BLT+L: Efficient Signatures from Timestamping and Endorsements

Denis Firsov[1,2][a], Henri Lakk[1][b], Sven Laur[3][c] and Ahto Truu[1][d]
[1] *Guardtime, A. H. Tammsaare tee 60, 11316 Tallinn, Estonia*
[2] *Tallinn University of Technology, Tallinn, Estonia*
[3] *University of Tartu, Tartu, Estonia*

Keywords: Digital Signatures, Timestamping, Hash Functions, Endorsements.

Abstract: We propose a new digital signature scheme based on combining cryptographic timestamping with an endorsement scheme, both of which can be constructed from one-way and collision-resistant hash functions. The signature scheme is efficient and allows balancing of key generation and signing time for signature size and verification time. The security analysis is based on a realistic model of timestamping. As part of our construction, we introduce the novel concept of endorsements, which may be of independent interest.

1 INTRODUCTION

In server-assisted digital signature schemes, signers can't produce signatures on their own, but have to co-operate with servers. Historically, the two main motivations for such schemes have been: (a) performance—costly computations can be offloaded from an underpowered device to a more capable server; and (b) security—risks of key misuse can be reduced by either keeping the keys in a server environment (which can presumably be managed better than an end-user's personal device) or having the server perform additional checks (such as getting out-of-band confirmations) as part of the signature generation protocol.

A trivial solution would be to just let the server handle the signing operations based on requests from the users, but then the server would have to be completely trusted. To address this, Asokan et al. proposed a method where a user can prove the server's misbehavior when presented with a signature that the server created without the user's request (Asokan et al., 1997). However, such signatures appear valid to a verifier until challenged by the signer. Thus, this protocol is usable in contexts where a dispute resolution process exists, but unsuitable for applications with immediate and irrevocable effects, such as authentication for access control purposes or committing transactions to append-only ledgers. This also applies to later improvements of the approach (Bicakci and Baykal, 2004; Goyal, 2004).

Several methods have been proposed for outsourcing the more expensive computation steps of specific signature algorithms, notably RSA, but most early schemes have been shown to be insecure. Due to increasing computational power of handheld devices and wider availability of hardware-accelerated implementations, attention has now shifted to splitting keys between end-user devices and back-end servers to improve the security of the private keys (Camenisch et al., 2016; Buldas et al., 2017a).

Motivated by the legally binding electronic signature use case and the lack of post-quantum security of underlying algebraic signature schemes Buldas, Laanoja, and Truu recently constructed the new type of server-assisted hash-based signature schemes (BLT schemes in the following) from backdating resistant cryptographic timestamping (Buldas et al., 2017b; Buldas et al., 2018; Buldas et al., 2019). Their protocols produce small signatures with efficient signing and verification, but suffer from expensive key generation.

In this work, we propose the BLT+L signature scheme that improves on their results as follows:

- In contrast with the existing BLT schemes, where key generation phase is expensive by design, our BLT+L scheme defers the key generation costs; additionally, the BLT+L scheme is a framework that allows key generation and signing time to be balanced for signature size and verification time,

[a] https://orcid.org/0000-0003-1267-7898
[b] https://orcid.org/0000-0001-8356-2540
[c] https://orcid.org/0000-0002-9891-3347
[d] https://orcid.org/0000-0001-7427-5005

Firsov, D., Lakk, H., Laur, S. and Truu, A.
BLT+L: Efficient Signatures from Timestamping and Endorsements.
DOI: 10.5220/0010530000750086
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 75-86
ISBN: 978-989-758-524-1

depending on the priorities of the use case.

- Our security analysis is based on a more realistic model of timestamping. In particular, we handle the need for timestamping aggregation layers which improve the performance of hash-then-publish timestamping services. In our analysis, we assume only the publisher to be trusted while treating the aggregation layers as black-box and potentially adversarial.
- We carry out our construction and security analysis in the concurrent model and explicitly account for network delays that may be controlled by the adversary.

To support our constructions, we developed the notion of *endorsement schemes*, which may be of independent interest. Endorsements (Sect. 3) share similarities with signatures and accumulators, and can be constructed from either of those. However, endorsements have strictly weaker security requirements, and therefore allow more efficient specialized constructions.

2 BACKGROUND

Definition 2.1. A *signature scheme* is a triple of probabilistic polynomial-time (PPT) algorithms $(G_{\mathcal{S}}, S_{\mathcal{S}}, V_{\mathcal{S}})$, where $G_{\mathcal{S}}$ is a key pair generation algorithm, $S_{\mathcal{S}}$ a signing algorithm, and $V_{\mathcal{S}}$ a signature verification algorithm.

The signature scheme is *correct* if the algorithm $V_{\mathcal{S}}$ agrees with $S_{\mathcal{S}}$ for all valid key pairs and messages:

$$\Pr[(pk, sk) \leftarrow G_{\mathcal{S}} : V_{\mathcal{S}}(pk, S_{\mathcal{S}}(sk, m), m) = 1] = 1.$$

The signature scheme is *existentially unforgeable* under the *chosen message attack* (EUF-CMA) if the probability

$$\Pr\left[\begin{array}{c} (pk, sk) \leftarrow G_{\mathcal{S}},\ (s, m) \leftarrow A^{S_{\mathcal{S}}(sk,\cdot)}(pk) : \\ V_{\mathcal{S}}(pk, s, m) = 1,\ m \notin S_{\mathcal{S}}.log \end{array}\right]$$

is negligible for any PPT adversary A. That is, A has negligible chance of winning the following game:

- The environment generates a key pair (pk, sk).
- The adversary learns the public key pk and gets access to a signing oracle $S_{\mathcal{S}}$ which contains the secret key sk. The adversary is then allowed to query the oracle for signatures on messages of his choice. The oracle $S_{\mathcal{S}}$ keeps an internal variable log which records the queries.
- The adversary wins if the produced signature-message pair (s, m) verifies and the message m is fresh (was not previously submitted to the oracle).

In a *one-time* signature scheme (OTS), a secret key can be used to sign at most one message and key re-use may result in catastrophic security loss. In a *stateful* scheme, the signer has an internal state that must be updated by the signing algorithm; rolling back to an earlier state may result in security loss. A signature scheme is *server-assisted* if the signer needs support of an external service to generate signatures.

For an offline and stateless signature scheme, its security in sequential computational model implies also security in concurrent setting. The situation is different in case of server-assisted signature schemes, since adversaries could take advantage of asynchronous calls to the server and combine information from different execution threads.

In this work, we build a signature scheme assisted by a *timestamping service* (cf. Sec. 2.1) and this motivates us to interpret the existential unforgeability stated in Def. 2.1 in the concurrent model. More specifically, we assume that the adversary can call the signing oracle asynchronously and also manipulate network delays between the signer and the timestamping service.

