of the adversary in the forgery game $\mathcal{F}$ (Fig. 5).

```
Game $\mathcal{F}$ (forgery)
$(z_1, \ldots, z_N) \leftarrow \mathcal{G}(N)$
$x_i \leftarrow h(i, z_i)$ for $i \in \{1, \ldots, N\}$
$p \leftarrow T^h(x_1, \ldots, x_N)$
$\boxed{(m, (t, z, a, c)) \leftarrow \mathcal{A}^{h,\mathcal{S},\mathcal{R}}(x_1, \ldots, x_N)}$
If $\mathcal{A}$ did not call $\mathcal{S}.\mathtt{Sig}(m, t)$, $\mathcal{S}.\mathtt{Get}(t)$,
    but $h(t, z) \overset{c}{\rightsquigarrow} p$ and $h(m, z) \overset{a}{\rightsquigarrow} r_t$
then:
    return 1
else:
    return 0
```

Fig. 5. The forgery game.

Assume that the adversary does not call $\mathcal{S}.\mathtt{Get}(t)$. To win the game $\mathcal{F}$, he must produce t, z, c such that $h(t, z) \overset{c}{\rightsquigarrow} p$. For that, the output of the last step of the chain computation must equal the root of the tree $T^h(x_1, \ldots, x_N)$. Let's now consider the inputs to that step. If they equal the corresponding children of the root of the tree, we can repeat the reasoning for the second last step and the corresponding node of the tree, and so on. As we walk a finite chain and simultaneously traverse a finite tree from the root towards leaves, one of the following events must eventually happen:

1. We run out of the chain at the same time we run out of the tree. This means the adversary has found t and z such that $x_i = h(t, z)$ for some $i \in \{1, \ldots, N\}$. If $i \neq t$, then the adversary has found a second pre-image for the x_i originally computed as $h(i, z_i)$. With h being a random oracle, the probability of a T-time adversary achieving that for any given i is $\leq T/2^k$. If $i = t$, then the adversary may have found a second pre-image for x_t, with probability $\leq T/2^k$, or may have guessed z_t, also with probability $\leq T/2^k$. Thus the total probability of $h(t, z)$ matching a leaf of the tree is $\pi_{A,1} \leq (N + 1)T/2^k$.
2. We run out of the chain before we run out of the tree. This means $h(t, z)$ matches one of the internal nodes of the tree, say x. This can be the case in two ways:
   (a) The left child of x contains t and the adversary uses the right child of x as z. The probability of any given node having the given value t is $1/2^k$. As there are $N - 1$ candidate nodes and N possible values of t, the total probability is $\leq (N - 1)N/2^k$.
   (b) The adversary has found a second pre-image for x. The probability of a T-time adversary achieving that for any given node is $\leq T/2^k$. As the adversary has $N - 1$ nodes as potential targets for such a hit, the total probability is $\leq (N - 1)T/2^k$.

   Thus the total probability of $h(t, z)$ matching an internal node of the tree is $\pi_{A,2} \leq (N - 1)(N + T)/2^k$.

3. We run out of the tree before we run out of the chain. This means that the adversary has found a pre-image for one of the $2N$ values $\{1, z_1, \dots, N, z_N\}$. The probability of that is $\pi_{A,3} \leq 2NT/2^k$.
4. We encounter a hash step where the output of the step equals an internal node in the tree, say x, but the inputs of the step do not match the children of x. This means the adversary has found a second pre-image for x. The probability of that is $\pi_{A,4} \leq (N-1)T/2^k$.

So, the total success probability of a T-time adversary who does not call $\mathcal{S}.\mathtt{Get}(t)$ is $\pi_A \leq \pi_{A,1}+\pi_{A,2}+\pi_{A,3}+\pi_{A,4} \leq (N+1)T/2^k+(N-1)(N+T)/2^k+2NT/2^k+(N-1)T/2^k < (N^2+5NT)/2^k$.