Definition 2.2. A *message authentication code* (MAC) is a triple of PPT algorithms $(G_{\mathcal{M}}, S_{\mathcal{M}}, V_{\mathcal{M}})$, where $G_{\mathcal{M}}$ is a key generation algorithm, $S_{\mathcal{M}}$ a tagging algorithm, and $V_{\mathcal{M}}$ a tag verification algorithm. Similarly to a signature scheme, a MAC is *correct* if

$$\Pr[sk \leftarrow G_{\mathcal{M}} : V_{\mathcal{M}}(sk, S_{\mathcal{M}}(sk, m), m) = 1] = 1$$

and *existentially unforgeable* under the *chosen message attack* (EUF-CMA) if

$$\Pr\left[\begin{array}{c} sk \leftarrow G_{\mathcal{M}},\ (s, m) \leftarrow A^{S_{\mathcal{M}}(sk,\cdot)} : \\ V_{\mathcal{M}}(sk, s, m) = 1,\ m \notin S_{\mathcal{M}}.log \end{array}\right]$$

is negligible for any PPT adversary A.

Definition 2.3. A *cryptographic accumulator* $\mathcal{A}$ is a quadruple $(G_{\mathcal{A}}, D_{\mathcal{A}}, P_{\mathcal{A}}, V_{\mathcal{A}})$ of PPT algorithms, where $G_{\mathcal{A}}$ outputs a public key pk, $D_{\mathcal{A}}(pk, S)$ computes a digest (compressed representation) of the set S, $P_{\mathcal{A}}(pk, S, m)$ outputs a membership proof p of m in S, and $V_{\mathcal{A}}$ is a membership verification algorithm, so that $V_{\mathcal{A}}(pk, D_{\mathcal{A}}(pk, S), P_{\mathcal{A}}(pk, S, m), m)$ outputs 1 if $m \in S$, and 0 if $m \notin S$. The accumulator is *collision-resistant* if the following probability is negligible for any PPT adversary A:

$$\Pr\left[\begin{array}{c} pk \leftarrow G_{\mathcal{A}},\ (S, p, m) \leftarrow A(pk) : \\ V_{\mathcal{A}}(pk, D_{\mathcal{A}}(pk, S), p, m) = 1,\ m \notin S \end{array}\right].$$

Definition 2.4. A hash function $h : D \to R$ is *preimage resistant* (*one-way*) if the probability

$$\Pr\left[x \xleftarrow{\$} D,\ x' \leftarrow A(h(x)) : h(x') = h(x)\right]$$

is negligible for any PPT adversary A.

Definition 2.5. A family $\mathcal{H}$ of hash functions is *collision resistant* if the probability

$$\Pr\left[h \xleftarrow{\$} \mathcal{H},\ (x_1,x_2) \leftarrow A(h) : x_1 \neq x_2,\ h(x_1) = h(x_2)\right]$$

is negligible for any PPT adversary A.

2.1 Cryptographic Timestamping

Cryptographic timestamping allows users to prove that some data existed at some moment in the past. Haber and Stornetta introduced the first protocol which did not rely on trusted service providers. They proposed a scheme where each timestamp would include a hash of the immediately preceding one and a reference to the immediately succeeding one (Haber and Stornetta, 1991). Benaloh and de Mare showed how to increase the efficiency by operating in rounds, where the messages to be timestamped within one round would be aggregated into an accumulator (Benaloh and de Mare, 1991).

All bounded binding commitment schemes (D_C,T_C,V_C) can be used to construct timestamping services. Practical instantiations of such services usually involve a *timestamping aggregation server* $\mathcal{X}$ and a *publisher* $\mathcal{P}$. It is assumed that the publisher is an append-only trusted party whose resources are limited and very expensive (e.g., public blockchain, newspaper, etc.). Module 1 presents an ideal implementation of publisher which keeps the current time value in variable t and m is a mapping of previous time values to data items.

The server $\mathcal{X}$ operates in rounds. During each round, it collects requests $y_1,\dots,y_n$ from the users. When the t-th round is over, $\mathcal{X}$ computes the digest $d_t \leftarrow D_C(y_1,\dots,y_n)$ and sends it in an authenticated manner to $\mathcal{P}$, who then makes it public. After that, $\mathcal{X}$ computes certificates $(c_1,\dots,c_n) \leftarrow T_C(y_1,\dots,y_n)$ and issues (t,c_i) as a timestamp for y_i. Intuitively, this proves that y_i existed at the time t when d_t was published. To verify a timestamp (t,c) on y, the user queries the publisher for digest d_t and runs $V_C(d_t,c,y)$. Note that for this scenario to be practical, the digest must be reasonably small.

In this work, we will be interested in timestamping servers based on bounded and binding tag-commitment schemes, where each committed input is accompanied by a user-chosen tag value.

Definition 2.6. A *tag-commitment scheme* $\mathcal{C}$ consists of a triple of PPT algorithms (D_C,T_C,V_C):

- $D_C((a_1,x_1),\dots,(a_n,x_n))$ outputs a digest d which is a compressed representation of its input. The input to D_C consists of data items x_i, each paired with a tag a_i. If the input list contains multiple x's for a tag a, the first such pair is kept and remaining ones are ignored.
- $T_C((a_1,x_1),\dots,(a_n,x_n))$ outputs $(c_1,\dots,c_n)$, where c_i is a certificate (opening) for the pair (a_i,x_i).
- $V_C(d,c,(a,x))$ checks whether the certificate c is a valid proof of inclusion of the pair (a,x) in the digest d.

The tag-commitment scheme is *binding* if any PPT adversary A is unable to open the item associated with the tag a in two different ways, i.e., the following probability is negligible:

$$\Pr\left[\begin{array}{c}(d,a,c_1,x_1,c_2,x_2) \leftarrow A : x_1 \neq x_2,\\ V_C(d,c_1,(a,x_1)) = V_C(d,c_2,(a,x_2)) = 1\end{array}\right].$$

The tag-commitment scheme is *bounded* if every digest d contains a natural d_b which limits the number of elements in it. The opening c is valid for (a,x) iff $V_C(d,c,(a,x)) = 1$ and $\mathsf{loc}(a) \leq d_b$, where $\mathsf{loc}(\cdot)$ is a fixed injective mapping to naturals.

Module 1: Publisher implementation.

```
module $\mathcal{P}$
    var $t : \mathbb{N}$, $m : map$
    fun $publish(t', d) = \{$
        if $t < t'$ then
            $t \leftarrow t'$
            $m[t] \leftarrow d$
        end if
    $\}$
    fun $get(t) = \{$return $m[t]\}$
end
```

If a timestamping server $\mathcal{X}$ is built from a tag-commitment scheme then each request it serves must be a tuple $y = (a,x)$, where a is a user-picked tag and x is the document to be timestamped.

We consider the model where the commitment scheme $\mathcal{C} = (D_C,T_C,V_C)$ is publicly known and the publisher $\mathcal{P}$ is a trusted party with limited and expensive resources, but the server $\mathcal{X}$ which implements the scheme $\mathcal{C}$ is black-box and possibly adversarial. Moreover, the server potentially receives a large number of queries per round from multiple users and then every user only gets the timestamps for her queries and does not see the queries the other users submitted in that round. As a result, the user has no way of verifying that the server is honest and publishes digests which are indeed computed by D_C. The last point makes it non-trivial to formally define the security of timestamping services.

The main security property associated with timestamping servers is *backdating resistance*. Buldas and Laur studied different variations of this property to capture the idea that once a user of timestamping server receives a timestamp on her "fresh" document and the associated digest is reliably published, then no adversary will be able to present a timestamp for "similar" document and earlier time (Buldas and Laur, 2006). We skip the technical definition of backdating resistance, but mention that every bounded binding commitment scheme induces a secure backdating resistant timestamping service (Buldas and Laur, 2007). For our purposes, we only need the binding property of the underlying bounded tag-commitment schemes.