Assume now that the adversary does call $\mathcal{S}.\mathtt{Get}(t)$. Then we can, without loss of generality, also assume that

– he calls $\mathcal{S}.\mathtt{Get}(t)$ only after committing r_t, as before that $\mathcal{S}.\mathtt{Get}(t)$ would always return $\perp$, which would provide no useful information;
– he calls $\mathcal{S}.\mathtt{Get}(t)$ only once, as all additional calls to $\mathcal{S}.\mathtt{Get}(t)$ would return the same result, which would provide no new information;
– he never calls $\mathcal{S}.\mathtt{Sig}(m, t)$, as he is not allowed to call $\mathcal{S}.\mathtt{Sig}(m, t)$ before calling $\mathcal{S}.\mathtt{Get}(t)$ according to the security condition, but after calling $\mathcal{S}.\mathtt{Get}(t)$ he already has z_t and can compute $h(m, z_t)$ directly with no need to call the signing oracle any more.

Finally, we can also assume that in order to win the game $\mathcal{F}$, the adversary must produce m and a such that $h(m, z_t) \overset{a}{\rightsquigarrow} r_t$. Indeed, if the adversary wins the game with $h(m, z) \overset{a}{\rightsquigarrow} r_t$ where $z \neq z_t$, then he has not used the information gained from the $\mathcal{S}.\mathtt{Get}(t)$ call and thus could not have done any better than without the call, a case we have already analyzed.

Let H be the set of h-calls $y \leftarrow h(x_1, x_2)$ the adversary made before committing r_t. As the adversary is T-time, we have $|H| \leq T$. Consider now the h-calls to be made during the computation of $h(m, z_t) \overset{a}{\rightsquigarrow} r_t$:

1. If all the calls are in H, then the adversary must have called $h(m, z_t)$ before committing r_t and thus also before learning z_t from the call to $\mathcal{S}.\mathtt{Get}(t)$. This means that the adversary guessed z_t. The probability of a T-time adversary achieving that is $\pi_{B,1} \leq T/2^k$.
2. If none of the calls are in H, then there are two possibilities:
   (a) The value r_t was not returned from any of the calls in H. This means the adversary was able to find a pre-image of r_t after committing it, the probability of which is $\leq T/2^k$.
   (b) The value r_t was returned by some call in H. Since the chain a is computed entirely using calls not in H, the inputs of the final step of the computation represent a second pre-image of r_t. The probability of a T-time adversary achieving that is also $\leq T/2^k$.

   Thus the total probability of the adversary finding a chain entirely outside of H is $\pi_{B,2} \leq 2T/2^k$.

3. Some, but not all of the calls are in H. Let's examine, among the calls that are not in H, the one made last during the computation of the chain. Let it be $y \leftarrow h(x_1, x_2)$. Again, there are two possibilities:
   (a) The value y was not returned from any of the calls in H. However, the next step in a is already a call in H. This means that y is among the inputs of calls in H and the adversary was able to find a pre-image of it. The probability of the adversary achieving that for any given y is $\leq T/2^k$. As there are $2|H|$ possible values of y, the total probability is $\leq 2|H|T/2^k$.
   (b) The value y was returned by some call in H. Since the call $y \leftarrow h(x_1, x_2)$ is not in H, the adversary must have found a second pre-image of y. The total probability of that over all available values of y is $\leq |H|T/2^k$.

   Thus the probability of the adversary finding a chain entering into H is $\pi_{B,3} \leq 3|H|T/2^k \leq 3T^2/2^k$.

Hence the total success probability of a T-time adversary who calls $\mathcal{S}.\mathtt{Get}(t)$ is $\pi_B \leq \pi_{B,1} + \pi_{B,2} + \pi_{B,3} \leq T/2^k + 2T/2^k + 3T^2/2^k = (3T + 3T^2)/2^k$.

Summary. If the adversary does not call $\mathcal{S}.\mathtt{Get}(t)$, he can win the forgery game $\mathcal{F}$ with probability $\pi_A < (N^2 + 5NT)/2^k$. If he does call $\mathcal{S}.\mathtt{Get}(t)$, he can win with probability $\pi_B \leq (3T + 3T^2)/2^k$. Overall, he can win with probability $\pi = \max(\pi_A, \pi_B)$.

Since generating the N keys $z_1, \ldots, z_N$ and making the $2N - 1$ calls to h to compute $x_1, \ldots, x_N$ and $T^h(x_1, \ldots, x_N)$ is something the signers are expected to do routinely, we can assume that $N \ll T$. Already with $N < T/10$, we have $\pi_A < (N^2 + 5NT)/2^k < (T^2/100 + T^2/2)/2^k < T^2/2^k$. With $T > 10N \geq 10$, we have $T^2 > 3T$ and thus $\pi_B \leq (3T + 3T^2)/2^k < 4T^2/2^k$.