3 ENDORSEMENT SCHEME

Endorsement scheme is a novel concept motivated by the design of the BLT+L signature scheme. The main goal of endorsement schemes is to allow users to authenticate (or endorse) elements from a public list M. If M is reasonably small then an obvious solution is to include M into the public key of the user. This approach is not practical if M is large.

Each BLT+L key pair includes an activation and an expiration time. The design of BLT+L requires the user to authenticate a tuple of hash values for each time slot (i.e., round of timestamping service) within the entire lifespan of a key pair. Hence, if the round duration of the timestamping server is one second, and we need to generate a key pair with a lifespan of ten years then $|M| > 3 \cdot 10^8$ and announcing M as part of the public key is not a feasible solution. Instead, we propose an *endorsement scheme* which comes with its own succinct public key and can be used to endorse elements of the list M.

Definition 3.1. An *endorsement scheme* $\mathcal{E}$ is a triple of PPT algorithms $(G_\mathcal{E}, E_\mathcal{E}, V_\mathcal{E})$, where for any input list M:

- $G_\mathcal{E}(M)$ is a key-generation algorithm that produces a public key pk and a secret key sk.
- $E_\mathcal{E}(sk, M, t)$ is an endorsement-generation algorithm that, given as input the secret key sk, the list M, and an index $1 \le t \le |M|$, produces an endorsement of message M_t.
- $V_\mathcal{E}(pk, e, m, t)$ is a verification algorithm that, given a public key pk, an endorsement e, a message m, and an index t, returns either 0 or 1.

The endorsement scheme is *correct* if $V_\mathcal{E}$ agrees with $E_\mathcal{E}$ for all valid inputs:

$$\Pr\left[\begin{array}{c}(pk,sk) \leftarrow G_\mathcal{E}(M),\ t \leftarrow \{1,\dots,|M|\}: \\ V_\mathcal{E}(pk, E_\mathcal{E}(sk,M,t), M_t, t) = 1\end{array}\right] = 1.$$

The endorsement scheme is *collision-resistant* if the following probability is negligible for any PPT adversary A:

$$\Pr\left[\begin{array}{c}M \leftarrow A,\ (pk,sk) \leftarrow G_\mathcal{E}(M), \\ (e,m,t) \leftarrow A^{E_\mathcal{E}(sk,M,\cdot)}(pk): \\ V_\mathcal{E}(pk,e,m,t) = 1,\ 1 \le t \le |M|,\ m \ne M_t\end{array}\right].$$

Here, A chooses the list M and then receives the public key pk, and access to the oracle $E_\mathcal{E}$ initialized with the secret key sk and the list M. The adversary then tries to produce an endorsement e, a message m, and an index t which verify with pk, so that M_t exists, but m is different from it.

Theorem 3.1. Every collision-resistant accumulator $\mathcal{A}$ gives rise to a collision-resistant endorsement scheme $\mathcal{E}$ as follows:

- $G_\mathcal{E}(M)$: Generate the public key of $\mathcal{A}$ by $pk \leftarrow G_\mathcal{A}$. Compute the digest $d \leftarrow D_\mathcal{A}(pk, M')$, where $M' = \{(t, M_t) : 1 \le t \le |M|\}$. Return (pk, d) as the public key and pk as the "secret" key.
- $E_\mathcal{E}(pk, M, t) := P_\mathcal{A}(pk, M', M'_t)$.
- $V_\mathcal{E}((pk,d), e, m, t) := V_\mathcal{A}(pk, d, e, (t, m))$.

It is straightforward to verify that whenever the adversary breaks collision resistance of the accumulator-induced endorsement scheme, he also breaks collision resistance of the underlying accumulator.

The main disadvantage of accumulators is that the computations can become inefficient for large lists. For example, Merkle trees (Merkle, 1979) are hash-based collision-resistant accumulators, but the complexities of public key and inclusion proof generation are proportional to the size of M, which is prohibitively slow for the BLT+L use case.

Every signature scheme also induces an endorsement scheme in a trivial way. However, the security requirement, namely collision-resistance, associated with endorsements is much lighter than the existential unforgeability of signatures, and we expect specifically designed endorsements to be more efficient than the more general signatures or commitments.

3.1 Goldreich-Merkle Endorsement

Goldreich proposed a stateless many-time signature scheme (GSS) using a binary authentication tree (cf. Sec. 6). Here we combine the top-down (assuming root is "up") authentication mechanism of Goldreich signatures with the bottom-up mechanism of Merkle trees to design an efficient and secure hash-based endorsement scheme.

More specifically, we build a tree with two types of inner nodes: a Goldreich node has an associated OTS key pair and authenticates its children with a one-time signature; a Merkle node authenticates its children with a hash value of their concatenation. We then define the *Goldreich-Merkle endorsement scheme* based on a complete binary tree where all nodes are numbered top-down left-to-right and leaves contain the elements of M. It should be easy to see that to accommodate $|M|$ leaves, the tree must have $2|M|-1$ nodes in total, so the height of such a tree is $\lceil \log_2(|M|)+1 \rceil$, and M_t is located in the node number $t+|M|-1$.

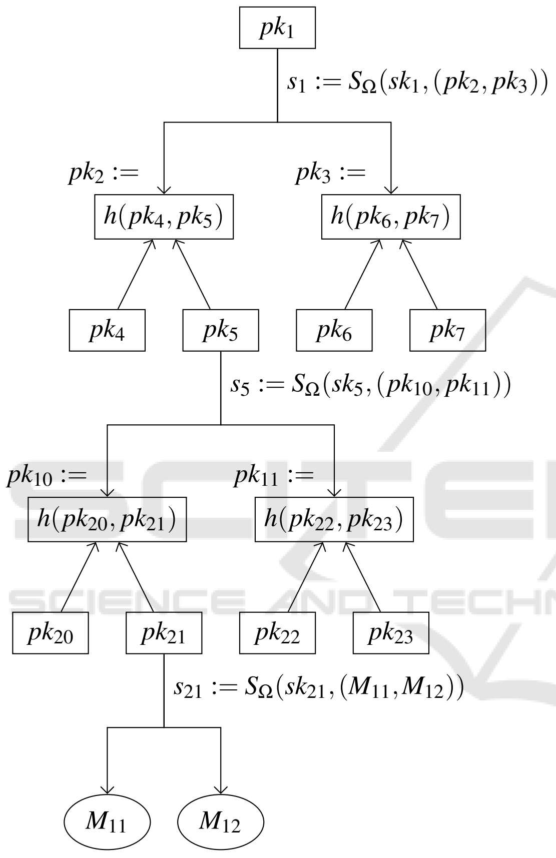

Figure 1: The tree slice needed to endorse M_{11} and M_{12}.