Therefore, $\pi = \max(\pi_A, \pi_B) < 4T^2/2^k$, or $T^2/\pi > 2^{k-2}$. As $\pi \leq 1$, we also have $(T/\pi)^2 \geq T^2/\pi$, which yields the claim $T/\pi > 2^{k/2-1}$.

7 Practical Considerations

Key Generation. In the description of the scheme we assumed that the signing keys $z_1, \ldots, z_N$ are unpredictable values drawn from $\{0,1\}^k$, but left unspecified how they might be generated in practice. Obviously they could be generated as independent truly random values, but this would be rather expensive and also would necessitate keeping a large number of secret values over a long time. It would be more practical to generate them pseudo-randomly from a single random seed s. There are several known ways of doing that:

– Iterated hashing: $z_N \leftarrow s$, $z_{i-1} \leftarrow h(z_i)$ for $i \in \{2, \ldots, N\}$.
  This idea of generating a sequence of one-time keys from a single seed is due to Lamport [32] and has also been used in the TESLA protocol by Perrin *et al.* [40]. Implemented this way, our scheme would also bear some resemblance to the Guy Fakes protocol by Anderson *et al.* [1]. Note that the keys have to be generated in reverse order, otherwise the earlier keys released

as signature components could be used to derive the later ones that are still valid for signing. To be able to use the keys in the direct order, the signer would have to either remember them all, re-compute half of the sequence on average, or implement a traversal algorithm such as the one proposed by Schoenmakers [45].

- Counter hashing: $z_i \leftarrow h(s, i)$.
  With a hash function behaving as a random oracle, this scheme would generate keys indistinguishable from truly random values, but there does not appear to be much research on the security of practical hash functions when used in this mode.
- Counter encryption: $z_i \leftarrow E_s(i)$.
  The signing keys are generated by encrypting their indices with a symmetric block cipher using the seed as the encryption key. This is equivalent to using the block cipher in the counter mode as first proposed by Diffie and Hellman [20]. The security of this mode is extensively studied and well understood for all common block ciphers. Another benefit of this approach is that it can be implemented using standard hardware security modules where the seed is kept in a protected storage and the encryption operations are performed in a security-hardened environment.

Time-Stamping. As already mentioned, we side-step the key state management problems [36] common for most N-time signing schemes by making the signing keys not one-time, but *time-bound* instead. This in turn raises the issue of clock synchronization.

We first note that even when the signer's local clock is running fast, premature key release is easy to prevent by having the signer verify the time-stamp on $h(m, z_t)$ before releasing z_t. This is how the condition $c \geq t$ of the signing oracle $\mathcal{S}$ in Fig. 3 should be implemented in practice.

The next issue is that the signer needs to select the key z_t before computing $h(m, z_t)$ and submitting it to time-stamping. If, due to clock drift or network latency, the time in the time-stamp received does not match t, the signature can't be composed. To counter clock drift and stable latency, the signer can first time-stamp a dummy value and use the result to compare its local clock to that of the time-stamping service.

To counter network jitter, the signer can compute the message authenticators $h(m, z_{t'})$ for several consecutive values of t', submit all of them in parallel, and compose the signature using the components whose t' matches the time t in the time-stamps received. Buldas *et al.* [12] have shown that with careful scheduling the latency can be made stable enough for this strategy even in an aggregation network with world-wide scale.

Finally, we note that time-stamping services operating in discrete aggregation rounds are particularly well suited for use in our scheme, as they only return time-stamps once the round is closed, thus eliminating the risk that a fast adversary could still manage to acquire a suitable time-stamp after the signer has released a key.

Efficiency. In the following estimates, we assume the use of SHA-256, a common 256-bit hash function. On small inputs, a moderate laptop can perform about a million SHA-256 evaluations per second. We also assume a signing key sequence containing one key per second for a year, or a total of a bit less than 32 million, or roughly 2^{25} keys.

Using the techniques described above, generation of N signing keys takes N applications of either a hash function or a symmetric block cipher. Binding them into a public key takes $2N-1$ hashing operations. Thus, the key generation in our example takes about 100 seconds.