Definition 3.2. Let Ω be a one-time signature scheme and $\mathcal{H}$ a hash function family. Fix a coloring function $f : \mathbb{N} \to \{\mathsf{G},\mathsf{M}\}$. The *Goldreich-Merkle endorsement scheme* induced by Ω, $\mathcal{H}$, and f is defined as follows.
Key Generation. Pick a hash function $h \leftarrow \mathcal{H}$. Generate a list K of $2|M|-1$ tuples (pk_i, sk_i), where

$$(pk_i, sk_i) \leftarrow \begin{cases} (M_{i-(|M|-1)}, \perp) & \text{if } i \geq |M|, \\ (h(pk_{2i}, pk_{2i+1}), \perp) & \text{if } f(i) = \mathsf{M}, \\ G_\Omega & \text{if } f(i) = \mathsf{G}. \end{cases}$$

The public key of the endorsement scheme is $(pk_1, |M|, h)$ and the secret key is K, which is the list of the tree nodes. The coloring function defines which of the inner nodes of the endorsement tree are Goldreich nodes and which are Merkle nodes, and presents an optimization opportunity, as discussed in Sec. 5.
Endorsing. To endorse the t-th element of the list M, construct the authentication path from the root of the endorsement tree to the leaf containing M_t. For every Goldreich node i on the path, add to the endorsement its public key pk_i, the one-time signature $S_\Omega(sk_i,(pk_{2i},pk_{2i+1}))$, and pk_j of the child j (where $j=2i$ or $j=2i+1$) that is not on the path to M_t. For every Merkle node i on the path, add pk_j of the child j that is not on the path to M_t.
Verification. To verify the triple (e,m,t) with respect to the public key $(pk', |M|, h)$, where pk' is the public key of the root of the endorsement tree, first check that $1 \leq t \leq |M|$. Next, verify that e represents the path going from the root to the node $t+|M|-1$ in the complete binary tree with $2|M|-1$ nodes using h. Then, verify that every node in e is authenticated by its parent node. Finally, check that the public key in the first element in e is pk'.

Example. Let $|M| = 32$. Then Fig. 1 illustrates the tree slice needed to endorse the message M_{12}. The authentication tree is based on coloring function which assigns Goldreich type to nodes at odd levels and Merkle type to nodes at even levels. The resulting endorsement consists of the following records: (pk_1,s_1,pk_3), (pk_4), (pk_5,s_5,pk_{11}), (pk_{20}), and (pk_{21},s_{21},M_{11}). This can be verified against $(pk',|M|,h)$ bottom-up as follows:

- check that $V_\Omega(pk_{21},s_{21},(M_{11},M_{12})) = 1$;
- compute $pk_{10} \leftarrow h(pk_{20},pk_{21})$;
- check that $V_\Omega(pk_5,s_5,(pk_{10},pk_{11})) = 1$;
- compute $pk_2 \leftarrow h(pk_4,pk_5)$;
- check that $V_\Omega(pk_1,s_1,(pk_2,pk_3)) = 1$;
- check that $pk_1 = pk'$.

Theorem 3.2. Let Ω be a one-time signature scheme and $\mathcal{H}$ a collision-resistant hash function family. Then for any adversary A and list M, the probability of A breaking collision-resistance of the induced Goldreich-Merkle endorsement scheme does not exceed $p_\mathcal{H} + p_\Omega \cdot w$, where $p_\mathcal{H}$ is the adversary's probability of breaking collision resistance of $\mathcal{H}$, p_Ω the adversary's probability of breaking existential unforgeability of Ω, and w is the number of Goldreich nodes in the authenticaton tree.

3.2 Lazy Evaluation and Caching

The storage complexity of the entire Goldreich-Merkle endorsement tree can be very high. However, only a slice of the tree is needed to produce an endorsement. The exact size of the slice is determined by the underlying coloring function (see Sec. 5).

Also, note that the key generation phase of the Goldreich-Merkle endorsement scheme requires the user to populate the list K (i.e., secret key) with OTS key pairs computed by G_Ω. To defer the cost of key generation, the elements of the list K may be lazily evaluated on demand from a pseudo-random function (PRF). More specifically, during the key generation phase instead of producing all entries of the secret key K we only compute the key pair associated with the root node of the endorsement tree. The remaining keys are evaluated on demand during the endorsing phase.

On the other hand, the key pairs of the nodes which are close to the root of the authentication tree are needed frequently, and it may be more efficient to pre-generate and cache them, as discussed in Sec. 5.

4 BLT+L SIGNATURE SCHEME

The main idea behind the BLT+L signature scheme is to sign messages by timestamping them together with time-related tags. We must take into account that a client device may send multiple asynchronous timestamping requests at time t and the responses can arrive in any order and may be associated with different times of the form $t+\ell$, where ℓ is an adversary-controlled lag (network delay).

Throughout the lifespan of a BLT+L key pair, the scheme uses many one-time key pairs (pk_i, sk_i), one for each round of the timestamping service. To tolerate network lags up to L rounds, we let each user have a list of tuples of the form $sk_i = (r_i^0, \ldots, r_i^L)$ as the secret key and a list of tuples $pk_i = (h(r_i^0), \ldots, h(r_i^L))$ as the public key, where each r_i^j is sampled from an unpredictable distribution, and i corresponds to a time when the key should be usable for signing.

To sign a message m at time t, timestamp the hash $h(m)$ under the tag r_t^0. If the timestamp on $h(m)$ is c and the respective digest d was published by $\mathcal{P}$ at time $t+\ell$ then reveal the ℓ-th component of the tuple sk_t, namely r_t^ℓ, and let the signature be $(t, \ell, c, r_t^0, r_t^\ell)$.

To verify such a signature, the user must query the publisher $d \leftarrow \mathcal{P}.get(t+\ell)$, verify the timestamp by checking that $V_\mathcal{C}(d, c, (r_t^0, h(m))) = 1$, and verify the tokens r_t^0 and r_t^ℓ against the public key component $pk_t = (pk_t^0, \ldots, pk_t^L)$ by checking that $h(r_t^0) = pk_t^0$ and $h(r_t^\ell) = pk_t^\ell$.

The signature scheme described above is secure, but if we want the key pair to have the lifespan of n timestamping rounds then both *sk* and *pk* must contain n of the $L+1$-element tuples, which would make the size of the key pair proportional to its lifespan. This problem can be solved by generating the elements of *sk* and *pk* on demand from a PRF and using an endorsing mechanism on the list *pk*.

Another complication is that since we have only one tag for every time t, namely r_t^0, then the described signature scheme can sign no more than one message per round of the timestamping service. We solve this complication by introducing an aggregator service.

Definition 4.1. Let $\mathcal{X}$ be a timestamping service based on the tag-commitment scheme $\mathcal{C}$. We say that $\mathcal{Q}$ is an *aggregator* for $\mathcal{X}$ if it offers the *put* query implemented in the following way:

- $\mathcal{Q}$ is based on accumulator scheme $\mathcal{A}$ and the respective public-key is $pk_\mathcal{Q}$.
- $\mathcal{Q}$ works in rounds.
- During a round the aggregator receives from users tagged messages $(a, m_1), (a, m_2), \ldots$
- Once the round ends, the pairs are accumulated into sets $S_a, S_b, \ldots$, where the set S_x contains all the messages which were paired with the tag x.
- $\mathcal{Q}$ timestamps the tagged digests of the accumulators $(a, D_\mathcal{A}(S_a)), (b, D_\mathcal{A}(S_b)), \ldots$ with $\mathcal{X}$ and receives respective timestamps $(t, c_a), (t, c_b), \ldots$
- Finally, the user who sent the request (a, m) receives the triple (t, c_a, S_a), where (t, c_a) is the timestamp on $D_\mathcal{A}(S_a)$.

In practical deployments, the aggregator can be implemented as an independent server, as part of the client's software, or be coupled with the timestamping service. In our security model, we assume that it is black-box and therefore potentially adversarial.

Now, we are ready to put it all together and define the BLT+L signature scheme.

Definition 4.2. Let $\mathcal{U}$ be a uniform unpredictable distribution, $\mathcal{H}$ a hash function family, $\mathcal{E}$ an endorsement scheme, $\mathcal{Q}$ an aggregator (based on an accumulator $\mathcal{A}$) for the timestamping service $\mathcal{X}$ (based on a tag-commitment scheme $\mathcal{C}$), and $\mathcal{M}$ a message authentication code (MAC). They induce the *BLT+L signature scheme* as follows:

Key Generation. The key generation algorithm receives the parameters C, E, and L, where C is the time from which the key pair is active, E is the number of rounds of $\mathcal{X}$ until the key pair expires, and L is the highest tolerable network delay. The key pair is generated as follows:

1. Generate a MAC key $sk_\mathcal{M} \leftarrow G_\mathcal{M}$.
2. Sample $r_i^j \xleftarrow{\$} \mathcal{U}$ for $0 \le i < E$ and $0 \le j \le L$.
3. Sample a hash function h from $\mathcal{H}$.

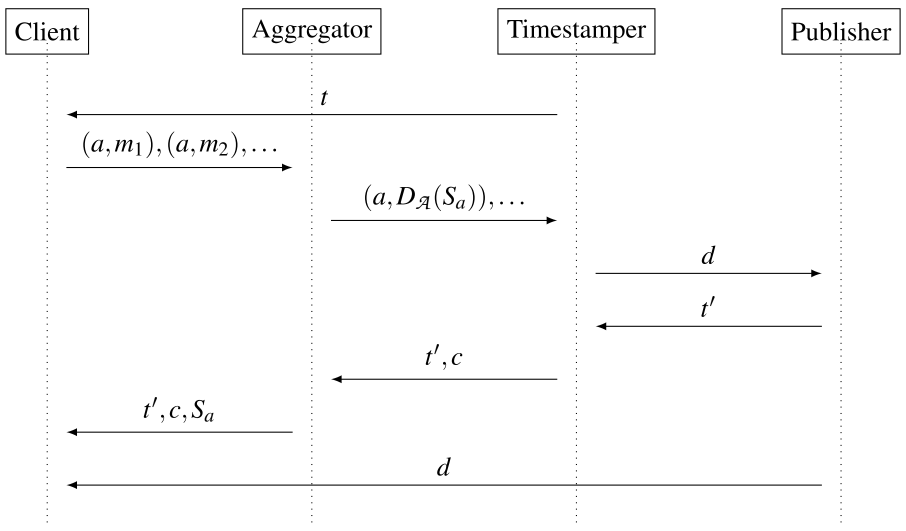

Figure 2: Data flows of the BLT+L signing process.

4. Generate an endorsement scheme key pair $(pk_{\mathcal{E}},sk_{\mathcal{E}}) \leftarrow G_{\mathcal{E}}(M)$, where M is a list such that $M_i = (h(r_i^0),\ldots,h(r_i^L))$ for $0 \leq i < E$.
5. Store the tuple $(sk_{\mathcal{M}},sk_{\mathcal{E}},r)$ as the secret key, and make $(pk_{\mathcal{E}},h,C,E,L)$ available as the public key.

Signing. To sign a message m:

1. Query the time $t \leftarrow \mathcal{X}.clock$.
2. If $C \leq t < C+E$ then set $i \leftarrow t-C$ else abort.
3. Compute the endorsement $e \leftarrow E_{\mathcal{E}}(sk_{\mathcal{E}},M,i)$ on M_i.
4. Compute the MAC $p \leftarrow S_{\mathcal{M}}(sk_{\mathcal{M}},h(m))$ on the hash of m.
5. Send a request to the aggregator $\mathcal{Q}$ and receive the triple $(t',c,S) \leftarrow \mathcal{Q}.put(r_i^0,h(m)\|p)$.
6. Verify that $t < t' \leq t+L$; then query the digest $d \leftarrow \mathcal{P}.get(t')$ for time t'; then verify that $V_C(d,c,(r_i^0,D_{\mathcal{A}}(S))) = 1$ and $h(m)\|p \in S$.
7. Verify that each element in S has a valid MAC according to $V_{\mathcal{M}}(sk_{\mathcal{M}},\cdot)$.
8. Output $(e,M_i,i,\ell,r_i^0,r_i^\ell,c,q,z,p)$ as the signature on m, where $\ell = t'-t$, $q = D_{\mathcal{A}}(S)$, $z = P_{\mathcal{A}}(pk_{\mathcal{Q}},S,h(m)\|p)$, where $pk_{\mathcal{Q}}$ is the public key of the aggregator $\mathcal{Q}$.

Verification. To verify the message m against the signature (e,w,i,ℓ,a,r,c,q,z,p) with respect to the public key $(pk_{\mathcal{E}},h,C,E,L)$:

1. If $V_{\mathcal{E}}(pk_{\mathcal{E}},e,w,i) \neq 1$ then return 0.
2. If $\ell \leq 0$ or $\ell > L$ then return 0.
3. Set $t \leftarrow C+i$ and $t' \leftarrow t+\ell$; if $t < C$ or $t \geq C+E$ then return 0.
4. If $w_0 \neq h(a)$ or $w_\ell \neq h(r)$ then return 0.
5. Query the digest $d \leftarrow \mathcal{P}.get(t')$ for time t'; if $V_C(d,c,(a,q)) \neq 1$ then return 0.
6. If $V_{\mathcal{A}}(q,z,h(m)\|p) \neq 1$ then return 0.
7. Otherwise, return 1.

Fig. 2 shows the data flows of the signing process.

Note that the timestamping server and BLT+L aggregators do not need to accumulate any data for long-term storage and are only expected to receive and compress the user queries into respective digests which is then sent for a long-term storage to a publisher. The computational complexity of the user is analyzed in Sec. 5.

4.1 Security Analysis

Theorem 4.1. Let $\mathcal{X}$ be a black-box timestamping service based on tag-commitment scheme $\mathcal{C}$ and publishing its digests via the publisher $\mathcal{P}$. Let $\mathcal{Q}$ be a black-box aggregator based on accumulator scheme $\mathcal{A}$ timestamping accumulators ($\mathcal{A}$ digests) via $\mathcal{X}$. If the highest tolerable network delay is L and the lifespan of the key pair is E then for any adversary the probability of breaking the existential unforgeability of BLT+L in asynchronous setting does not exceed

$$p_C + p_{\mathcal{E}} + p_{\mathcal{A}} + p_{\mathcal{M}} + p_{\mathcal{H}} + E \cdot (L+1) \cdot p'_{\mathcal{H}},$$

where p_C is the probability of breaking the binding property of $\mathcal{C}$, $p_{\mathcal{E}}$ the probability of breaking collision-resistance of the endorsement scheme $\mathcal{E}$, $p_{\mathcal{A}}$ the probability of breaking collision-resistance of $\mathcal{A}$, $p_{\mathcal{M}}$ the probability of breaking unforgeability of the MAC scheme $\mathcal{M}$, $p_{\mathcal{H}}$ the probability of breaking collision-resistance of the hash-function family $\mathcal{H}$, and $p'_{\mathcal{H}}$ the probability of breaking preimage resistance of $\mathcal{H}$.