The resulting public key consists of just one hash value. In the private key, only the seed s has to be kept secret. The signing keys $z_1, \ldots, z_N$ can be erased once the public key has been computed, an then re-generated as needed for signing. The hash tree $T^h(x_1, \ldots, x_N)$ presents a space-time trade-off. It may be kept (in regular unprotected storage, as it contains no sensitive information), taking up $2N-1$ nodes, or about 1 GB, and then the key authentication hash chains can be just read from the tree with no additional computations needed. Alternatively, one can use a hash tree traversal algorithm, such as the one proposed by Szydlo [47], to keep only $3 \log_2 N$ nodes of the tree and spend $2 \log_2 N$ hash function evaluations per chain extraction, assuming all chains are extracted consecutively.

The size of the signature (t, z_t, a_t, c_t) is dominated by the two hash chains. The key authentication chain consists of $\log_2 N$ hash values, for a total of about 800 B for our 1-year key sequence. The time-stamping chain consists of $\log_2 M$ hash values, where M is the number of requests received by the time-stamping service in the round t. Assuming the use of the KSI service described in [12] under its theoretical maximum load of 2^{50} requests, this adds about 1 600 B. Thus we can expect signatures of less than 3 kB.

As the verification means re-computing the hash chains, it amounts to less than a hundred hash function evaluations.

8 Conclusions and Outlook

We have presented a simple and efficient digital signature scheme built from a hash function and a hash-then-publish time-stamping scheme. Considering that the existence of hash functions is a necessary pre-condition for the existence of digital signatures [44], one could argue our scheme is based on minimal assumptions. However, there is still much room for improvement in both theoretical and practical aspects.

Current security proofs are given in the random oracle model and in the classical setting. It would be desirable to prove the security also in the standard model and in the quantum setting, in particular taking into account the effects of quantum-oracle access to the hash function [6] and possible quantum interactions between the aggregation and the hash chain extraction phases of time-stamping, as these are all under the adversary's control.

It would also be good to reduce, or at least defer, the key generation costs, perhaps by adopting some of the incremental tree generation approaches, and to develop a version of the scheme suitable for personal signing devices like smart cards and USB dongles. These devices, in addition to having significantly less memory and computational power, also lack several functional qualities of the full-sized computers: they are powered on only intermittently, and do not have on-board real-time clocks or independent network communication capabilities.

References

1. R. J. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, 1998.
2. D. Bayer, S. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II, Proceedings*, volume 9056 of *LNCS*, pages 329–334. Springer, 1992.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS'93, Proceedings*, pages 62–73. ACM, 1993.
4. D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. *Post-Quantum Cryptography.* Springer, 2009.
5. D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *EUROCRYPT 2015, Proceedings, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015.
6. D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In *ASIACRYPT 2011, Proceedings*, volume 7073 of *LNCS*, pages 41–69. Springer, 2011.
7. G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN'98, Proceedings*, volume 1380 of *LNCS*, pages 163–169. Springer, 1998.
8. J. A. Buchmann, L. C. Coronado García, E. Dahmen, M. Döring, and E. Klintsevich. CMSS—An improved Merkle signature scheme. In *INDOCRYPT 2006, Proceedings*, volume 4329 of *LNCS*, pages 349–363. Springer, 2006.
9. J. A. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert. On the security of the Winternitz one-time signature scheme. *IJACT*, 3(1):84–96, 2013.
10. J. A. Buchmann, E. Dahmen, and A. Hülsing. XMSS—A practical forward secure signature scheme based on minimal security assumptions. In *PQCrypto 2011, Proceedings*, volume 7071 of *LNCS*, pages 117–129. Springer, 2011.
11. J. A. Buchmann, E. Dahmen, E. Klintsevich, K. Okeya, and C. Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *ACNS 2007, Proceedings*, volume 4521 of *LNCS*, pages 31–45. Springer, 2007.
12. A. Buldas, A. Kroonmaa, and R. Laanoja. Keyless signatures' infrastructure: How to build global distributed hash-trees. In *NordSec 2013, Proceedings*, volume 8208 of *LNCS*, pages 313–320. Springer, 2013.
13. A. Buldas and R. Laanoja. Security proofs for hash tree time-stamping using hash functions with small output size. In *ACISP 2013, Proceedings*, volume 7959 of *LNCS*, pages 235–250. Springer, 2013.