Proof. Let the public key be $(pk_{\mathcal{E}}, h, C, E, L)$ and the secret key $(sk_{\mathcal{M}}, sk_{\mathcal{E}}, r)$. Let A be an adversary who presents a BLT+L forgery $(e, w, i, \ell, a, r, c, q, z, p)$ for a fresh message m. Let $t \leftarrow C + i$ and $t' \leftarrow t + \ell$ when we analyze the following cases:

- If $w \neq M_i$ then A must have broken collision resistance of the endorsement scheme $\mathcal{E}$.
- If $w = M_i$, but $a \neq r_t^0$ or $r \neq r_t^\ell$, then A broke collision resistance of the hash function family $\mathcal{H}$.
- The remaining analysis is for the case when $w = M_i$, $a = r_t^0$, and $r = r_t^\ell$.
  - Let $Q = \{m_1, \ldots, m_k\}$ be the set of messages that the adversary submitted to the signing oracle at time t and $Z \subseteq Q$ be the subset of Q for which the signing oracle successfully finalized the signatures corresponding to time t'.
  - If Q is empty then A broke preimage resistance of h, as he observed $h(r_t^0)$ and found the preimage a.
  - If Z is empty then also A broke preimage resistance of h, as he observed $h(r_t^\ell)$ and found the preimage r.
  - Otherwise, pick a message $m_j \in Z$ and let S_j be the list of messages with which the aggregator $\mathcal{Q}$ replied to the *put* query of the signing oracle.
  - If $q \neq D_{\mathcal{A}}(S_j)$ then A broke the binding property of the tag-commitment scheme $\mathcal{C}$.
  - If $q = D_{\mathcal{A}}(S_j)$ and $h(m) \| p \notin S_j$ then A broke collision resistance of $\mathcal{A}$.
  - Otherwise, $q = D_{\mathcal{A}}(S_j)$ and $h(m) \| p \in S_j$ and therefore p is a valid MAC for $h(m)$, since we assumed that signature was finalized for m_j.
  - If the MAC p on $h(m)$ was not generated by the signing oracle then A broke unforgeability of $\mathcal{M}$.
  - Otherwise, $h(m)$ must equal to $h(m')$ for some m' submitted to the signing oracle and therefore m and m' is a collision of h.

In cases when preimage resistance of h is attacked, the adversary has the choice of $E \cdot (L+1)$ hashes to invert. Therefore, the unforgeability of BLT+L in these cases is upper-bounded by preimage resistance of h scaled up by $E \cdot (L+1)$. □

4.2 Discussion

It is important to understand the roles and limitations of servers in BLT+L. More specifically, the publisher $\mathcal{P}$ is a trusted party with expensive resources who can publish no more than a single (short) digest per round. The timestamper $\mathcal{X}$ is a black box which produces a digest from a large number of queries of independent clients. The server $\mathcal{Q}$ aggregates tagged requests into accumulators per tag. We assume that $(\mathcal{P}, \mathcal{X})$ is a widely available timestamping service. The aggregator $\mathcal{Q}$ can be an independent server, or run on the client device, or be coupled with the timestamping service. In the following, we highlight the main traits of the scheme:

- The standard approach of producing legally binding signatures is to produce "cryptographic signature" first and timestamp it afterwards. The BLT+L framework offers an alternative design where the final signature consists of an endorsement and a timestamp. We expect this design to induce more efficient signatures since the security property of endorsements is weaker than existential unforgeability of cryptographic signatures and the cryptographic properties of the timestamping are taken into account and contribute to existential unforgeability of the resulting BLT+L signature.
- All the cryptographic primitives used in BLT+L (including timestamping services and endorsement schemes) can be constructed from secure hash functions. In this case, we can upper-bound the probability of existential forgery by the probability of breaking collision resistance or preimage resistance of the hash function.
- We conjecture that BLT+L is secure in the post-quantum setting, since our proofs do not rely on rewinding or state-copying techniques.
- In the signing procedure, we let the user query the current time from the adversary (timestamping service $\mathcal{X}$) to highlight that the received value need not be correct. This means that in practice, the user can rely on her local clock without worrying whether it might be inaccurate.
- Observe that BLT+L key generation phase seems to require a user to produce a list r containing $E \cdot (L+1)$ random values and generate an endosement key pair for a list M which contains hashes of elements of r. However, if the endorsement key-generation algorithm $G_{\mathcal{E}}(M)$ does not require the values of M to be evaluated eagerly (as is the case with the Goldreich-Merkle scheme) then the elements of r and M can remain unevaluated until they are needed to form a signature. This allows us to defer key generation costs. Moreover, in practice, truly random values are replaced with PRF-generated pseudo-random values. In this case the security of the signature scheme drops by the indistinguishability of the used PRF.
- Computing endorsements is the main computational effort of the BLT+L scheme. However,

since the value of the endorsement does not depend on the message being signed, it is possible to share it across all the signatures produced during the same L-round interval. Our preliminary analysis suggests that this will allow us to asynchronously produce hundreds of signatures per second on devices that can only compute 20–30 thousand hashes per second and have 500 KB/s download bandwidth (e.g., smartphones). This compares favorably with the current state-of-the-art hash-based signature scheme SPHINCS, which is offline, but requires hundreds of thousands of hash operations per signature (Bernstein et al., 2015).

- We also note that, although our endorsements use one-time signatures, these signatures are not applied to signed documents directly. As a result, the number of signatures that can be securely produced *per round of timestamping service* is limited only by security parameters of used MAC, accumulator, and tag-commitment schemes (i.e., virtually unlimited for practical purposes).
- In our security proof we assumed that there is only one publisher. If we allow a user to choose a publisher at a signing time then for the signature to remain secure the “time values” of the publishers must be synchronized.
- The security proof for BLT+L relies on the fact that at most one digest of the tag-commitment scheme can be associated with a time slot by the publisher. If this property is violated then the signature becomes forgeable by an adversarial timestamping service by creating a second digest that the user is unaware of. This can be solved by timestamping a message binding of the form $h(m\|r_t^1),\dots,h(m\|r_t^L)\|p$ instead of $h(m)\|p$. In this case BLT+L is secure even when the publisher can associate multiple digests with the same time value, but the security will depend on non-malleability of the hash function h.
- In BLT+L, an unpredictable token r_t^0 is associated with every time t, and used as a tag for the timestamping requests. The value r_t^0 is kept secret until time t to prevent unauthorized parties from submitting requests that would be aggregated together with the user’s genuine requests.

5 PERFORMANCE ANALYSIS

In this section we analyze the performance parameters of BLT+L and propose an optimization strategy that accounts for availability of computational resources.

Both the user’s computational effort and the size of the resulting signatures are dominated by the underlying endorsement scheme, which in turn is determined by the coloring function. For a perfect binary tree of height H, the number of different coloring functions is 2^{2^H-1}. For uniform complexity of endorsing (and thus BLT+L signing), we are only interested in coloring functions that assign the same type to all nodes on the same tree level:

Definition 5.1. The coloring function $f:\mathbb{N}\to\{\mathsf{G},\mathsf{M}\}$ is *level-uniform* iff

$$f(i)=f(j)\text{ for all }\lfloor\log_2(i)\rfloor=\lfloor\log_2(j)\rfloor .$$

Every level-uniform coloring function for a tree of height H can be represented by a string of the form $T_1^{x_1}T_2^{x_2}\dots T_n^{x_n}$, where $\Sigma_{i=0}^n x_i = H$, and $T_i\in\{\mathsf{G},\mathsf{M}\}$. In this representation, G^x (M^x) stands for Goldreich (Merkle) tree of height x.

Previously we already mentioned that in practical deployments the key pair generation of endorsement and BLT+L schemes should be done by using secure PRFs. In particular, this allows us to generate the OTS key pairs for Goldreich nodes on-demand instead of having to pre-generate and store large amounts of confidential data.

However, with this approach the topmost OTS key pairs of Goldreich nodes of the endorsement tree are regenerated on every run of the endorsing algorithm. We noticed that for hash-based OTS schemes, it is computationally cheaper to generate secret keys, but more expensive to derive the corresponding public keys. Therefore, we propose a further optimization where the most frequently used public keys of the topmost Goldreich nodes are *pre-computed* during the initialization phase and cached for later use. In the following analysis, the cache is computed once during the initialization phase, it consists of OTS public keys only, and is never updated. The maximal available memory for *cache* is given as a parameter.

To sum it up, if we know the BLT+L parameters (E and L) and the available computational resources, we can optimize BLT+L performance by finding the fittest level-uniform coloring function for the Goldreich-Merkle endorsement scheme. We consider the following performance metrics:

1. *Initialization Complexity:* is the total computational effort needed to produce a BLT+L public key and the cache of OTS public keys up to the maximal allowed size.
2. *Signing Complexity:* is the total computational effort of a signer needed to produce a signature. The signer’s computations are dominated by endorsement generation.

3. *Verification Complexity:* is the total computational effort needed to verify a signature. It is composed of endorsement verification, accumulator inclusion proof verification, and timestamp verification.
4. *Signature Size:* is the sum of the sizes of the endorsement, accumulator inclusion proof, and timestamp.

5.1 Case Study

We base our analysis on the performance parameters of hash-based KSI timestamping service which aggregates user queries into Merkle trees and has round duration equal to one second. The size of a KSI timestamp is 1–2 KB and its verification takes less than 40 hash function evaluations (Buldas et al., 2013).

We consider BLT+L key pairs with lifespans of 10 years which induce lists of size $E = 3.154 \cdot 10^8$ and, therefore, the height of the endorsement tree for our use case is $H = \log_2(E) = 29$. For tree of that height there are 2^{29} level-uniform coloring functions. This is a reasonably small space and we can use exhaustive search to find the optimal function for each parameter set.

We further assume 256-bit hash functions and implement the Goldreich nodes of the endorsement tree using the Winternitz hash-based OTS (Merkle, 1987). The used variant of Winternitz OTS requires 532 hashing operations for PRF-based key generation and signing, and generates 4256-byte signatures. We also replace the Winternitz OTS public keys with their hash values as the OTS verification recreates the public key value. This significantly reduces the signature size, but adds only one extra hashing operation per OTS generation and verification.

Table 1 presents several sets of BLT+L performance parameters. The initialization, signing, and verification times are given in thousands of hashing operations required on the client side, beause hashing performance may differ radically between devices, from a few hundred operations per second for smart-cards to several million for "full-size" CPUs. The cache and signature sizes are given in kilobytes. The initialization consists of generation of the BLT+L public key and pre-computation of the cache of OTS public keys for the endorsement scheme. The scenarios listed highlight the adaptability of the scheme—key generation and signing time can be traded for signature size and verification time in very wide ranges.

The signing complexity is shown for a single thread of execution. In case of n parallel signing threads, the per-signature signing complexity is increased by $D(n)/n + P(n)$ operations, where $D(n)$ and $P(n)$ are the number of operations needed to compute the digest and an inclusion proof of an accumulator of size n, respectively. The verification complexity increases by $V(n)$, the number of operations needed to verify the inclusion proof.

6 RELATED WORK

Merkle proposed a signature scheme based on a hash tree which aggregates a large number of inputs into a single hash value so that any of the N inputs can be linked to it with a proof consisting of $\log_2(N)$ hash values. This allowed many instances of a one-time signature scheme to be combined into a many-time scheme by placing one-time public keys into the leaves of the tree (Merkle, 1979). This scheme is impractical if the number of one-time key pairs is large, as the whole tree has to be built during key generation.

To defer the key generation costs, Goldreich proposed a scheme based on authentication trees where each node contains a one-time key pair. The key pairs in leaf nodes are used to sign messages while the key pair in a non-leaf node is used to authenticate (sign) the public keys of both of its children. The public key of this scheme is the public key of the root node and the secret key consists of secret keys of all tree nodes, possibly pseudorandomly generated from a secret seed value (Goldreich, 2004).

Since these schemes are based on one-time signatures, it is crucial that no leaf node is ever used more than once. One way to prevent this would be to introduce state that keeps track of spent one-time keys. However, management of such state is a significant security risk (McGrew et al., 2016). To remove state management, the one-time key to be used could be selected by time, but this would prevent asynchronous parallel signing and would limit signing to only one message per time unit. For Merkle signatures, this would also waste keys when signing is performed infrequently. An alternative for Goldreich signatures is to set the tree height equal to the length of the messages (or their hashes) and use the message itself (or its hash) as the leaf index. Unfortunately, this would result in impractically large signatures. For example, with 256-bit hashes and the efficient Winternitz OTS, Goldreich signatures would be larger than 1 MB.

Yet another approach would be to use random selection in a sufficiently large tree so that the probability of using the same leaf twice becomes negligible. SPHINCS is a family of stateless signature schemes based on this idea that produces significantly smaller signatures (e.g. 41 KB for SPHINCS-256). This is achieved by reducing the tree height, combined with

Table 1: BLT+L performance metrics for some parameter sets. We assume 10-year lifespan of the BLT+L key pair, network lag up to 3 seconds, Winternitz OTS, and 256-bit hash function. Computation times are given in thousands of hash function evaluations, cache and signature sizes in kilobytes.

	Init. time	Cache size	Sig. time	Ver. time	Sig. size	Coloring function
1.	1	0	35	3.0	36	$M^1G^1M^1G^1M^1G^1M^2G^1M^2G^1M^2G^1M^2G^1M^2G^1M^8$
2.	555	33	9	5.0	55	$M^{10}G^{13}M^6$
3.	1 200	70	14	2.0	26	$M^{11}G^1M^2G^1M^2G^1M^2G^1M^2G^1M^2G^1M^2$
4.	4 500	270	80	0.6	10	$M^{13}G^1M^7G^1M^7$
5.	9 000	524	215	0.4	5	$M^{14}G^1M^{14}$

the use of Merkle trees (Bernstein et al., 2015). To mitigate the risk of reusing a one-time key in such reduced tree, the key pairs in the leaf nodes are from a few-time sheme (Reyzin and Reyzin, 2002).