14. A. Buldas, R. Laanoja, P. Laud, and A. Truu. Bounded pre-image awareness and the security of hash-tree keyless signatures. In *ProvSec 2014, Proceedings*, volume 8782 of *LNCS*, pages 130–145. Springer, 2014.
15. A. Buldas and M. Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In *ACISP 2010, Proceedings*, volume 6168 of *LNCS*, pages 318–335. Springer, 2010.
16. A. Buldas and M. Saarepera. On provably secure time-stamping schemes. In *ASIACRYPT 2004, Proceedings*, volume 3329 of *LNCS*, pages 500–514. Springer, 2004.
17. L. C. Coronado García. *Provably Secure and Practical Signature Schemes.* PhD thesis, Darmstadt University of Technology, Germany, 2005.
18. E. Dahmen, K. Okeya, T. Takagi, and C. Vuillaume. Digital signatures out of second-preimage resistant hash functions. In *PQCrypto 2008, Proceedings*, volume 5299 of *LNCS*, pages 109–123. Springer, 2008.
19. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.
20. W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proc. IEEE*, 67(3):397–427, 1979.
21. C. Dods, N. P. Smart, and M. Stam. Hash based digital signature schemes. In *Cryptography and Coding, Proceedings*, volume 3796 of *LNCS*, pages 96–115. Springer, 2005.
22. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.
23. S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.
24. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
25. L. K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC, Proceedings*, pages 212–219. ACM, 1996.
26. S. Haber and W. S. Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991.
27. A. Hülsing. W-OTS+—Shorter signatures for hash-based signature schemes. In *AFRICACRYPT 2013, Proceedings*, volume 7918 of *LNCS*, pages 173–188. Springer, 2013.
28. A. Hülsing, L. Rausch, and J. A. Buchmann. Optimal parameters for XMSS MT. In *CD-ARES 2013, Proceedings*, volume 8128 of *LNCS*, pages 194–208. Springer, 2013.
29. A. Hülsing, J. Rijneveld, and F. Song. Mitigating multi-target attacks in hash-based signatures. In *PKC 2016, Proceedings, Part I*, volume 9614 of *LNCS*, pages 387–416. Springer, 2016.
30. D. Johnson, A. Menezes, and S. A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, 1(1):36–63, 2001.
31. L. Lamport. Constructing digital signatures from a one way function. Technical report, SRI International, Computer Science Laboratory, 1979.
32. L. Lamport. Password authentification with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
33. M. Luby. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.
34. T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *EUROCRYPT 2002, Proceedings*, volume 2332 of *LNCS*, pages 400–417. Springer, 2002.

35. E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773–776, 2012.
36. D. A. McGrew, P. Kampanakis, S. R. Fluhrer, S.-L. Gazdag, D. Butin, and J. A. Buchmann. State management for hash-based signatures. In *SSR 2016, Proceedings*, volume 10074 of *LNCS*, pages 244–260. Springer, 2016.
37. R. C. Merkle. *Secrecy, Authentication and Public Key Systems.* PhD thesis, Stanford University, 1979.
38. R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO'87, Proceedings*, volume 293 of *LNCS*, pages 369–378. Springer, 1987.
39. A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *ACM CCS 2001, Proceedings*, pages 28–37. ACM, 2001.
40. A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA broadcast authentication protocol. *CryptoBytes*, 5(2):2–13, 2002.
41. L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *ACISP 2002, Proceedings*, volume 2384 of *LNCS*, pages 144–153. Springer, 2002.
42. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
43. P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *ACM CCS'99, Proceedings*, pages 93–100. ACM, 1999.
44. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM STOC, Proceedings*, pages 387–394. ACM, 1990.
45. B. Schoenmakers. Explicit optimal binary pebbling for one-way hash chain reversal. In *FC 2016, Revised Selected Papers*, volume 9603 of *LNCS*, pages 299–320. Springer, 2017.
46. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.
47. M. Szydlo. Merkle tree traversal in log space and time. In *EUROCRYPT 2004, Proceedings*, volume 3027 of *LNCS*, pages 541–554. Springer, 2004.