Although our work does not rely on few-time signatures, it is also based on combining the ideas of Goldreich and Merkle. It differs from SPHINCS in the tree size (that is much smaller in BLT+L) and in the leaf node selection mechanism. Another major difference is the property that BLT+L key pairs have a validity time after which the key can no longer be used for signing. The signing and verification time and signature size for SPHINCS are mainly dependent on the number of messages that can be signed without significant loss of security. In contrast, the same values for BLT+L mainly depend on the length of the validity time of the key pair.

The motivational work for our paper are the recent results by Buldas et al. They proposed two schemes, BLT-TB and BLT-OT, both based on combining a backdating resistant timestamping service with a forward resistant tag system, and both producing small signatures. As the stateless BLT-TB variant uses large Merkle trees, it suffers from slow key generation (approximately 96 million hashing operations to generate a key that would allow signing once per second for a year, for example), and as it uses time to select the one-time key to use, it is wasteful if signing is performed infrequently. The scheme does not specifically address the lag in the timestamping process and the authors suggest several parallel signing attempts for different time values as a remedy (Buldas et al., 2017b). The BLT-OT scheme addresses these issues (in particular, reducing the key generation costs for personal signing devices that are used only occasionally), but at the cost of introducing stateful client-side computations and thus making it more complicated to handle asynchronous calls (Buldas et al., 2019).

Our work is most related to the BLT variant proposed by Firsov et al. which uses the internal one-time signature values in the Goldreich authentication tree as tag values. The scheme has both efficient key generation and signing time without introducing a client side state, but it does not address the network lag and produces considerably larger signatures compared to BLT-TB and BLT-OT (Firsov et al., 2021).

BLT+L addresses several shortcomings of the preceding variants of BLT: the network lag is directly and efficiently addressed, and parallel asynchronous signing is supported. Our scheme is parametrized so that signature size and verification time can be traded for key generation and signing time depending on the priorities of the use case.

A property that the BLT+L scheme shares with other members of the BLT family is that each BLT+L signature is inherently timestamped. This is a benefit in the context of legally binding signature frameworks such as eIDAS (European Commission, 2014), but the timestamping service call is an additional expense compared to other signature schemes in contexts where proof of signing time is not needed. As a result, we expect BLT+L to be practical for signing legally binding documents and transactions by humans as opposed to, for example, authenticating SSH sessions.

7 CONCLUSIONS AND FUTURE WORK

We have presented a new digital signature scheme based on combining timestamping with an endorsement scheme, both of which can be constructed from one-way and collision-resistant hash functions.

To account for the server-assisted nature of the signature scheme, our security analysis is conducted in a realistic black-box model where only the publisher component of the timestamping service is trusted. Also, the design of the signature takes into account concurrent requests and network delays.

Finally, we showed that the signature scheme is efficient and allows trading of key generation and signing time for signature size and verification time. For example, our analysis suggests that BLT+L can produce hundreds of 26–55-KB signatures per second on devices with limited computational resources. Alternatively, BLT+L can produce 5-KB signatures on devices with sufficient computational power.

As the next steps, we plan to use theorem provers (e.g., EasyCrypt) to formally verify our correctness and security claims. Also, we want to address the security of BLT+L in the post-quantum setting.

ACKNOWLEDGEMENTS

This work was partially supported by the ESF-funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and the Estonian Centre of Excellence in ICT (EXCITE) research measure TAR16013 (TK148) (1.09.2016–1.03.2023).

REFERENCES

Asokan, Tsudik, G., and Waidner, M. (1997). Server-supported signatures. *Journal of Computer Security*, 5(1):91–108.

Benaloh, J. and de Mare, M. (1991). Efficient broadcast time-stamping. Technical report, Clarkson University.

Bernstein, D. J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., and Wilcox-O'Hearn, Z. (2015). SPHINCS: Practical stateless hash-based signatures. In *EUROCRYPT 2015, Proceedings, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer.

Bicakci, K. and Baykal, N. (2004). Server assisted signatures revisited. In *CT-RSA 2004, Proceedings*, volume 2964 of *LNCS*, pages 143–156. Springer.

Buldas, A., Firsov, D., Laanoja, R., Lakk, H., and Truu, A. (2019). A new approach to constructing digital signature schemes. In Attrapadung, N. and Yagi, T., editors, *Advances in Information and Computer Security*, pages 363–373, Cham. Springer International Publishing.

Buldas, A., Kalu, A., Laud, P., and Oruaas, M. (2017a). Server-supported RSA signatures for mobile devices. In *ESORICS 2017, Proceedings, Part I*, volume 10492 of *LNCS*, pages 315–333. Springer.

Buldas, A., Kroonmaa, A., and Laanoja, R. (2013). Keyless signatures' infrastructure: How to build global distributed hash-trees. In *NordSec 2013, Proceedings*, volume 8208 of *LNCS*, pages 313–320. Springer.

Buldas, A., Laanoja, R., and Truu, A. (2017b). A server-assisted hash-based signature scheme. In *NordSec 2017, Proceedings*, volume 10674 of *LNCS*, pages 3–17. Springer.

Buldas, A., Laanoja, R., and Truu, A. (2018). A blockchain-assisted hash-based signature scheme. In *NordSec 2018, Proceedings*, volume 11252 of *LNCS*, pages 138–153. Springer.

Buldas, A. and Laur, S. (2006). Do broken hash functions affect the security of time-stamping schemes? In Zhou, J., Yung, M., and Bao, F., editors, *Applied Cryptography and Network Security*, pages 50–65, Berlin, Heidelberg. Springer Berlin Heidelberg.

Buldas, A. and Laur, S. (2007). Knowledge-binding commitments with applications in time-stamping. In *PKC 2007, Proceedings*, volume 4450 of *LNCS*, pages 150–165. Springer.

Camenisch, J., Lehmann, A., Neven, G., and Samelin, K. (2016). Virtual smart cards: How to sign with a password and a server. In *SCN 2016, Proceedings*, volume 9841 of *LNCS*, pages 353–371. Springer.

European Commission (2014). Regulation no 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/EC (eIDAS regulation). *Official Journal of the European Union*, L 257:73–114.

Firsov, D., Lakk, H., and Truu, A. (2021). Verified multiple-time signature scheme from one-time signatures and timestamping. Cryptology ePrint Archive, Report 2021/528.

Goldreich, O. (2004). *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press.

Goyal, V. (2004). More efficient server assisted one time signatures. Cryptology ePrint Archive, Report 2004/135.

Haber, S. and Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111.

McGrew, D. A., Kampanakis, P., Fluhrer, S. R., Gazdag, S.-L., Butin, D., and Buchmann, J. A. (2016). State management for hash-based signatures. In *SSR 2016, Proceedings*, volume 10074 of *LNCS*, pages 244–260. Springer.

Merkle, R. C. (1979). *Secrecy, Authentication and Public Key Systems*. PhD thesis, Stanford University.

Merkle, R. C. (1987). A digital signature based on a conventional encryption function. In *CRYPTO'87, Proceedings*, volume 293 of *LNCS*, pages 369–378. Springer.

Reyzin, L. and Reyzin, N. (2002). Better than BiBa: Short one-time signatures with fast signing and verifying. In *ACISP 2002, Proceedings*, volume 2384 of *LNCS*, pages 144–153. Springer.